Altova RaptorXML Server 2021

ユーザーマニュアル

Altova RaptorXML Server 2021
ユーザーマニュアル

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.

公開日: 2015-2021

(C) 2015-2021 Altova GmbH

目次

1 はじめに 8

2 RaptorXML Server についての説明 9

2.1 エディションとインターフェイス 10
2.2 システムの必要条件 14
2.3 機能 15
2.4 サポートされる仕様 17

3 RaptorXML のセットアップ 19

3.1 プロセッサーコアとライセンス 20
3.2 Windows でのセットアップ 21
3.2.1 Windows へのインストール 21
3.2.2 Windows でのライセンス 24
3.3 Linux でのセットアップ 28
3.3.1 Linux へのインストール 28
3.3.2 Linux でのライセンス 31
3.3.3 環境についてのメモ 33
3.4 Mac OS X でのセットアップ 35
3.4.1 macOS へのインストール 35
3.4.2 macOS でのライセンス 37
3.4.3 環境についてのメモ 39
3.5 XML カタログ 40
3.6 カタログの仕組み 41
3.7 RaptorXML Server 内のカタログ構造 43
3.8 カタログのカスタム化 45
3.9 Windows システムロケーションのための変数 47
3.10 グローバルリソース 49
3.11 セキュリティの問題 51

4 コマンドライン インターフェイス (CLI) 52
4.1 XML、DTD、XSD 検証コマンド 53
4.1.1 valxml-withdtd (xml) 53
4.1.2 valxml-withxsd (xsi) 56
4.1.3 valdtd (dtd) 62
4.1.4 valxsd (xsd) 66
4.2 整形式チェック コマンド 72
4.2.1 wfxml 72
4.2.2 wfdtd 75
4.2.3 wfany 78
4.3 XQuery コマンド 82
4.3.1 xquery 82
4.3.2 xqueryupdate 88
4.3.3 valxquery 95
4.3.4 valxqueryupdate 101
4.4 XSLT コマンド 107
4.4.1 xslt 107
4.4.2 valxslt 114
4.5 JSON/Avro コマンド 120
4.5.1 avroextractschema 120
4.5.2 avrotojson 123
4.5.3 jsontoavro 125
4.5.4 valavro (avro) 129
4.5.5 valavrojson (avrojson) 132
4.5.6 valavroschema (avroschema) 135
4.5.7 valjsonschema (jsonschema) 138
4.5.8 valjson (json) 141
4.5.9 wfjson 145
4.6 XML 署名コマンド 149
4.6.1 xmlsignature-sign 149
4.6.2 xmlsignature-verify 153
4.6.3 xmlsignature-update 156
4.6.4 xmlsignature-remove 159

4.7 一般的なコマンド ... 161
4.7.1 valany ... 161
4.7.2 script ... 162
4.7.3 help ... 163
4.8 ローカライズコマンド ... 164
4.8.1 exportresourcestrings ... 164
4.8.2 setdeflang ... 166
4.9 ライセンスコマンド ... 167
4.9.1 licenseserver ... 167
4.9.2 assignlicense (Windows only) ... 168
4.9.3 verifylicense (Windows only) ... 169
4.10 管理コマンド ... 171
4.10.1 install ... 172
4.10.2 uninstall ... 172
4.10.3 start ... 173
4.10.4 setdeflang ... 174
4.10.5 licenseserver ... 175
4.10.6 assignlicense (Windows only) ... 176
4.10.7 verifylicense (Windows only) ... 177
4.10.8 createconfig ... 178
4.10.9 exportresourcestrings ... 179
4.10.10 debug ... 180
4.10.11 help ... 181
4.10.12 version ... 182
4.11 オプション ... 184
4.11.1 カタログ、グローバルリソース、ZIP ファイル ... 184
4.11.2 メッセージ、エラー、ヘルプ、タイムアウト、バージョン ... 185
4.11.3 処理 ... 186
4.11.4 XML ... 187
4.11.5 XSD ... 188
4.11.6 XQuery ... 190
4.11.7 XSLT ... 192
4.11.8 JSON/Avro ... 194
4.11.9 XML 署名 ... 194

5 サーバー API: HTTP、COM/.NET、Java 198

5.1 HTTP REST クライアントインターフェイス 200
- 5.1.1 サーバーセットアップ 201
- 5.1.2 クライアントリクエスト 211

5.2 COM/.NET API 230
- 5.2.1 COM インターフェイス 230
- 5.2.2 COM サンプル: VBScript 230
- 5.2.3 .NET インターフェイス 232
- 5.2.4 .NET Example: C# 233
- 5.2.5 .NET サンプル: Visual Basic .NET 236

5.3 Java API 238
- 5.3.1 インターフェイスの概要 238
- 5.3.2 サンプル Java プロジェクト 239

5.4 サーバー API レファレンス 241
- 5.4.1 インターフェイス/クラス 241
- 5.4.2 列挙 292

6 エンジン API: Python と .NET 303

6.1 ライセンス 305

6.2 Python API 306
- 6.2.1 Python API バージョン 307
- 6.2.2 Python パッケージとしての RaptorXML Server 309
- 6.2.3 サーバー側の Python スクリプトのデバッグ 311
- 6.2.4 Visual Studio コード内のスクリプトの Python デバッグ 311
- 6.2.5 よくある質問 313

6.3 .NET Framework API 314

7 追加情報 315

7.1 終了コード 316

7.2 スキーマのロケーションのヒント 317

8 エンジン情報 318
8.1 XSLT および XQuery エンジンに関する情報 319
8.1.1 XSLT 1.0 319
8.1.2 XSLT 2.0 319
8.1.3 XSLT 3.0 321
8.1.4 XQuery 1.0 321
8.1.5 XQuery 3.1 324
8.2 XSLT と XPath/XQuery 関数 325
8.2.1 Altova 拡張関数 326
8.2.2 その他の拡張関数 411

インデックス 428

1 はじめに

Altova RaptorXML Server（以下ではRaptorXML と省略されます）は、Altova の第 3 世代の、XML およびXBRL* 超高速プロセッサーです。RaptorXML は、最新の標準とパラレルコンピューティング環境を最適化するために開発されました。クロスプラットフォームに対する高い対応性のためにデザインされたエンジンは、今日至る所で使用されているマルチコアのコンピューターが高速でXML および XBRL データを処理することができます。

メモ XBRL 処理は、RaptorXML+XBRL Server のみで使用可能で、RaptorXML Server では使用することができません。

このドキュメント

このドキュメントは、Altova W eb サイトでも使用できるアプリケーションで提供されています。 Chrome ブラウザーは、ドキュメントがローカルで開かれる場合、目次（TOC）ペイン内の展開を防く制限があることに注意してください。ドキュメントがウェブブラウザーで開かれる場合のみ、Chrome 機能内での目次は正確に機能します。

このドキュメントは以下のセクションに整理されています:

- RaptorXML について（このセクション） 19
- RaptorXML のセットアップ 19
- コマンドライン インターフェイス 52
- サーバーAPI: HTTP、COM/.NET、Java 198
- エンジン API: Python と.NET 303
- 追加情報 315
- エンジン情報 318

Altova W eb サイト: XML 検証サーバー、XML バリデーター

最終更新日: 2021 年 02 月 24 日

 (C) 2015-2021 Altova GmbH

2 RaptorXML Server についての説明

エディションとオペレーティング システム

RaptorXML には エディションが 2 つあり、各エディションは異なる条件に合わせて使用することができます。これらのエディションはエディションとインターフェイス 10 のセクションで説明されています。RaptorXML は、Windows、Linux、および macOS に対して使用することができます。システムサポートの詳細に関しては、システムの必要条件 14 のセクションを参照してください。

機能とサポートされる仕様

RaptorXML は、それぞれ広い範囲にわたるパワフルなオプションを搭載した XML 検証、XSLT 変換、および XQuery 実行を提供します。使用できる機能の大まかなリストの機能に関しては、機能 15 のセクションを参照してください。サポートされる仕様 17 のセクションは、RaptorXML の準拠する詳しい仕様のリストを提供します。詳細に関しては、Altova Web サイトの RaptorXML ページに移動してください。

2.1 エディションとインターフェイス

エディション

RaptorXML には以下のエディションがあります:

- *RaptorXML Server* は、XML、XML スキーマ、XSLT、XPath、XQuery などをサポートする高速の XML 処理 エンジンです。
- *RaptorXML+XBRL Server* は、RaptorXML Server の全ての機能と、追加処理機能および、XBRL 一連の標準の検証をサポートします。

サポートさえる仕様 17 のリストのためにここ 17 を参照してください。

インターフェイス

RaptorXML は、以下のインターフェイスからアクセスすることができます:

- *コマンドライン インターフェイス (CLI):* RaptorXML の Windows、Linux、および macOS インストールために使用することができます。
- *HTTP REST クライアント インターフェイス:* RaptorXML HTTP インターフェイスを使用します。
- *COM/.NET サーバーインターフェイス (Windows):* RaptorXML (i) COM/.NET API と (ii) HTTP REST インターフェイスを使用します。
- *Java サーバーインターフェイス (Windows、Linux、macOS):* RaptorXML (i) Java API と (ii) HTTP REST インターフェイスを使用します。
- *Altova XMLSpy インターフェイス:* RaptorXML に Altova XMLSpy ユーザーインターフェイスからアクセスすることができます。
- *Python エンジン インターフェイス:* (i) 環境内の RaptorXML Python ホイール (ii) Python スクリプト内の RaptorXML の Python API を使用します。このようにして、RaptorXML 機能は、Python スクリプト内でサードパーティ Python パッケージと共に使用することができます。
- *.NET エンジン インターフェイス (Windows):* RaptorXML 機能を使用する個別の .NET アプリケーションを作成するために (i) RaptorXML DLL と (ii) RaptorXML の .NET API を使用します。

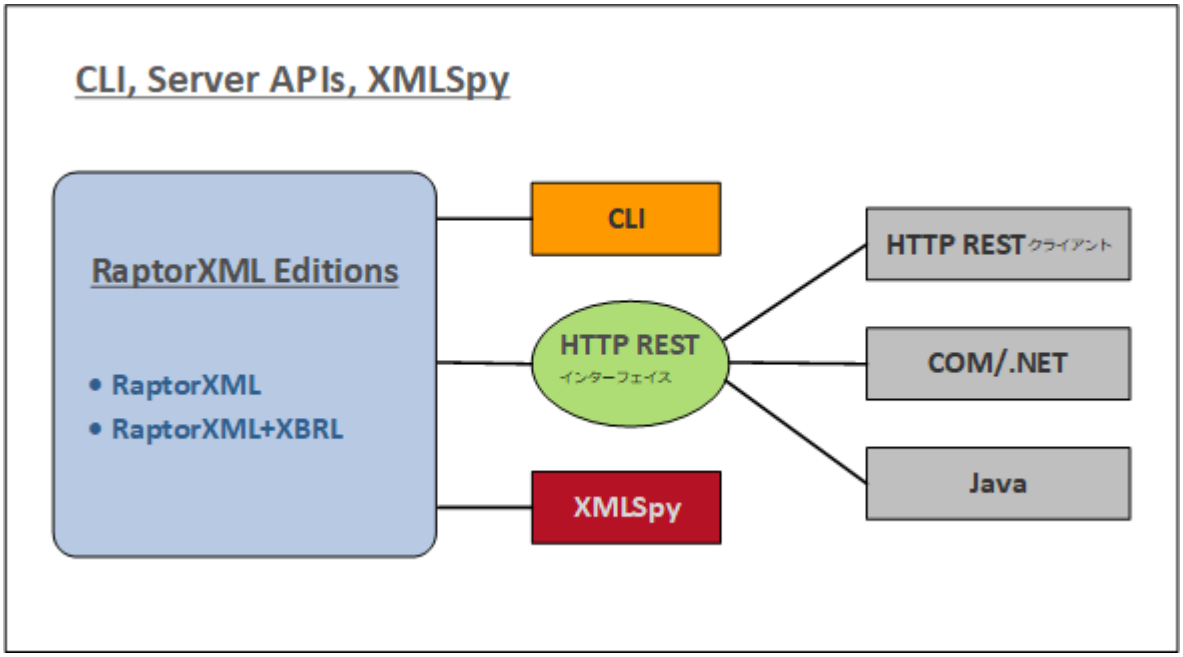
7 つのインターフェイスを以下の 4 つのグループに分類することができます

- コマンドライン インターフェイス (CLI) 52
- サーバー API: HTTP、COM/.NET、Java 198
- エンジン API: Python と .NET 303
- Altova XMLSpy 12

CLI、サーバー API と Altova XMLSpy、

CLI、Server API、Altova XMLSpy を介したアクセスは下の図で示されています。

RaptorXML Server はサーバーに検証ジョブを送信する HTTP REST インターフェイスを定義します。HTTP REST インターフェイスに直接、または、ハイレベルの COM/.NET と Java Server API を使用してアクセスすることができます。これらの API は、HTTP REST リクエストの作成と送信を管理する COM/.NET と Java クラスの簡単な使用を提供します。更に、リモートの RaptorXML Server. 上のジョブの検証を実行するように Altova XMLSpy を構成することもできます。



コマンドライン インターフェイス (CLI)

- RaptorXML は、インストールされているマシン上でライセンスを供与することができ、コマンドラインを介してこのインスタンスにアクセスすることができます。
- W indows、Linux、および macOS にインストールすることができます。
- XML, XML スキーマ、XML 署名、XQuery および XSLT ドキュメントの検証、および、処理のためのコマンドライン使用法 52 を提供します。
- Python 3.7 は RaptorXML 内でバンドルされており、Python スクリプトが `--script` オプションと呼び出されると使用されます。

HTTP REST クライアント インターフェイス

- RaptorXML は、インストールされているマシン上でライセンスを供与することができ、HTTP REST クライアント インターフェイス 200 を介してこのインスタンスにアクセスすることができます。
- クライアント リクエストは JSON フォーマットで作成されます。各リクエストは、出力ファイルが保存されるサーバー上のジョブディレクトリに割り当てられます。クライアントに対するサーバーレスポンスは、ジョブに関する関連したすべての情報を含みます。
- Python 3.7 は RaptorXML 内でバンドルされており、Python スクリプトが `--script` オプションと呼び出されると使用されます。

COM/.NET インターフェイス

- W indows のみで使用することができます。
- RaptorXML は COM サーバーオブジェクトとしてインストール時に自動的に登録され、COM の呼び出しのためにプログラミングのサポートをするアプリケーションとスクリプト言語から呼び出すことができます。
- RaptorXML は、インストールされているマシン上でライセンスを供与することができます。
- .NET インターフェイスは COM インターフェイスのラッパーとしてビルドされます。

- RaptorXML の COM/.NET Server API 230 は、RaptorXML 機能にアクセスするための COM/.NET スクリプト言語内で使用することのできるオブジェクトを提供します。
- Python 3.7 は RaptorXML 内でバンドルされており、Python スクリプトが `--script` オプションと呼び出されると使用されます。

Java インターフェイス

- RaptorXML は、インストールされているマシン上でライセンス供与を行うことができ、このインスタンスは Java プログラムによりアクセスすることができます。
- RaptorXML 機能は Java Server API 238 内の Java プログラム内で Java クラスとして使用することができます。
- Python 3.7 は RaptorXML 内でバンドルされており、Python スクリプトが `--script` オプションと呼び出されると使用されます。

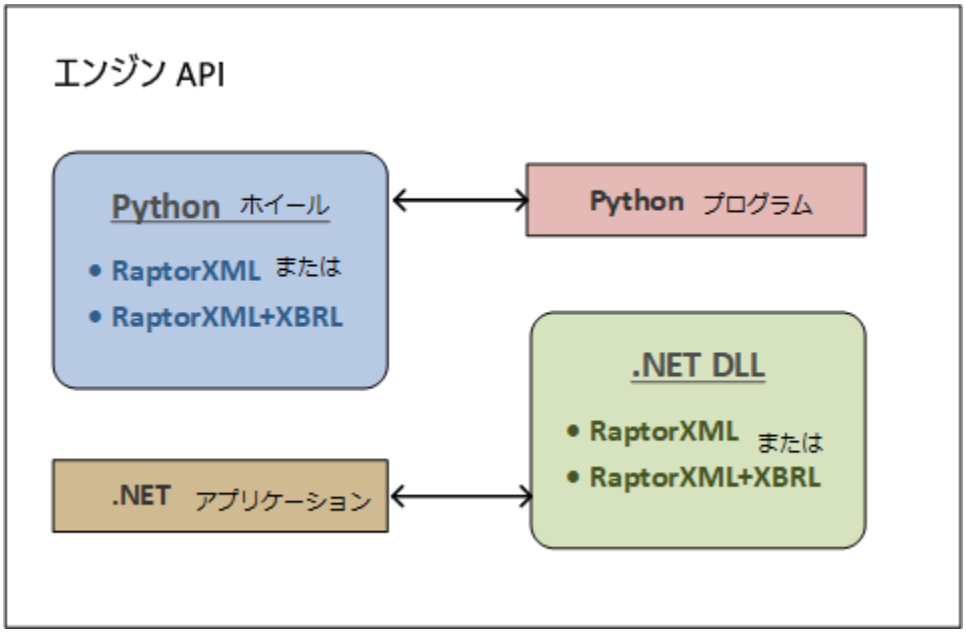
Altova XMLSpy

- ライセンス供与済みの Altova XMLSpy がインストールされており、XMLSpy が RaptorXML Server にネットワーク上でアクセスできる場合、XML ドキュメントを検証するために XMLSpy GUI 内から RaptorXML Server を使用し、XSLT と XQuery 変換を実行ことができます。
- XMLSpy プロジェクトフォルダー内でアクティブなドキュメントまたはすべてのドキュメントを検証することができます。
- XMLSpy GUI のメッセージウィンドウ内で検証結果を表示することができます。
- XMLSpy 内で、(i) ドキュメントを検証 (ii) XMLSpy のエンジン、または、RaptorXML Server を使用して XSLT/XQuery 変換の実行することができます。
- Raptor を使用する利点の 1 つは、検証オプションの大きな範囲を使用して個別の検証を構成できることです。更に、Raptor オプションを XMLSpy 内の「構成」として保管し、特定の Raptor 検証のために定義済みの構成を選択できることです。Raptor の使用は大きなデータコレクションを検証する際に役に立ちます。

エンジン API

エンジン API 303 は Python プログラムと .NET アプリケーションにより使用されるものと Python ホイール内に含まれる RaptorXML のサーバー API および .NET DLL とは異なります (*下の図参照*)。これらのプログラム/アプリケーションは、RaptorXML 機能にアクセスするために、それぞれ Raptor Python API 306 と Raptor .NET API 314 を使用する必要があります。

メモ Python API 306 と .NET API 314 により提供される機能は、例えば、ドキュメントの読み取り、および、データの操作など、CLI または サーバー API により提供される機能よりも多種です。



Python インターフェイス

- RaptorXML は Python 3.7 環境内にインストールすることのできる Python ホイール パッケージ内で使用することができます。
- RaptorXML の Python API 306 からのオブジェクトを使用するように Python プログラムを書くことができます。この API は CLI 内で更に使用することのできる機能を適用し、Python 環境内で適用されるサード パーティライブラリにより提供される機能と共に使用することができます。
- RaptorXML の Python ホイールを介して RaptorXML 機能が呼び出されると、コマンドが実行される前にマシン上の有効な RaptorXML ライセンスに対するチェックが実行されます。

.NET インターフェイス

- RaptorXML は .NET Framework をサポートするアプリケーション内に埋め込むことのできる DLL 内で使用することができます。API に関する上は、.NET Framework API 314 のセクションを参照してください。
- RaptorXML の .NET API 314 は、RaptorXML へのアクセスを提供します。RaptorXML CLI 内で使用することのできる機能よりも多くの機能を使用することができます。
- .NET アプリケーションを介して RaptorXML 機能が呼び出されると、マシン上で有効な RaptorXML ライセンスに対するチェックが実行されます。

2.2 システムの必要条件

RaptorXML Server は以下のオペレーティング システムでサポートされています:

- *Windows*

  プラットフォーム更新済みの Windows 7 SP1、Windows 8、Windows 10

- *Windows Server*

  プラットフォーム更新済みの Windows Server 2008 R2 SP1 または以降

- *Linux*

  - CentOS 7 または以降
  - RedHat 7 または以降
  - Debian 8 または以降
  - Ubuntu 16.04 LTS または以降

  次のライブラリはアプリケーションをインストールし実行するために必要とされるライブラリです。下のパッケージが使用中 Linux のマシンで使用できない場合、`yum`（または、適用できる場合、`apt-get` を）コマンドを実行してインストールしてください。

サーバー	CentOS, RedHat	Debian	Ubuntu
LicenseServer	krb5-libs	libgssapi-krb5-2	libgssapi-krb5-2
RaptorXML Server	qt5-qtbase-gui, krb5-libs	libqt5gui5, libgssapi-krb5-2	libqt5gui5, libgssapi-krb5-2

  メモ Altova チャート機能を使用する場合、チャートが正確に表示されるように少なくともフォントの 1 つがシステム上にインストールされている必要があります。インストールされているフォントをリストするには、例えば Fontconfig ライブラリの `fc-list` コマンドを使用してください。

- *macOS*

  macOS 10.13 または以降

RaptorXML 32 ビットおよび 64 ビットのマシンで使用することができます。具体的には、これらは、x86 および amd64（x86-64）命令セットをベースにしたコアです: Intel Core i5、i7、XEON E5。RaptorXML を COM インターフェイス介して使用するには、ユーザーは、アプリケーションを登録して、対応するアプリケーションとおよびまたはスクリプトを実行する、COM インターフェイスを使用する特権を有する必要があります。

 (C) 2015-2021 Altova GmbH

2.3 機能

RaptorXML は、以下にリストされる機能を提供します。機能の多くはコマンドライン使用およびCOM インターフェイス使用と共通です。大きな違いの1つは、Windows 上のCOM インターフェイス使用は、(XML、DTD、XML スキーマ、XSLT、またはXQuery ファイルを参照するかわりに)アプリケーションまたはスクリプトコードからドキュメントをテキスト文字から構成できることです。

XML 検証

- 与えられたXML ドキュメントを内部または外部 DTD またはXML スキーマに対して検証。
- XML、DTD、XML スキーマ、XSLT、およびXQuery ドキュメントの整形式のチェック。

XSLT 変換

- 与えられたXSLT 1.0、2.0、または3.0 ドキュメントを使用してXML を変換します。
- XML とXSLT ドキュメントは、(URL を介して)ファイルとして、またはCOM 使用の場合はテキスト文字列として提供されます。
- 出力はファイルで返されます(名前の付けられた場所で)または、COM を使用の場合はテキスト文字列として返されます。
- XSLT パラメーターは、コマンドラインを介して、またはCOM インターフェイスを介して提供されます。
- Altova 拡張関数、XBRL、Java および.NET 拡張関数により特別な処理を有効化することができます。これにより、例えば、出力ドキュメント内でのチャートとバーコード機能などを作成することができます。

XQuery 実行

- XQuery 1.0 と3.0 ドキュメントの実行.
- XQuery とXML ドキュメントは、(URL を介して)ファイルとして、またはCOM を使用の場合はテキスト文字列として返されます。
- 出力はファイルで返されます(名前の付けられた場所で)または、COM を使用の場合はテキスト文字列として返されます。
- External XQuery 変数は、コマンドラインを介して、またはCOM インターフェイスを介して提供されます。
- シリアル化オプションは以下を含みます:出力エンコード、出力メソッド、(すなわち出力がXML、XHTML、HTML、またはテキスト)、宣言の省略、およびインデント。

JSON とAvro 検証/変換

- JSON スキーマとAvro スキーマドキュメントの検証
- JSON インスタンスのJSON スキーマとAvro スキーマに対しての検証
- Avro バイナリの検証
- JSON フォーマットのAvro バイナリのAvro スキーマとAvro データの変換
- Avro JSON データのAvro バイナリへの変換

ハイパーパフォーマンス機能

- 超高速パフォーマンスコード最適化

  - ネイティブの命令セットの実装
  - 32 ビット または64 ビット バージョン
- 超低用量メモリフットプリント
  - XML 情報セットの超コンパクトなインメモリ表記
  - ストリーミングインスタンス検証
- クロスプラットフォーム機能
- マルチ-CPU/マルチ-コア/パラレル コンピューティングのための高度にスケール化することのできるコード
- デザインによるパラレルロード、検証、および処理

開発者のための機能

- 優れたエラー報告機能
- Windows サーバーモードとUnix daemon モード (コマンド-ライン オプションを介して)
- スクリプトのためのPython 3.x インタプリタ
- Python パッケージ内のRaptorXML 機能により機能をPython ライブラリの機能としてインポートできます。
- .NET Framework API により、基となるXML データモデルにアクセスすることができます。
- Windows プラットフォーム上のCOM API
- あらゆる箇所でのJava API
- XPath 拡張関数、Java、.NET、など。
- ストリーミング シリアル化
- REST 検証 API 付きビルトインHTTP サーバー

詳細に関しては、サポートされる仕様 17 およびAltova Web サイト のセクションを参照してください。

2.4 サポートされる仕様

RaptorXML は以下の仕様をサポートします。

W3C 勧告

Web サイト: WWW コンソーシアム(World Wide Web Consortium) (W3C)

- Extensible Markup Language (XML) 1.0 (第 5 エディション)
- Extensible Markup Language (XML) 1.1 (第 2 エディション)
- Namespaces in XML 1.0 (第 3 エディション)
- Namespaces in XML 1.1 (第 2 エディション)
- XML Information Set (第 2 エディション)
- XML Base (第 2 エディション)
- XML Inclusions (XInclude) Version 1.0 (第 2 エディション)
- 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 (第 2 エディション)
- XSL Transformations (XSLT) Version 2.0
- XQuery 1.0: An XML Query Language (第 2 エディション)
- XQuery 1.0 and XPath 2.0 Functions and Operators (第 2 エディション)
- XSLT 2.0 and XQuery 1.0 Serialization (第 2 エディション)
- 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 作業ドラフト & 候補勧告

Web サイト: World Wide Web Consortium (W3C)

- XSL 変Transformations換 (XSLT) Version 3.0 (サブセット)
- XQuery 3.1: An XML Query Language
- XPath and XQuery Functions and Operators 3.1
- XQuery Update Facility 3.0
- XSLT とXQuery Serialization 3.1

OASIS 標準

Web サイト: OASIS 標準

- XML Catalogs V 1.1 - OASIS Standard V1.1

JSON/Avro 標準

Web サイト: JSON スキーマとApache Avro

- JSON Schema Draft 4
- JSON Schema Draft 6
- JSON Schema Draft 7
- Apache Avro™ 1.8.1

(C) 2015-2021 Altova GmbH

3 RaptorXML のセットアップ

このセクションは、RaptorXML Server の設定方法を説明します。

- プロセッサーコアとライセンス 20 に関する情報
- Windows 21 、Linux 28 および macOS 35 システムへの RaptorXML のインストールとライセンス。
- XML カタログ 40 の使用方法。
- Altova グローバルリソース 49 との作業方法。
- RaptorXML に関連したセキュリティの問題 51

RaptorXML には、ポータブル性とモジュール性を強化する XML カタログ 40 および Altova グローバルリソース 49 をサポートする特別のオプションがあります。

メモ セキュリティに関する問題と重要なセキュリティソリューションの設定方法はセキュリティの問題 51 のセクションで説明されています。

3.1 プロセッサーコアとライセンス

Altova サーバー製品へのライセンス供与は（ロジカルなコアの数量ではなく）製品マシン上で使用することのできる **物理的なプロセッサーのコア** の数量をベースにしています。例えば、デュアルコアプロセッサーには、クアッドコアプロセッサー4コア、ヘキサコア6コアなどの2つのコアが存在します。製品にライセンス供与されたコア数は、物理または仮想マシン上に関わらず、サーバーマシン上で使用することのできるコアの数量と同数、または、それ以上である必要があります。例えば、サーバーが8コア持つ場合、8コアライセンスを購入する必要があります。コアの数量を満たすためにライセンスを組み合わせることもできます。ですから、2つの4コアライセンスを使用して、8コアライセンスの代わりに8コアを達成することもできます。

処理する容量は少ないが、CPU コアの大きなコンピューターサーバーを使用している場合、少ない数量のコアに割り当てられている仮想マシンを作成し、その数量のライセンスを購入することもできます。このようなデプロイの場合、もちろん、コンピューターのすべてのコアを使用する場合に比べこのようなデプロイの処理スピードは遅いです。

メモ 各 Altova サーバー製品ライセンスは、使用されていないライセンス供与能力が存在する場合でも、Altova サーバー製品がインストールされているマシンで、一度に1つのクライアントマシンのためにのみ使用することができます。例えば、10 コアライセンスが6 CPU コアを持つマシンのために使用されている場合、残りのライセンス許与能力の4 コアは、他のクライアントマシンのために同時に使用することはできません。

単一スレッドの実行

Altova サーバー製品が単一スレッドの実行を許可すると *単一スレッドの実行* を使用できるようになります。この場合、Altova サーバー製品ライセンスはライセンスプール内で1コアのみ使用することができます。複数のコアを持つマシンはこの1コアライセンスに割り当てることができます。このような場合、その製品を作動するマシンは単一コアで作動します。（マルチコアでは使用可能な）複数スレッドを使用できないため処理はこのために遅くなります。製品はそのマシン上で単一スレッドモードで実行されます。

マルチコアマシンに単一コアライセンスを割り当てるには、その製品のためにLicenseServer 内で *「単一スレッド実行に制限する」* チェックボックスを選択してください。

必要なコアの見積もり

サーバーが扱うことのできるデータの量と処理回数を影響する多くの外部要素が存在します。（例えば ハードウェア、CPU 上の現在のロード、サーバー上で作動する他のアプリケーションのメモリの割り当てなどが挙げられます）。パフォーマンスを正確に測定するために、データの量、条件、およびビジネスの環境に近い状態でアプリケーションをテストしてください。

3.2 Windows でのセットアップ

このセクションはWindows システムへのRaptorXML Server のインストール21とライセンス24について説明します。

Windows へのインストール21
- システム必要条件21
- インストールRaptorXML Server21
- Altova LicenseServer21
- LicenseServer バージョン21
- トライアルライセンス21
- アプリケーション フォルダーの場所21

Windows でのライセンス24
- ServiceController の開始24
- LicenseServer の開始24
- RaptorXML Serverの開始24
- RaptorXML Serverの登録24
- ライセンスの割り当て24

メモ バージョン2021 以降からRaptorXML Server の32 ビットバージョンは64 ビットバージョンにインストールすることはできません、または64 ビットバージョンを32 ビットバージョンにインストールすることはできません。(i) 新しいバージョンをインストールする前に古いバージョンを削除する、または(ii) 古いインストールと同じビットバージョンを新規のバージョンに更新する必要があります。

3.2.1 Windows へのインストール

RaptorXML Server はWindows システムへインストールすることができます。インストールとセットアップについては以下で説明されます。

- システム必要条件
  - *Windows*

    プラットフォーム更新済みのWindows 7 SP1、Windows 8、Windows 10

  - *Windows Server*

    プラットフォーム更新済みのWindows Server 2008 R2 SP1 または以降

- RaptorXML Serverのインストール

  RaptorXML Server のWindows システムへのインストールは以下の手順で行います:

  - 個別のスタンドアロンサーバー製品はRaptorXML Server と称されます。RaptorXML Server をインストールするには、RaptorXML Server のインストーラーをダウンロードして実行してください。スクリーンの手順に従ってください。
  - FlowForce Server インストールパッケージの一部として。FlowForce Server パッケージの一部としてRaptorXML Server をインストールするには、FlowForce Server インストーラーをダウンロードして実行します。RaptorXML Server のインストールオプションを確認して、スクリーンの手順に従ってください。

RaptorXML Server と FlowForce Server のインストーラーは Altova ダウンロードセンター (http://www.altova.com/ja/download.html) で入手でき、必要な登録手続きとともに製品をインストールすることができます。

インストール後、実行可能な RaptorXML Server はデフォルトで以下で見つけることができます:

```
<ProgramFilesFolder>\Altova\RaptorXMLServer2021\bin\RaptorXML.exe
```

COM インターフェイスおよび .NET 環境を介して RaptorXML Server を使用するために必要な登録はインストーラにより行われます。この手順には、実行可能な RaptorXML Server の COM サーバーオブジェクトとしての登録、`Altova.RaptorXML.dll` ファイルを .NET リファレンスライブラリへの追加などが含まれます。

Windows Server でのインストール

Windows Server Core は GUI 機能の多くを使用しない最小の Windows インストールです。RaptorXML Server を Windows Server Core マシンに以下のようにインストールすることができます:

1. Altova Web サイトから RaptorXML Server インストーラー実行可能ファイルをダウンロードします。このファイルは **`RaptorXMLServer<version>.exe`**. という名前です。サーバープラットフォーム (32-bit または 64-bit) に一致する実行可能ファイルが選択されていることを確認してください。
2. (Windows Server Core マシンではなく) 標準の Windows マシンではコマンド **`RaptorXMLServer<version>.exe /u`** を実行します。これは **`.msi`** ファイルをインストーラー実行ファイルと同じフォルダーで解凍します。
3. 解凍されていない **`.msi`** ファイルを Windows Server Core マシンにコピーします。
4. RaptorXML Server の前のバージョンを更新している場合、次のステップを継続する前に RaptorXML Server をシャットダウンしてください。
5. コマンド **`msiexec /i RaptorXMLServer.msi`** を実行してインストールのために **`.msi`** ファイルを使用します。Windows Server Core でのインストールがこれにより開始されます。

`.msi` ファイルを同じ場所に保管します。インストールを後にアンインストール、修復、または変更する場合にこれらのファイルが必要になります。

インストールの戻り値をテストするには、以下のスクリプトを実行します。戻りコードは `%errorlevel%` 環境関数の中にあります。0 の値を持つリターンコードは成功を意味します。

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

戻りコードとインストールプロセルのログを持つサイレントインストールの場合、以下を実行します:

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

インストールを変更する場合、以下を実行します:

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

インストールを修復する場合、以下を実行します:

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

RaptorXML Server をアンインストールするには、以下を実行します:

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

RaptorXML Server をサイレントにアンインストールし、ログファイル内に詳細な結果をレポートするには、以下を実行します:

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

メモ Windows Server Core では RaptorXML Server のチャート 機能を使用することはできません。

▼ Altova LicenseServer

- RaptorXML Server が作動するためには、ネットワークの Altova LicenseServer からライセンスを供与される必要があります。
- RaptorXML Server または FlowForce Server を Windows システムにインストールするには、 Altova LicenseServer を RaptorXML Server または FlowForce Server をダウンロードしてインストールするオプションがあります。
- Altova LicenseServer が既にネットワークにインストールされている場合、新しいバージョンの Altova LicenseServer が必要ない限り、再度インストール必要はありません。（*次のポイント，LicenseServer のバージョンを参照してください。*）
- RaptorXML Server または FlowForce Server のインストール中、適宜 Altova LicenseServer のインストールのオプションをチェックしてください。
  a. Altova LicenseServer がまだインストールされていない場合、デフォルトの設定を保持してください。ウィザードが最新バージョンの Altova LicenseServer をウィザードを作動しているコンピューターにインストールします。
  b. Altova LicenseServer がまだインストールされていなく、他のコンピューターに Altova LicenseServer をインストールする場合、「**使用中のコンピューターに Altova LicenseServer をインストールする**」チェックボックスをクリアして、「**後で登録**」を選択します。この場合、 Altova LicenseServer のインストールと RaptorXML Server の登録を個別に行う必要があります。
  c. Altova LicenseServer が既に使用中のコンピューターにインストールされているが、インストールウィザードにより示されているバージョンよりも低い場合は、デフォルトの設定を保持してください。この場合、インストールウィザードが自動的に LicenseServer バージョンをダイアログボックスに表示されたバージョンにアップグレードします。既存の登録およびライセンス情報はアップグレードの後も保存されます。
  d. Altova LicenseServer が既に使用中のコンピューター、または、ネットワークにインストールされており、ウィザードに示されるバージョンと同じの場合は、以下を行ってください：
     i. 「**使用中のコンピューターに Altova LicenseServer をインストールする**」チェックボックスをクリックしてクリアします。
     ii. 「**この製品を登録する**」から RaptorXML Server を登録する Altova LicenseServer インスタンスを選択、または、「**後で登録**」を選択してください。LicenseServer への関連を無視し、RaptorXML Server のインストールを継続する場合、「**後で登録**」を常に選択することができます。

RaptorXML Server の Altova LicenseServer への登録とライセンス供与の詳細に関しては、セクション Windows でのライセンス 24 を参照してください。

▼ LicenseServer バージョン

- Altova サーバー製品はインストールされた RaptorXML Server バージョンに適切な LicenseServer のバージョン、または、LicenseServer の最新のバージョンが必要です。
- RaptorXML Server の特定のバージョンに適切な LicenseServer のバージョンが RaptorXML Server のインストール中表示されます。
- LicenseServer の新しいバージョンをインストールする前に、古いバージョンはアンインストールされる必要があります。LicenseServer インストーラーは古いバージョンを検出すると自動的に行います。
- LicenseServer バージョンは下位互換性があります。RaptorXML Server の全ての古いバージョンと作動します。
- RaptorXML Server の新しいバージョンをインストールする場合、そして、インストールされている LicenseServer バージョンが適切な LicenseServer バージョンより古い場合、Altova Web サイトから利用可能な最新バージョンをインストールします。
- LicenseServer をアンインストールする際、古いバージョンの LicenseServer のすべての登録とライセンス情報はサーバーマシーンのデータベースに保存されます。このデータは新しいバージョンがインストールされる際、自動的に新しいバージョンにインポートされます。
- 現在インストールされている LicenseServer のバージョン番号は LicenseServer 構成ページ（全てのタブ）の下部にあります。

現在のバージョン: 3.6

▼ トライアルライセンス

インストール中、30 日間のRaptorXML Server のトライアルライセンスをリクエストすることができます。リクエストを送信すると、登録した電子メールアドレスにトライアルライセンスが送信されます。

▼ アプリケーション フォルダの場所

アプリケーションは以下のフォルダーにインストールされます:

Windows 7、8、10	`C:\Program Files\Altova\`
64 ビット OS 上の32 ビットバージョン	`C:\Program Files (x86)\Altova\`

3.2.2 Windows でのライセンス

RaptorXML Server は作動するためにAltova LicenseServer にライセンスされている必要があります。ライセンス供与は以下の2 つのステップから構成されています:

1. **RaptorXML Server** にLicenseServer を登録します。RaptorXML Server から登録することができます。
2. RaptorXML Server のライセンスを割り当てます。ローカルマシンまたは、ネットワーク上のマシンにインストールされる必要のある LicenseServer からライセンスを割り当てることができます。

実行するステップは以下に手短に説明されています。詳細は、Altova W eb サイト のユーザーマニュアルを参照してください。

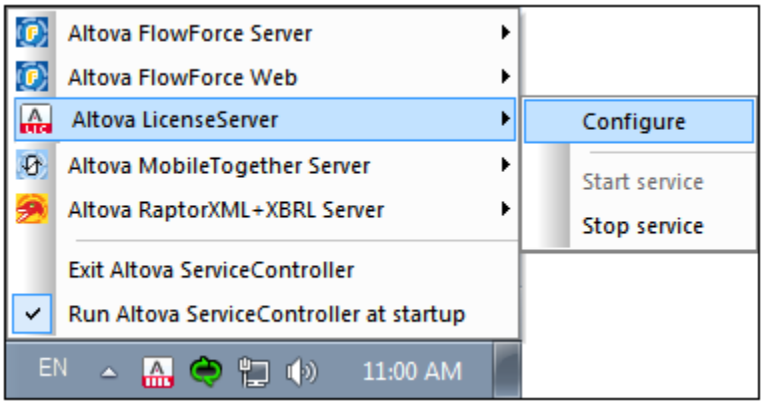
▼ ServiceController の開始

Altova ServiceController はAltova LicenseServer とAltova RaptorXML Server を開始するために必須です。

Altova ServiceController (略してServiceController) はW indows システム上で Altova サービスを便利に開始、停止、構成できるアプリケーションです。

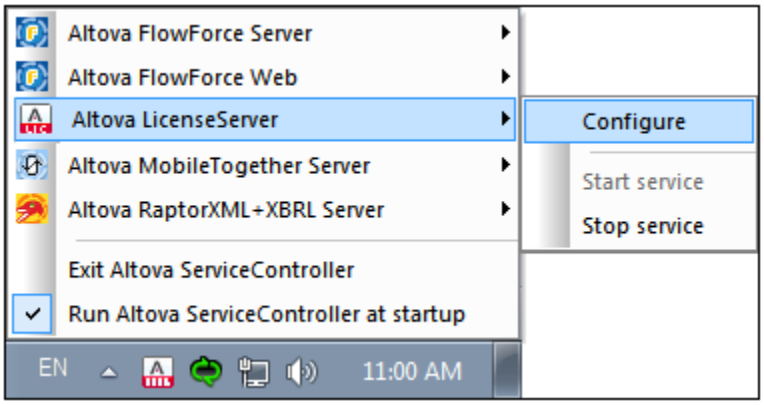
ServiceController はAltova LicenseServer とおよび *サービスとしてインストールされる Altova サーバー製品* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server).**スタート | Altova LicenseServer | Altova ServiceController** をクリックして開始されます。(このコマンドは「**開始**」メニューフォルダーでも使用することができます *サービスとしてインストールされる Altova サーバー製品*(FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server)。)ServiceController が開始した後、システムトレイからアクセスすることができます。(*下部スクリーンショット*)。

システムログイン時にServiceController の自動開始を指定するには、システムトレイの**ServiceController** アイコンをクリックして **ServiceController** メニューを表示します(*下部スクリーンショット*)。**スタートアップ時に** Altova ServiceController を作動する(Run Altova ServiceController at Startup) コマンドに切り替えます。(このコマンドはデフォルトで切り替えられています。)ServiceController を終了するには、システムトレイの**ServiceController** アイコンをクリックして、表示されるメニューから **Altova ServiceController** の終了(Exit Altova ServiceController) をクリックします(*下部スクリーンショット参照*)。



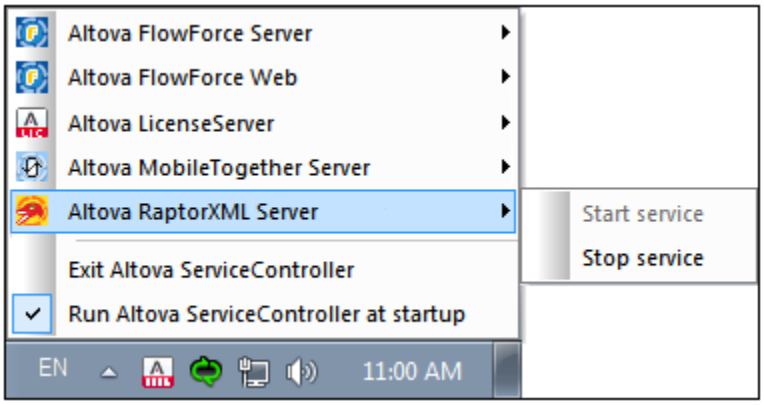
LicenseServer の開始

LicenseServer を開始するにはシステムトレイの **[ServiceController]** アイコンをクリックします。メニューの **[Altova LicenseServer]** にポイントすると、(*下部スクリーンショット参照*) がポップアップします。サブメニューから **[Start Service]** (サービスの開始) を選択します。LicenseServer が既に作動している場合、*Start Service* オプションは無効化されます。



RaptorXML Server の開始

RaptorXML Server を開始するには、システムトレイの **[ServiceController]** アイコンをクリックします。メニューの **Altova RaptorXML Server** にポイントすると サブメニュー(*下部スクリーンショット参照*) がポップアップします。RaptorXML Server サブメニューから **[Start Service]** (サービスの開始) を選択します。RaptorXML Server が既に作動している場合、*Start Service* オプションは無効化されます。



メモ RaptorXML Server がシングルスレッド実行を作動するようにライセンスされている場合、(通常これは、使用中のマシンがマルチコアにもかかわらず、ライセンスがシングルコアの場合を指します)、1度にRaptorXML Server の1つのインスタンスのみをサーバー、または、コマンドラインから使用することができます。これは、シングルコアライセンスが自動的に開始される最初のインスタンスに自動的に割り当てられるからです。2番目のインスタンスは、最初のインスタンスが作動を停止するまで開始することはできません。

- サービスが既に作動しており、RaptorXML Server をコマンドラインから使用する場合、コマンドラインを使用する前にサービスを停止する必要があります。
- RaptorXML Server をサービスから開始する場合、コマンドラインアクションが実行中でないことを確認してください。それ以外の場合、サービスを開始することはできません。

RaptorXML Server の登録

RaptorXML Server をコマンドラインインターフェイスから登録する場合、`licenseserver` コマンドを使用します:

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

例えば、`localhost` がLicenseServer がインストールされているサーバーの名前の場合、:

```
RaptorXML licenseserver localhost
```

RaptorXML Server がFlowForce Server のインストールの一部としてインストールされる場合、 LicenseServer へのFlowForce Server の登録の際、自動的にRaptorXML Server も登録されます。

FlowForce Server ドキュメンテーションにFlowForce Server の登録の手順が説明されています。原則: (i) Altova FlowForce W eb をサービスとしてServiceController から開始します(*前述のポイント参照*); (ii) パスワードを入力してセットアップページにアクセスします; (iii) LicenseServer 名を選択または[**LicenseServer により登録**] をクリックします。

登録に成功した後、LicenseServer 構成ページのServer Management(サーバーの管理)タブへ移動してRaptorXML Server へライセンスを割り当てます。

ライセンスの割り当て

RaptorXML Server の登録に成功した後、LicenseServer の構成ページのClient Management(クライアントの管理タブ)にリストされます。移動してRaptorXML Server にライセンスの割り当てを行います。

Altova サーバー製品へのライセンスは製品マシンで使用可能なプロセッサコアの数をベースにしています。例えば、デュアルコア プロセッサはコアが2 つ、クアッド コア プロセッサはコアが4 つ、ヘキサコア プロセッサはコアが6 つ等々。特定のサーバーマシン上の製品にライセンスされたコアの数は、物理または仮想マシンで、サーバーで使用可能なコア数よりも多くまたは同数である必要があります。例

えば、サーバーが8 コア(オクタルコア プロセッサ)の場合、少なくとも 8-コア ライセンスを購入する必要があります。また、ライセンスを合計してコア数を満たすこともできます。2 つの4-コアライセンスは、8-コアライセンスの代わりにオクタルコア サーバーで使用できます。

大きいCPU コアを持つコンピューターサーバーを使用し、少量を処理する場合、少ないコアを割り当てる仮想マシンを作成し、その数のライセンスを購入することもできます。このようなデプロイは、もちろん、サーバーの全ての利用可能なコアが利用されている場合に比べ、処理スピードが落ちます。

メモ 各 Altova サーバー製品のライセンスは、使用されていないライセンス容量があっても、1 度に1 つのクライアントマシンにだけしか使用することができません。例えば10-コアライセンスが6 CPU コアのクライアント マシンに使用される場合、残りの 4 コア ライセンスは他のマシンで同時に使用することができません。

3.3 Linux でのセットアップ

このセクションはLinux システム(Debian、Ubuntu、CentOS、RedHat) へのRaptorXML Server のインストール28 とライセンス31 について説明します。

Linux へのインストール28
- システム必要条件28
- root ユーザーのメモ28
- Altova サーバー製品の古いバージョンのアンインストール28
- Linux パッケージのダウンロード28
- RaptorXML Serverのインストール28
- Altova LicenseServer28
- LicenseServer のバージョン28
- トライアルライセンス28

Linux でのライセンス31
- root ユーザーのメモ31
- LicenseServer の開始31
- RaptorXML Server の開始31
- RaptorXML Server の登録31
- ライセンスの割り当て32

3.3.1 Linux へのインストール

RaptorXML Server のLinux システムへのインストールは利用可能です。インストールとセットアップについては以下で説明されます。

- システムの必要条件
  - *Linux*
    - CentOS 7 または以降
    - RedHat 7 または以降
    - Debian 8 または以降
    - Ubuntu 16.04 LTS または以降

    次のライブラリはアプリケーションをインストールし実行するために必要とされるライブラリです。下のパッケージが使用中 Linux のマシンで使用できない場合、`yum` (または、適用できる場合、`apt-get` を) コマンドを実行してインストールしてください。

サーバー	CentOS, RedHat	Debian	Ubuntu
LicenseServer	krb5-libs	libgssapi-krb5-2	libgssapi-krb5-2
RaptorXML Server	qt5-qtbase-gui, krb5-libs	libqt5gui5, libgssapi-krb5-2	libqt5gui5, libgssapi-krb5-2

メモ Altova チャート機能を使用する場合、チャートが正確に表示されるように少なくともフォントの 1 つがシステム上にインストールされている必要があります。インストールされているフォントをリストするには、例えばFontconfig ライブラリの`fc-list` コマンドを使用してください。

FlowForce Server 統合

RaptorXML Server を FlowForce Server と共にインストールする場合、FlowForce Server を最初にインストールすることが奨励されます。それ以外の場合、RaptorXML Server と FlowForce Server の両方をインストールした後、以下のコマンドを実行してください：

```
cp /opt/Altova/RaptorXMLServer2021/etc/*.tool /opt/Altova/FlowForceServer2021/tools
```

このコマンドは **.tool** ファイルを RaptorXML Server の **/etc** ディレクトリから FlowForce Server **/tools** ディレクトリにコピーします。FlowForce Server は **.tool** ファイルを必要とし このファイルは RaptorXML Server 実行可能ファイルへの パスを含みます。FlowForce Server を RaptorXML Server をインストールする前にインストールする場合このコマンドを実行する必要はありません。

root ユーザーに関するメモ

RaptorXML Server をインストールするには、管理者(root) 特権を有する必要があります。ですから、インストールは root ユーザーとして実行されなければなりません。`root` としてログインする場合、`sudo` キーワードを以下のコマンドから省略することができます。

Altova サーバー製品の古い バージョンアンインストールする方法

前の バージョンをアンインストールする場合、以下の手順を踏んでください。Linux コマンドラインインターフェイス（CLI）で、Altova サーバー製品がインストールされているか、以下のコマンドで確認できます:

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

RaptorXML Server がインストールされていない場合、以下の *RaptorXML Serverのインストール* で説明されている手順を踏んでください。

RaptorXML Server が既にインストールされており、RaptorXML Server の新しい バージョンをインストールしたい場合、古い バージョンを以下のコマンドでアンインストールしてください：

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

Altova LicenseServer の古い バージョンをアンインストールする場合、以下のコマンドで行ってください：

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

Debian と Ubuntu システムでは、RaptorXML Server がインストール後にインストール済みの製品としてリストする場合があります。この場合、**purge** コマンドを実行して RaptorXML Server をリストからクリアします。**purge** コマンドを 上記の **remove** コマンドの *instead* 代わりに使用することもできます。

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

Linux パッケージのダウンロード

以下の RaptorXML Server の Linux システムへの パッケージは Altova Web サイト で使用可能です。

配布	パッケージ拡張子
Debian	`.deb`

配布	パッケージ拡張子
Ubuntu	.deb
CentOS	.rpm
RedHat	.rpm

Linux パッケージのダウンロード後、Linux システムに直接コピーしてください。 RaptorXML Server を作動するためにはAltova LicenseServer が必要なため、Altova W eb サイト から、RaptorXML Server をダウンロードと同時にLicenseServer をダウンロードしてください。

RaptorXML Server のインストール

ターミナルウィンドウで、Linux パッケージをコピーしたいディレクトリに切り替えてください。例えば、MyAltova と称されるユーザーディレクトリにコピーしたとします、(例えば /home/User ディレクトリに存在するとします)、ディレクトリを以下のようにスイッチします:

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

以下のコマンドを使用して RaptorXML Server インストールする

```
[Debian]:   sudo dpkg --install raptorxmlserver-2021-debian.deb
[Ubuntu]:   sudo dpkg --install raptorxmlserver-2021-ubuntu.deb
[CentOS]:   sudo rpm -ivh raptorxmlserver-2021-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh raptorxmlserver-2021-1.x86_64.rpm
```

RaptorXML Server パッケージはフォルダにインストールされます:

```
/opt/Altova/RaptorXMLServer2021
```

Altova LicenseServer

RaptorXML Serverを含むAltova サーバー製品を作動するには、サーバー製品はネットワークでAltova LicenseServer を介して、ライセンスを与えられなければなりません。

Linux システムでは、Altova LicenseServer は個別にインストールされる必要があります。 Altova W eb サイト から LicenseServer をダウンロードして、パッケージを Linux システムのディレクトリにコピーします。 RaptorXML Server 同様インストールします(*前のステップ参照*)。

```
[Debian]:   sudo dpkg --install licenseserver-3.6-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-3.6-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-3.6-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-3.6-1.x86_64.rpm
```

LicenseServer パッケージは以下にインストールされます:

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

RaptorXML Server をAltova LicenseServer で登録して、ライセンスを与えることに関しての詳細は、セクション Linux でのライセンス 31 を参照してください。

LicenseServer バージョン

- Altova サーバー製品はインストールされたRaptorXML Server バージョンに適切な LicenseServer のバージョン、または、LicenseServer の最新のバージョンが必要です。
- RaptorXML Server の特定のバージョンに適切な LicenseServer のバージョンがRaptorXML Serverのインストー

ル中表示されます。

- LicenseServer の新しいバージョンをインストールする前に、古いバージョンはアンインストールされる必要があります。LicenseServer インストーラーは古いバージョンを検出すると自動的に行います。
- LicenseServer バージョンは下位互換性があります。RaptorXML Server の全ての古いバージョンと作動します。
- RaptorXML Server の新しいバージョンをインストールする場合、そして、インストールされている LicenseServer バージョンが適切な LicenseServer バージョンより古い場合、Altova Web サイトから利用可能な最新バージョンをインストールします。
- LicenseServer をアンインストールする際、古いバージョンの LicenseServer のすべての登録とライセンス情報はサーバーマシーンのデータベースに保存されます。このデータは新しいバージョンがインストールされる際、自動的に新しいバージョンにインポートされます。
- 現在インストールされている LicenseServer のバージョン番号は LicenseServer 構成ページ（全てのタブ）の下部にあります。

現在のバージョン: 3.6

3.3.2 Linux でのライセンス

RaptorXML Server は作動するために Altova LicenseServer にライセンスされている必要があります。ライセンス供与は以下の 2 つのステップから構成されています:

1. **RaptorXML Server** に LicenseServer を登録します。RaptorXML Server から登録することができます。
2. RaptorXML Server のライセンスを割り当てます。ローカルマシンまたは、ネットワーク上のマシンにインストールされる必要のある LicenseServer からライセンスを割り当てることができます。

実行するステップは以下に手短に説明されています。詳細は、Altova Web サイト のユーザーマニュアルを参照してください。

▼root ユーザーに関するメモ

RaptorXML Server をインストールするには、管理者（root）特権を有する必要があります。ですから、インストールは ユーザーとして実行されなければなりません。`root` としてログインする場合 `sudo` キーワードを以下のコマンドから省略することができます。

▼ LicenseServer の開始

RaptorXML Server を LicenseServer に正しく登録しライセンス与えるためには、LicenseServer はネットワークのデーモンとして作動していなければなりません。以下のコマンドで、LicenseServer をデーモンとして開始してください！

[< Debian 8]	`sudo /etc/init.d/licenseserver start`
[≥ Debian 8]	`sudo systemctl start licenseserver`
[< CentOS 7]	`sudo initctl start licenseserver`
[≥ CentOS 7]	`sudo systemctl start licenseserver`
[< Ubuntu 15]	`sudo initctl start licenseserver`
[≥ Ubuntu 15]	`sudo systemctl start licenseserver`
[RedHat]	`sudo initctl start licenseserver`

LicenseServer を停止する必要がある場合、上記のコマンドの `start` を `stop` と置換えてください。例えば、:

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

RaptorXML Serverの開始

RaptorXML Server を以下のコマンドを使用してデーモンとして開始します:

[< 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

RaptorXML Serverの登録

- コマンドラインインターフェイスから RaptorXML Server を登録する場合、`licenseserver` コマンドを使用します:
  ```
  sudo /opt/Altova/RaptorXMLServer2021/bin/raptorxml licenseserver [options]
  ServerName-Or-IP-Address
  ```
  例えば、`localhost` がLicenseServer のインストールされたサーバーの名前である場合:
  ```
  sudo /opt/Altova/RaptorXMLServer2021/bin/raptorxml licenseserver localhost
  ```
  上記のコマンドで、`localhost` がLicenseServer のインストールされたサーバーの名前です。RaptorXML Server 実行可能の場所を確認してください:
  ```
  /opt/Altova/RaptorXMLServer2021/bin/
  ```

登録に成功した後 LicenseServer の構成ページのクライアント管理タブに移動し RaptorXML Server にライセンスを割り当てます。

ライセンスの割り当て

登録に成功した後 LicenseServer の構成ページのクライアント管理タブに移動し RaptorXML Server にライセンスを割り当てます。

Altova サーバー製品へのライセンスは製品マシンで使用可能なプロセッサコアの数をベースにしています。例えば、デュアルコア プロセッサはコアが2 つ、クアッド コア プロセッサはコアが4 つ、ヘキサコア プロセッサはコアが6 つ等々。特定のサーバーマシン上の製品にライセンスされたコアの数は、物理または仮想マシンで、サーバーで使用可能なコア数より多くまたは同数である必要があります。例えば、サーバーが8 コア(オクタルコア プロセッサ)の場合、少なくとも 8-コア ライセンスを購入する必要があります。また、ライセンスを合計してコア数を満たすこともできます。2 つの4-コアライセンスは、8-コアライセンスの代わりにオクタルコア サーバーで使用できます。

大きいCPU コアを持つコンピューターサーバーを使用し、少量を処理する場合、少ないコアを割り当てる仮想マシンを作成し、その数のライセンスを購入することもできます。このようなデプロイは、もちろん、サーバーの全ての利用可能なコアが利用されている場合に比べ、処理スピードが落ちます。

メモ 各 Altova サーバー製品のライセンスは、使用されていないライセンス容量があっても、1 度に1 つのクライアントマシンにだけしか使用することができません。例えば10-コアライセンスが6 CPU コアのクライアント マシンに使用される場合、残りの4 コア ライセンスは他のマシンで同時に使用することができません。

3.3.3 環境についてのメモ

フォルダー

以下が RaptorXML Server セットアップの重要なフォルダーのリストです。

- *インストール ルート*

  /opt/Altova/RaptorXMLServer2021/

- *ライセンスファイル*

  /var/opt/Altova/RaptorXML

- *環境設定*

  /etc/profile.d/jdbc.sh
  (通常 **jdbc.sh** と称される) 環境設定ファイルはシステムの起動時に実行されます。内部の定義は、使用する特定の環境に指定される必要があります。上のパスのサンプルは一般的なガイドとして使用してください。
  メモ　環境設定ファイルはシステム上のすべてのユーザーの変数を設定するため、設定する際はお注意してください。例えば、このファイル内でクラスパスを変更すると、変更はシステム全体に適用されます。RaptorXML Server ためにのみ変更を加える場合、ユニットファイルの使用を考慮してください (*JDBC 接続*のセクションで説明されています)。

データベースへの接続

LInux 上では、以下のデータベース接続がサポートされています:

- JDBC — Microsoft Access 以外のサポートされているすべてのデータベースに JDBC を使用することができます。
- ネイティブな接続 — SQLite と PostgreSQL データベースのために使用することができます。
-

JDBC を使用する場合、以下の点に注意してください:

- Java Runtime Envioronment または SDK がインストールされる必要があります。
- ターゲット データベースのための JDBC-接続がインストールされる必要があります。
- 次の環境変数が環境のために正しく設定される必要があります:
  - `CLASSPATH`: JDBC データベースへ接続する .jar ファイルを検索するには以下を行います; .jar ファイルを以下として入力することができます (i) システムの起動時に実行される (`jdbc.sh` などの) 実行可能スクリプト、または、(ii) RaptorXML Server がサービスとして開始される際に実行されるユニットファイル。.jar ファイルを指定するためのユニットファイルの使用は、RaptorXML Server の JDBC 接続のために必要なファイルが既存のシステムの構成を変更することなく使用できる利点があります。ユニットファイルは下にリストされています。
  - `PATH`: JRE を検索します。しかし、インストールによっては必要のない場合があります。
  - `JAVA_HOME`: インストールに応じて必要な場合設定します。

重要なファイルのリスト

以下のシェルスクリプト (または、ユニットファイル) がフォルダー **/opt/Altova/RaptorXML/etc** にコピーされ、既存の構成ファイルを上書きします。必要に応じて必要な変更を加えます。また、上のセクション *JDBC 接続* を参照してください。青でハイライトされた環境固有であり、使用中の環境に合うよう調整されなければなりません。

- シェルスクリプト (ユニットファイル)

```
#- jdbc - environment -
export PATH=/usr/local/jdk1.7.0_17/bin:/usr/lib64/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/qa/bin
export JAVA_HOME=/usr/local/jdk1.7.0_17
export
CLASSPATH=/usr/local/jdbc/oracle/ojdbc6.jar:/usr/local/jdbc/oracle/xdb.jar:/usr/local
/jdbc/oracle/xmlparserv2.jar:/usr/local/jdbc/postgre/postgresql-9.0-
801.jdbc4.jar:/usr/local/jdbc/mssql/sqljdbc4.jar:/usr/local/jdbc/iseries/lib/jt400.ja
r:/usr/local/jdbc/mysql/mysql-connector-java-5.1.16-
bin.jar:/usr/local/jdbc/sqlite/sqlitejdbc-
v056.jar:/usr/local/jdbc/Informix_JDBC_Driver/lib/ifxjdbc.jar:/usr/local/jdbc/sybase/
jconn7/jconn4.jar:/usr/local/jdbc/db2/db2jcc.jar:/usr/local/jdbc/db2/db2jcc_license_c
u.jar:./:
```

3.4 Mac OS X でのセットアップ

このセクションはRaptorXML Server のmacOS システムへのインストール35 とライセンス37 について説明します。

macOS へのインストール35
- システム必要条件35
- root ユーザーのメモ35
- 古いバージョンのAltova サーバー製品のアンインストール35
- macOS パッケージのダウンロード35
- RaptorXML Serverのインストール35
- Altova LicenseServer35
- LicenseServer のバージョン35
- トライアルライセンス35

macOS でのライセンス37
- root ユーザーのメモ37
- LicenseServer の開始37
- RaptorXML Server の開始37
- RaptorXML Serverの登録37
- ライセンスの割り当て37

3.4.1 macOS へのインストール

RaptorXML Server のmacOS へのインストールは利用可能です。インストールとセットアップについては以下で説明されます。

▼ システム必要条件

▼ *macOS*

macOS 10.13 または以降

▼ FlowForce Server 統合

RaptorXML Server をFlowForce Server と共にインストールする場合、FlowForce Server を最初にインストールすることが奨励されます。RaptorXML Server がFlowForce Server の前にインストールされている場合、RaptorXML Server と FlowForce Server を両方インストールした後に、次のコマンドを実行します:

```
cp /usr/local/Altova/RaptorXMLServer2021/etc/*.tool /usr/local/Altova/FlowForceServer
2021/tools
```

このコマンドは**.tool** ファイルをRaptorXML Server の**/etc** ディレクトリからFlowForce Server **/tools** ディレクトリにコピーします。FlowForce Server は**tool** ファイルを必要とし このファイルはRaptorXML Server 実行可能へのパスを含みます。FlowForce Server をRaptorXML Server をインストールする前にインストールする場合このコマンドを実行する必要はありません。

▼ ルートユーザーに関するメモ

RaptorXML Server をインストールするには、管理者(root) 特権を有する必要があります。ですから、インストールはroot ユーザーとして実行されなければなりません。`root` としてログインする場合、`sudo` キーワードを以下のコマンドから省略することができます。

Altova サーバー製品の古いバージョンのアンインストール

RaptorXML Server をインストールする前に、サービスを以下のコマンドで停止します::

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.RaptorXMLServer2021.plist
```

サービスが停止されたか確認するには、アクティビティ モニター ターミナルを開き、RaptorXML Serverがリストにないことを確認します。アプリケーション ターミナルで、%APPNAME%> アイコンを右クリックし、[**ごみ箱へ移動**] を選択します。アプリケーションはごみ箱に移動されます。しかし、 `usr` フォルダーからアプリケーションを削除しなければなりません。このためには以下のコマンドを使用します:

```
sudo rm -rf /usr/local/Altova/RaptorXMLServer2021/
```

Altova LicenseServer の古いバージョンをアンインストールする場合、サービスとしての作動を停止しなければなりません。このためには以下のコマンドを使用します:

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

サービスが停止されたか確認するには、アクティビティ モニター ターミナルを開き、LicenseServer がリストにないことを確認します。RaptorXML Server の説明と同じ手順でアンインストールします。

ディスクイメージファイルのダウンロード

Altova Web サイトからディスクイメージファイル(.dmg) をダウンロードします (http://www.altova.com/ja/download.html)。

RaptorXML Server のインストール

ダウンロードされたディスクイメージ (.dmg) をクリックして開きます。これにより RaptorXML Server インストーラーが使用中のコンピューター上で新規の仮想ドライブとして表示されます。新規の仮想ドライブ上で、インストーラー パッケージ (.pkg) をダブルクリックします。インストーラーウィザードに従います。これらは、続行するために必要なライセンス使用許諾書への同意が必要なステップを含むすぐに理解することのできるステップです。インストール後、ドライブを取り出すには、右クリックして **取り出し** を選択します。

RaptorXML Server パッケージはフォルダー内にインストールされます

/usr/local/Altova/RaptorXMLServer2021 (アプリケーション バイナリ)
/var/Altova/RaptorXMLServer (データファイル: データベースとログ)

RaptorXML Server サーバデーモンがインストール後とマシンの再起動後に自動的に開始されます。次のコマンドを使用して RaptorXML Server をデーモンとして開始することができます:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.RaptorXMLServer2021.plist
```

Altova LicenseServer

RaptorXML Serverを含む Altova サーバー製品を作動するには、サーバー製品はネットワークで Altova LicenseServer を介して、ライセンスを与えられなければなりません。

macOS システムでは、Altova LicenseServer は個別にインストールされる必要があります。Altova Web サイト から Altova LicenseServer をダウンロードして、インストーラー パッケージをダブルクリックし、インストールを開始します。インストールを続行するためには、使用許諾契約書に合意する必要があります。

Altova LicenseServer を Altova Web サイトからダウンロードし個別にインストールすることもできます (http://www.altova.com/ja/download.html)。

LicenseServer パッケージは以下のフォルダーにインストールされます:

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

RaptorXML Server を Altova LicenseServer に登録し、ライセンスを供与するには macOS でのライセンス 37 のセクションを参照してください。

▼ LicenseServer バージョン

- Altova サーバー製品はインストールされた RaptorXML Server バージョンに適切な LicenseServer のバージョン、または、LicenseServer の最新のバージョンが必要です。
- RaptorXML Server の特定のバージョンに適切な LicenseServer のバージョンが RaptorXML Server のインストール中表示されます。
- LicenseServer の新しいバージョンをインストールする前に、古いバージョンはアンインストールされる必要があります。LicenseServer インストーラーは古いバージョンを検出すると自動的に行います。
- LicenseServer バージョンは下位互換性があります。RaptorXML Server の全ての古いバージョンと作動します。
- RaptorXML Server の新しいバージョンをインストールする場合、そして、インストールされている LicenseServer バージョンが適切な LicenseServer バージョンより古い場合、Altova Web サイトから利用可能な最新バージョンをインストールします。
- LicenseServer をアンインストールする際、古いバージョンの LicenseServer のすべての登録とライセンス情報はサーバーマシーンのデータベースに保存されます。このデータは新しいバージョンがインストールされる際、自動的に新しいバージョンにインポートされます。
- 現在インストールされている LicenseServer のバージョン番号は LicenseServer 構成ページ（全てのタブ）の下部にあります。

  現在のバージョン: 3.6

3.4.2 macOS でのライセンス

RaptorXML Server は作動するために Altova LicenseServer にライセンスされている必要があります。ライセンス供与は以下の 2 つのステップから構成されています:

1. **RaptorXML Server** に LicenseServer を登録します。RaptorXML Server から登録することができます。
2. RaptorXML Server のライセンスを割り当てます。ローカルマシンまたは、ネットワーク上のマシンにインストールされる必要のある LicenseServer からライセンスを割り当てることができます。

実行するステップは以下に手短に説明されています。詳細は、Altova Web サイト のユーザーマニュアルを参照してください。

▼root ユーザーに関するメモ

RaptorXML Server をインストールするには、管理者(root) 特権を有する必要があります。 ですからインストールは ユーザーとして実行されなければなりません。`root` としてログインする場合、`sudo` キーワードを以下のコマンドから省略できます。

▼ LicenseServer の開始

RaptorXML Server を LicenseServer に正しく登録しライセンス与えるためには、LicenseServer はネットワークのデーモンとして作動していなければなりません。以下のコマンドで、LicenseServer をデーモンとして開始してください:

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

LicenseServer を停止する必要がある場合、上記コマンドの load を unload と置換えてください:

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

RaptorXML Serverの開始

RaptorXML Server サーバデーモンがインストール後とマシンの再起動後に自動的に開始されます。次のコマンドを使用して RaptorXML Server をデーモンとして開始することができます:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.RaptorXMLServer2021.plist
```

RaptorXML Server を停止する必要がある場合、次を使用します:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.RaptorXMLServer2021.plist
```

RaptorXML Server の登録

コマンドラインインターフェイスから RaptorXML Server を登録する場合、licenseserver コマンドを使用します:

```
sudo /usr/local/Altova/RaptorXMLServer2021/bin/RaptorXML licenseserver
[options] ServerName-Or-IP-Address
```

例えば、localhost が LicenseServer のインストールされたサーバーの名前である場合:

```
sudo /usr/local/Altova/RaptorXMLServer2021/bin/RaptorXML licenseserver localhost
```

上記のコマンドで、localhost が LicenseServer のインストールされたサーバーの名前です。RaptorXML Server 実行可能の場所を確認してください:

```
/usr/local/Altova/RaptorXMLServer2021/bin/
```

登録に成功した後、LicenseServer 構成ページのクライアント管理タブに移動して、RaptorXML Server へライセンスを割り当てます。

ライセンスの割り当て

RaptorXML Server の登録に成功した後、LicenseServer の構成ページのクライアント管理タブにリストされます。移動して RaptorXML Server にライセンスを割り当てます。

Altova サーバー製品へのライセンスは製品マシンで使用可能なプロセッサコアの数をベースにしています。例えば、デュアルコア プロセッサはコアが2 つ、クアッド コア プロセッサはコアが4 つ、ヘキサコア プロセッサはコアが6 つ等々。特定のサーバーマシン上の製品にライセンスされたコアの数は、物理または仮想マシンで、サーバーで使用可能なコア数よりも多くまたは同数である必要があります。例えば、サーバーが8 コア(オクタルコア プロセッサ)の場合、少なくとも 8-コア ライセンスを購入する必要があります。また、ライセンスを合計してコア数を満たすこともできます。2 つの4-コアライセンスは、8-コアライセンスの代わりにオクタルコア サーバーで使用できます。

大きいCPU コアを持つコンピューターサーバーを使用し、少量を処理する場合、少ないコアを割り当てる仮想マシンを作成し、その数のライセンスを購入することもできます。このようなデプロイは、もちろん、サーバーの全ての利用可能なコアが利用されている場合に比べ、処理スピードが落ちます。

メモ 各 Altova サーバー製品のライセンスは、使用されていないライセンス容量があっても、1 度に1 つのクライアントマシンにだけしか使用することができません。例えば10-コアライセンスが6 CPU コアのクライアント マシンに使用される場合、残りの4 コア ライセンスは他のマシンで同時に使用することができません。

3.4.3 環境についてのメモ

フォルダー

以下がRaptorXML Server セットアップの重要なフォルダーのリストです。

- *インストール ルート*

  /usr/local/Altova/RaptorXMLServer2021/

- *ライセンスファイル*

  /var/Altova/RaptorXML

- *環境設定*

  /Library/LaunchDaemons/com.altova.RaptorXML.plist
  環境設定ファイルは特定の環境に応じて定義してください。上記の例のパスは一般的なガイドとして使用してください。
  メモ　これらの環境変数は、RaptorXML Server として定義されたファイルに存在しなくてはなりません。デフォルトではこのファイルは、次の通りです:

データベースへの接続

MacOS 上では、以下のデータベース接続がサポートされています:

- JDBC － Microsoft Access 以外のサポートされているすべてのデータベースにJDBC を使用することができます。
- ネイティブな接続 － SQLite とPostgreSQL データベースのために使用することができます。
-

JDBC を使用する場合、以下の点に注意してください:

- Java Runtime Envioronment またはSDK がインストールされる必要があります。
- ターゲット データベースのためのJDBC-接続がインストールされる必要があります。
- 次の環境変数が環境のために正しく設定される必要があります:
  - `CLASSPATH`: JDBC データベースへ接続する.jar ファイルを検索するには以下を行います; .jar ファイルを以下として入力することができます (i) システムの起動時に実行される (`jdbc.sh` などの) 実行可能スクリプト、または、(ii) RaptorXML Server がサービスとして開始される際に実行されるユニットファイル。.jar ファイルを指定するためのユニットファイルの使用は、RaptorXML Server のJDBC 接続のために必要なファイルが既存のシステムの構成を変更することなく使用できる利点があります。ユニットファイルは下にリストされています。
  - `PATH`: JRE を検索します。しかし、インストールによっては必要のない場合があります。
  - `JAVA_HOME`: インストールに応じて必要な場合設定します。

3.5 XML カタログ

The XML カタログ構造によりローカルフォルダーからファイルを抽出することが可能になり、—変更のためにカタログファイルURI のみが必要になるため、に処理スピード全体が向上し、ドキュメントのポータビリティも向上します。詳細に関しては、カタログの機能 41 のセクションを参照してください。

Altova のXML 製品はDTD とXML スキーマなど共通に使用されるファイルに素早くアクセスしロードするためにカタログメカニズムを使用します。このカタログ構造はカスタム化することが可能で、またユーザーにより拡張可能です。セクション RaptorXML Server 内のカタログ構造 43 とカタログのカスタム化 45 内で説明されています。セクション システムロケーションのための変数 47 は共通のシステムのロケーションのためにW indows 変数をリストしています。共通で使用されているフォルダーをロケートするためにこれらの変数はカタログファイル内で使用することができます。

このセクションは次のサブセクションに整理されています:

- カタログの仕組み 41
- RaptorXML Server 内のカタログ構造 43
- カタログのカスタム化 45
- W indows システムロケーションのための変数 47

これらのカタログの詳細に関してはXML カタログ仕様) を参照してください。

3.6 カタログの仕組み

カタログをDTD とXML スキーマをリダイレクトするために使用することができます。両方のケースのメカニズムの後ろにあるコンセプトは同じですが、違いの詳細は以下に説明されている通りです。

DTD

カタログはDTD への呼び出しをローカルのURI にリダイレクトするために使用されます。これはマッピング、カタログファイル、ローカルURI により必要とされるパブリックまたはシステム識別子により達成されます。XML ファイル内の**DOCTYPE** 宣言が読み取れれると、パブリックまたはシステム識別子が必要とされるローカルリソースをカタログファイルマッピングによりロケートします。

人気のあるスキーマに関しては**PUBLIC** 識別子は通常定義済みです。このためカタログファイル内のURI のみが正しいローカルコピーに対して**PUBLIC** 識別子 をマップします。XML ドキュメントが解析されると内部の**PUBLIC** 識別子が読み取られます。この識別子がカタログファイル内で見つかると、カタログファイル内の対応するURL がルックアップされ、スキーマがこのロケーションから読み取られます。例えば、次のSVG ファイルがRaptorXML Server 内で開かれるとします:

```
<?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>
```

カタログはこのSVG ファイルの**PUBLIC** 識別子を検索します。カタログファイルは以下のエントリを含んでいます:

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

この場合 **PUBLIC** 識別子のために一致するものがあります。この結果 SVG DTD のためのルックアップが(カタログファイルに相対的な)URL **schemas/svg/svg11.dtd** にリダイレクトされます。これはSVG ファイルのためにDTD として使用されるローカルファイルです。カタログ内で **Public** ID のためのマッピングが存在しない場合、XML ドキュメント内のURL が使用されます(上のSVG ファイルのサンプルの場合、これはインターネット URL です: **http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd**)。

XML スキーマ

RaptorXML Server 内では XML スキーマとカタログを使用することができます。XML インスタンスファイル内で、スキーマへのレファレンスがXML ドキュメントのトップレベルの要素の**xsi:schemaLocation** 属性内で発生します。例えば、

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

xsi:schemaLocation 属性の値には2 つの部分があります: (上で緑色で表示されている)名前空間のパートと(ハイライトされている)URI パート。名前空間の部分は代替リソースをマップするためにカタログ内で使用されています。例えば、以下のカタログエントリは上のスキーマレファレンスを代替のロケーションのスキーマにリダイレクトします。

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

通常 **xsi:schemaLocation** 属性の値のURI 部分は実際のスキーマのロケーションのパスです。しかしながら、スキーマがカタログを介

して参照される場合、URI の部分は実際の XML スキーマをポイントする必要はなく、**xsi:schemaLocation** 属性の文法の妥当性が保持されるために存在する必要があります。Foo の値は、例えば、属性の値の部分が妥当であるため十分です。

3.7 RaptorXML Server 内のカタログ構造

RaptorXML Server が開始されると、(*以下のリスティングで示されるような構造を持った*) **RootCatalog.xml** というファイルがロードされ、ファイル内部に記述されているカタログファイルが検索されます。このファイルを修正して、検索するカタログファイルを増やすこともできます(各カタログファイルは **nextCatalog** 要素にて記述されます)。各カタログファイル内に記述されている URI が、ファイル内で指定されたマッピングに従って解決されます。

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>
```

上のリスティングはカスタムカタログ(named **CustomCatalog.xml**)と(W3C XML スキーマと SVG スキーマ)などの一般的に使用されるスキーマをロケートするカタログのセットを参照します

- **CustomCatalog.xml** は RaptorXML Server アプリケーションフォルダーの その他 サブフォルダーにあります。自身のマッピングを作成するスケルトンファイルです。共通のスキーマフォルダー内のカタログファイルにより指されていない必要とするスキーマのためにマッピングを **CustomCatalog.xml** に追加することができます。OASIS カタログメカニズムのサポートされる要素を使用してこれを行います(*次のセクションを参照*)。
- (共通に使用されるスキーマのセットを含む変数 **%CommonSchemasFolder%** を使用してロケートされる) 共通のスキーマフォルダーこれらのスキーマフォルダの内部は保存された対応するスキーマのコピーをポイントする URI に対するパブリックおよびまたはシステム識別子をマップする **catalog.xml** ファイルです。
- **CoreCatalog.xml** は RaptorXML Server アプリケーションフォルダー内にあり、XML ドキュメントの Altova の Authentic ビューを生成するために使用される StyleVision パワースタイルシートなど RaptorXML Server 固有のプロセスにより使用されるスキーマとスタイルシートをロケートするために使用されます。

ロケーション変数

RootCatalog.xml 内で使用されている変数は以下の値を持っています(*上のリスティング*):

%PersonalFolder%	現在のユーザーの個人的なフォルダー、例えば C:\Users\<name>\Documents
%CommonSchemasFolder%	C:\ProgramData\Altova\Common2021\Schemas
%ApplicationWritableDataFolder%	C:\ProgramData\Altova

カタログファイルとスキーマのロケーション

多種のカタログファイルの場所に注意してください。

- **RootCatalog.xml**、**CustomCatalog.xml**, と **CoreCatalog.xml** は RaptorXML Server アプリケーションフォルダー内にあります。
- **Catalog.xml** 入るはそれぞれ特定のスキーマフォルダー内にあり、これらのスキーマフォルダーは共通のスキーマフォルダー内にあります。

3.8 カタログのカスタム化

CustomCatalog.xml（または RaptorXML Server により読み取られる他のカタログファイル）内でエントリを作成する場合OASIS カタログ仕様の以下の要素のみを使用してください。下の各要素は属性の値の説明と共にリストされています。詳細に関しては XML カタログ仕様）を参照してください。各要素はその要素のベース URI を指定するために使用される **xml:base** 属性を取ることができます。

- `<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"/>`

以下の点に注意してください：

- 多くのスタイルシート同様パブリック識別子が存在しない場合、システム識別子を **system** 要素を介して直接 URL にマップすることができます
- uri 要素を使用して URI を他の URI にマップすることができます。
- rewriteURI と rewriteSystem 要素により URI またはシステム識別子の開始の部分をそれぞれ書き換えることが可能になります。これによりファイルパスの開始部分が置き換えられこの結果他のディレクトリをターゲットにすることができます。これらの要素の詳細に関しては XML カタログ仕様 を参照してください。

リリース 2014 以降RaptorXML Server は XML Catalogs specification (OASIS Standard V1.1, 7 October 2005) 仕様を厳密に順守しています。この仕様は厳密に（パブリック ID またはシステム ID）外部識別子ルックアップを（パブリック ID またはシステム ID を持たない URI ）URI ルックアップと区別しています。名前空間 URI は単純にパブリック ID またはシステム ID ではなく URI と考えられ外部識別子ルックアップではなく URI ルックアップとして使用されるべきです。バージョン 2014 以前の RaptorXML Server バージョンではスキーマ名前空間 URI は **<public>** マッピングを使用して訳されてきました。2014 以降から **<uri>** マッピングが使用されています。

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

RaptorXML Server が参照されたスキーマを検索する方法

xsi:scemaLocation 属性を介して XML ドキュメント内で参照されるスキーマ（*以下に表示されています*）。
xsi:schemaLocation 属性の値には2 つの部分があります：（緑色で表示されている）名前空間のパートとハイライトされている）URI パート。

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

参照されるスキーマを検索するためのステップは検証オプション **--schemalocation-hints** と **--schema-mapping** により異なります。以下は2 つのオプションのそれぞれの値のための手順です：

- **--schemalocation-hints=load-by-schemalocation | load-by-namespace | load-combining-both | ignore**
  xsi:schemaLocation と **xsi:noNamespaceSchemaLocation** 属の振る舞いを指定します：スキーマドキュメントをロードするか、はいの場合どの情報を使用して検索するかなど。（デフォルトは **load-by-schemalocation** です）。
  - **load-by-schemalocation**
    1. **xsi:schemaLocation** の URI 部分がカタログにマップされている場合、結果する URI をロードします。

2. URI により直接ロード

❖ **load-by-namespace**

1. **xsi:schemaLocation** の名前空間の部分がカタログにマップされている場合、結果する URI をロードします。
2. 何もロードしない。

❖ **load-combining-both**

1. **xsi:schemaLocation** の URI 部分がカタログにマップされている場合、結果する URI をロードします。
2. **xsi:schemaLocation** の名前空間の部分がカタログにマップされている場合、結果する URI をロードします。
3. URI の部分を直接ロード

- **--schema-mapping=prefer-schemalocation | prefer-namespace**
  スキーマロケーションと名前空間が両方スキーマドキュメントを検索するために使用される場合、このオプションはどのオプションがカタログルックアップの際に優先されるかを指定します。(デフォルトは **prefer-schemalocation** です)。このオプションは **load-combining-both** 上記のバリアント内の最初の2つのステップを変更するために使用されています。

XML スキーマ仕様

XML スキーマ仕様情報は RaptorXML Server にビルトされており XML スキーマ(`.xsd`)ドキュメントの妥当性はこの内部情報に対してチェックされます。XML スキーマドキュメント内では、このためレファレンスは XML スキーマ仕様を定義するスキーマになってはなりません。

`%AltovaCommonSchemasFolder%\Schemas\schema` フォルダー内の **catalog.xml** ファイルは古い XML スキーマ仕様を実装する DTD へのレファレンスを含んできます。XML スキーマドキュメントをこれらのスキーマに対して検証してはなりません。参照されるファイルはドキュメントをこれらの古い奨励に従い作成する場合に編集の目的で RaptorXML Server に入力ヘルパー情報を与えるために含まれています。

(C) 2015-2021 Altova GmbH

3.9 Windows システムロケーションのための変数

nextCatalog 要素ではシェル環境変数が使用され、システムに関係する場所へのパスが参照されます（*上の RootCatalog.xml の内容を参照*）。以下のシェル環境変数がサポートされます:

%PersonalFolder%	現在のユーザーの個人的なフォルダーへのフルパス、例えば C:\Users\<name>\Documents
%CommonSchemasFolder%	C:\ProgramData\Altova\Common2021\Schemas
%ApplicationWritableDataFolder%	C:\ProgramData\Altova
%AltovaCommonFolder%	C:\Program Files\Altova\Common2021
%DesktopFolder%	現在のユーザーのデスクトップフォルダーへのフルパス。
%ProgramMenuFolder%	現在のユーザーのプログラムメニューフォルダーへのフルパス。
%StartMenuFolder%	現在のユーザーの開始メニューフォルダーへのフルパス。
%StartUpFolder%	現在のユーザーのスタートフォルダーへのフルパス。
%TemplateFolder%	現在のユーザーのテンプレートフォルダーへのフルパス。
%AdminToolsFolder%	現在のユーザーの管理ツールを保管するファイルシステムディレクトリへのフルパス。
%AppDataFolder%	現在のユーザーのアプリケーションデータフォルダーへのフルパス。
%CommonAppDataFolder%	全てのユーザーのアプリケーションデータを含むファイルディレクトリへのフルパス。
%FavoritesFolder%	現在のユーザーの[お気に入り]フォルダへフルパス。
%PersonalFolder%	現在のユーザーの個人フォルダーへのフルパス。
%SendToFolder%	現在のユーザーの送信フォルダーへのフルパス。
%FontsFolder%	システムフォントフォルダーへのフルパス。
%ProgramFilesFolder%	現在のユーザーのプログラムファイルフォルダーへのフルパス。
%CommonFilesFolder%	現在のユーザーの共通ファイルフォルダーへのフルパス。
%WindowsFolder%	現在のユーザーの Windows フォルダーへのフルパス。
%SystemFolder%	現在のユーザーのシステムフォルダーへのフルパス。

%LocalAppDataFolder%	ローカル(非ローミング) アプリケーションのためのデータレポジトリとしての役割を果たすファイルシステムディレクトリへのフルパス。
%MyPicturesFolder%	ピクチャフォルダーへのフルパス。

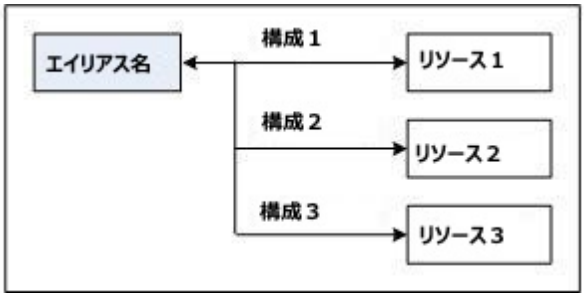
3.10 グローバルリソース

このセクション:

- グローバルリソースに関して 49
- グローバルリソースの使用方法 49

グローバルリソースに関して

Altova グローバルリソースファイルは、下の図に示されるように、異なる構成を介して、エイリアスを複数のリソースにマップします。エイリアスは、ですから、構成を切り替えることにより異なるリソースをアクセスするために切り替えられることができます。



グローバルリソースは、Altova XMLSpy などのAltova 製品内で定義されており、グローバルリソースXML ファイル内に保存されています。RaptorXML は、グローバルリソースを入力として、使用することができます。これを行うには、グローバルリソースファイルの名前と場所、および使用されるエイリアスと構成が必要になります。

グローバルリソースを利用する利点は、リソースを構成名を切り替えるのみで変更できることです。RaptorXML を使用する際には、これは、異なる `--globalresourcesconfig | --gc` オプションの値を提供することを意味し、異なるリソースを使用することができます。(*下のサンプルを参照してください*)

RaptorXML を使用してのグローバルリソースの使用方法

RaptorXML コマンドのための入力としてグローバルリソースを指定する場合、以下のパラメーターが必要です:

- グローバルリソースXML ファイル(`--globalresourcesfile | --gr` オプションと共にCLI で指定されています)
- 必要な構成(`--globalresourcesconfig | --gc`オプションと共にCLI で指定されています)
- エイリアス。ファイル名が必要な個所またはRaptorXML が(`xsi:schemaLocation` 内などの)ファイル名を検索するXML ファイル内の箇所で、CLI で直接指定されることができます。

例えば、`input.xml` を`transform.xslt` と`output.html` に変換する場合、通常ファイル名を使用する次のコマンドをCLI 上で使用して行うことができます:

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

しかし、グローバルリソース定義が`FirstConfig` と呼ばれる構成を介して、ファイルリソース`FirstInput.xml` に対してエイリアス`MyInput`に一致する場合、CLI 上でエイリアス`MyInput` を以下のように使用することができます:

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

SecondConfig と呼ばれる構成を介してエイリアス MyInput に一致する、SecondInput.xml などの他のファイルリソースがある場合、このリソースは、前のコマンドの --gc オプションを変更するだけで使用されることができます:

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

メモ 上のサンプルでは、ファイルリソースが使用されています。ファイルリソースは altova://file_resource/ と共にプレフィックスを付けられなければなりません。フォルダーであるグローバルリソースも使用することができます。フォルダーリソースを識別するには、次を使用します: altova://folder_resource/AliasName。CLI 上では、フォルダーリソースをファイルパスの一部として使用することもできることに注意してください。例: altova://folder_resource/AliasName/input.xml 。

3.11 セキュリティの問題

このセクション:

- HTTP インターフェイスに関連するセキュリティの問題 51
- Python スクリプトを安全にする 51

RaptorXML Server のインターフェイス機能の一部はセキュリティの問題を伴う可能性があります。この点に関しては以下に解決策と共に説明されています。

HTTP REST インターフェイスに関連するセキュリティの問題

HTTP REST インターフェイスは、デフォルトでは、クライアントにより指定された(HTTP プロトコールを使用してアクセスすることのできる)すべての箇所に結果ドキュメントを書き込むことができます。ですから、RaptorXML Server を構成する際、このセキュリティのアスペクトを考慮することは重要です。

セキュリティが危害を受けるまたはインターフェイスが誤用される問題がある場合、結果ドキュメントが、サーバー上の専用の出力ディレクトリに書き込まれるようにサーバーを構成することができます。これには、サーバー構成ファイルの `server.unrestricted-filesystem-access` 205 のオプションの設定を false に指定します。この様にアクセスが制限されていると、クライアントは結果ドキュメントを専用の出力ディレクトリから `GET` リクエストを使用してダウンロードすることができます。または、管理者が結果ドキュメントファイルをサーバーからターゲットの場所にコピー/ダウンロード することができます。

Python スクリプトを安全にする

HTTP を介して RaptorXML Server に対して、Python スクリプがコマンド内で指定されている場合、スクリプトは信頼できるディレクトリ 205 にある場合のみ作動します。スクリプトは信頼できるディレクトリから実行されます。Python スクリプトをこれ以外のディレクトリから指定するとエラーが生じます。信頼できるディレクトリは サーバー構成ファイル 203 の `server.script-root-dir` 204 設定で指定されており、信頼できるディレクトリは Python スクリプトを使用する場合**指定されていなければなりません。**使用されるすべての Python スクリプトがこのディレクトリに保存されていることを確認してください。

HTTP ジョブリクエストのためにサーバーにより生成される出力は、(`output-root-directory` 205 のサブディレクトリである) ジョブ出力ディレクトリ 205 に書き込まれますが、この制限はあらゆる箇所に書き込むことのできる Python スクリプトに対しては適用されません。サーバー管理者は、信頼できるディレクトリ 205 内のスクリプトを脆弱性に関する問題の可能性の点からレビューする必要があります。

4 コマンドライン インターフェイス (CLI)

RaptorXML Server 実行可能ファイルはコマンドラインインターフェイス(CLI) から呼び出すことのできるアプリケーション機能を提供します。実行可能ファイルへのパスは以下の通りです:

Linux	`/opt/Altova/RaptorXMLServer2021/bin/`**`raptorxml`**
Mac	`/usr/local/Altova/RaptorXMLServer2021/bin/`**`raptorxml`**
Windows	`<`*`ProgramFilesFolder`*`>\Altova\RaptorXMLServer2021\bin\`**`RaptorXML.exe`**

使用方法

コマンドライン構文は以下の通りです:

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

- `--help` (短縮形 `--h`) は与えられたコマンドのヘルプテキストを表示します。コマンドに名前がない場合、実行可能ファイルの全てのコマンドがコマンドの短い説明と共にリストされます。
- `--version` はRaptorXML Server のバージョン番号を表示します。
- `<command>` は実行されるコマンドです。このセクションのサブセクション内でコマンドについて説明されています *(のリストを参照してください)*。
- `[options]` はコマンドのオプションです。これらのコマンドはリストされ、対応するコマンドと共に説明されています。
- `[arguments]` はコマンドの引数です。これらのコマンドはリストされ、対応するコマンドと共に説明されています。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**`RaptorXML`** (および管理コマンドのための**`RaptorXMLServer`**)
W indows とUnix (Linux, Mac) 上の **`raptorxml`** (および管理コマンドのための**`raptorxmlserver`**)

* 小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はW indows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

CLI コマンド

使用可能な CLI コマンドは、機能別に整理されており、このセクションのサブセクションで説明されています。

- XML、DTD、XSD 検証コマンド 53
- 整形式チェックコマンド 72
- XQuery コマンド 82
- XSLT コマンド 107
- JSON/Avro コマンド 120
- XML 署名コマンド 149
- 一般的なコマンド 161
- ローカライズコマンド 164
- ライセンスコマンド 167
- 管理コマンド 171

4.1 XML、DTD、XSD 検証コマンド

XML 検証コマンドはこのセクションのサブセクションで詳しく述べられています:

- valxml-withdtd [53]: DTD に対してインスタンスドキュメントを検証します。
- valxml-withxsd [56]: XML スキーマに対して XML インスタンスドキュメントを検証します。
- valdtd [62]: DTD ドキュメントを検証します。
- valxsd [66]: W 3C XML スキーマ(XSD) ドキュメントを検証します。

4.1.1 valxml-withdtd (xml)

valxml-withdtd | xml コマンドは、1 つまたは複数の XML インスタンスドキュメントを DTD に対して検証します。

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

- *InputFile* 引数は、検証する XML ドキュメントです。XML ドキュメント内に、DTD に対する参照が存在する場合、`--dtd` のオプションは必要ありません。
- 複数のドキュメントを検証するには、以下を行います: (i)各ファイルを空白で区切り、CLI で検証されるファイルをリストします。または(ii) 検証されるファイルをテキストファイル(`.txt` ファイル) でファイル名を各ラインに表示し、リストします。そして、このテキストファイルを*InputFile* 引数として、`true` と設定された`--listfile` [186] オプションと共に返します。(*下のオプションのリストを参照してください*)。

サンプル

valxml-withdtd コマンドのサンプル:

- **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`

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(W indows、Linux、および Mac) で使用することができますが、小文字(`RaptorXML`) は W indows と Mac のみで使用することができます。
＊Linux と Mac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: `--option=value`。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。`--h, --help` オプションを使用してコマンドに関する情報を表示します。

検証と処理

dtd

--dtd = *FILE*
検証に使用される外部 DTD ドキュメントを指定します。XML ドキュメント内に外部 DTD への参照が存在する場合、CLI オプションが外部参照を上書きします。

listfile

--listfile = `true|false`
`true` の場合、コマンドの *InputFile* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

namespaces

--namespaces = `true|false`
名前空間対応処理を有効化します。これは、XML インスタンス内の間違った名前空間のため発生するエラーをチェックするために役に立ちます。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

recurse

--recurse = `true|false`
ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

streaming

--streaming = `true|false`
ストリーミング検証を有効化します。デフォルトは `true` です。ストリーミングモードでは、メモリに保管されるデータが最小化され、処理がより速くなります。欠点は、後に情報が必要になる可能性があります。例えば、XML インスタンスドキュメントのデータモデルが使用できない場合があります。この様なシチュエーションでは、ストリーミングモードは(`--streaming` に `false` の値を与え、)オフにされる必要があります。 `valxml-withxsd` コマンドを使用して、`--script` オプションを使用するとストリーミングを無効化することができます。`--streaming` オプションは、`--parallel-assessment` が `true` に設定されている場合、の場合無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

カタログとグローバルリソース

- catalog

  --catalog = *FILE*
  インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス(`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`)です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *FILE*
  ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン

  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きの XML フォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は `100` です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は100 です。

4.1.2 valxml-withxsd (xsi)

valxml-withxsd | xsi コマンドは、1 つまたは複数のXML インスタンスドキュメント をW 3C XML スキーマ定義言語(XSD) 1.0 および1.1 仕様に従い検証します。

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

- *InputFile* 引数は、検証されるXML ドキュメントです。--schemalocation-hints 188 オプションはスキーマを検索するためにどのメカニズムが使用されるかを指定します。--xsd=FILE 187 オプションはXML ファイルがスキーマ参照を含まない場合使用されるスキーマを指定します。
- 複数のドキュメントを検証するには、以下を行います: (i)各ファイルを空白で区切り、CLI で検証されるファイルをリストします。または(ii) 検証されるファイルをテキストファイル(.txt ファイル) でファイル名を各ラインに表示し、リストします。そして、このテキストファイルを *InputFile* 引数として、true と設定された--listfile 186 オプションと共に返します。(*下のオプションのリストを参照してください*)。

メモ --script オプションを使用する場合は、Python スクリプト 306 を実行してください。--streaming=false も指定されていることを確認してください。

サンプル

valxml-withxsd コマンドのサンプル:

- **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

コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字が文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

検証と処理

assessment-mode

--assessment-mode = lax|strict
XSD 仕様で定義されているようにスキーマの有効性評価モードを指定します。デフォルト値は strict です。 XML インスタンスドキュメントはこのオプションで指定されたモードに従い検証されます。

ct-restrict-mode

--ct-restrict-mode = 1.0|1.1|default
複合型の制約のチェック方法を指定します。**1.0** の値は、XSD 1.1 検証モードであっても、XSD 1.0 仕様に従い複合型の制約をチェックします。**1.1** の値は、XSD 1.0 検証モードであっても、XSD 1.1 仕様に従い複合型の制約をチェックします。**default** の値は、現在の検証モード (1.0 または 1.1) 内の仕様に従い複合型を検証します。デフォルトの値は **default** です。

listfile

--listfile = true|false
true の場合、コマンドの *InputFile* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は false です。 (代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI

には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

parallel-assessment [pa]

--pa | --parallel-assessment = `true|false`
`true` に設定されている場合、パラレルにスキーマの有効性の評価が実行されます。これは、あるレベルに128 個以上の要素が存在する場合、複数のスレッドを使用してこれらの要素が処理されることを意味します。ですから、大きな XML ファイルは、このオプションが有効化されている場合より早く処理されることができます。パラレル評価は、階層的なレベルごとに実行されますが、単一のインフォセットでは、複数のレベルで実行されることもできます。パラレル評価はストリーミングモードでは実行することができません。このため、`--streaming` オプションは `--parallel-assessment` が `true` に設定されている場合、無視されます。また、メモリ使用は `--parallel-assessment` オプションが使用される場合高いことに注意してください。 デフォルトの設定は `false` です。オプションの短いフォームは `--pa` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

recurse

--recurse = `true|false`
ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *`InputFile`* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションの デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

report-import-namespace-mismatch-as-warning

--report-import-namespace-mismatch-as-warning = `true|false`
`xs:import` を使用してエラーから警告にスキーマをインポートする際、名前空間またはターゲット名前空間の不一致エラーをダウングレードします。デフォルトの値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

schema-imports

--schema-imports = `load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only`
それぞれオプションの `namespace` 属性とオプションの `schemaLocation` 属性を持つ、`xs:import` 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL">`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: `load-preferring-schemalocation`.
振る舞いは以下の通りです:

- `load-by-schemalocation`: カタログマッピング 40 を考慮し、`schemaLocation` 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
- `load-preferring-schemalocation`: `schemaLocation` 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。`schemaLocation` 属性が存在しない場合、`namespace` 属性の値がカタログマッピング 40 を介して使用されます。スキーマこれはデフォルトの値です。
- `load-by-namespace`: カタログマッピング 40 を介して、`namespace` 属性の値がスキーマを検索するために使用されます。
- `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40, がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。カタログマッピング 40 が存在しない場合、`schemaLocation` 属性が使用されます。

- `license-namespace-only`: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: `load-by-schemalocation`。

- `load-by-schemalocation` 値はXML インスタンスドキュメント内の `xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性 の スキーマの場所のURL 317 使用します。これはデフォルト の値です。
- `load-by-namespace` 値は、`xsi:schemaLocation` の名前空間の部分 317 を、そして `xsi:noNamespaceSchemaLocation` の場合は空の文字列を取ります。カタログマッピング 40 を介してスキーマを検索します。
- `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が `ignore` の場合、`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(`--schemalocation-hints` または `--schema-imports` オプションが `load-combining-both` の値を有する場合、また、関連する名前空間とURL のパートが双方カタログマッピング 40 を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。`prefer-schemalocation` 値はURL マッピングを参照します)。) デフォルトは `prefer-schemalocation` です。

script

--script = *FILE*
検証が完了した後、提出されたファイル内のPython スクリプトを実行します。1 つ以上のスクリプトを指定するため、オプションを複数回追加します。

script-api-version

`--api, --script-api-version =` `1; 2; 2.1; 2.2; 2.3; 2.4; 2.4.1; 2.5; 2.6; 2.7; 2.8; 2.8.1; 2.8.2`
スクリプトで使用されるPython API バージョンを指定します。デフォルト値は 現在 **`2.8.2`** である最新バージョンです。`1` および `2` の値の代わりに、それぞれ値 `1.0` および `2.0` を使用することができます。

script-output

--script-output = *FILE*
FILE という名前のファイルにスクリプトの標準出力を書き込みます。

script-param

--script-param = *KEY:VALUE*
Python スクリプト実行中にアクセスすることのできる、追加ユーザー指定パラメーター。1 つ以上のスクリプトパラメーターを指定するため、オプションを複数回追加します。

- streaming

  --streaming = `true|false`
  ストリーミング検証を有効化します。デフォルトは `true` です。ストリーミングモードでは、メモリに保管されるデータが最小化され、処理がより速くなります。欠点は、後に情報が必要になる可能性があります。例えば、XML インスタンスドキュメントのデータモデルが使用できない場合があります。この様なシチュエーションでは、ストリーミングモードは `--streaming` に `false` の値を与え、)オフにされる必要があります。 `valxml-withxsd` コマンドを使用して、`--script` オプションを使用するとストリーミングを無効化することができます。`--streaming` オプションは、`--parallel-assessment` が `true` に設定されている場合、の場合無視されます。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xinclude

  --xinclude = `true|false`
  XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は `false` です。 `false` の場合、XInclude の `include` 要素は無視されます。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xml-mode

  --xml-mode = `wf|id|valid`
  XML インスタンスドキュメントのために使用される XML 処理モードを指定します: `wf`=整形式のチェック, `id`=ID/IDREF チェックと共に整形式のチェック, `valid`=検証。デフォルト値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-mode-for-schemas

  --xml-mode-for-schemas = `wf|id|valid`
  XML スキーマドキュメントのために使用される XML 処理モードを指定します。: `wf`=整形式のチェック, `id`=ID/IDREF チェックを持つ整形式, `valid`=検証。デフォルトの値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xsd

  --xsd = *FILE*
  1 つまたは複数の XML スキーマドキュメント を XML インスタンスの検証に使用することを指定します。1 つ以上のスキーマドキュメントを指定するため、オプションを複数回追加します。

- xsd-version

  --xsd-version = `1.0|1.1|detect`
  使用する W3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

- カタログとグローバルリソース

  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値は インストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *FILE*
  ルートカタログに追加して使用されるXML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXML フォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - log-output

    --log-output = *FILE*
    指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

  - network-timeout

--network-timeout = *VALUE*
リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.1.3 valdtd (dtd)

valdtd | dtd コマンドは1つまたは複数の DTD ドキュメントを XML 1.0 または XML 1.1 に従い検証します。

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

- *InputFile* 引数は検証する DTD ドキュメントです。
- 複数のドキュメントを検証するには、以下を行います: (i) 各ファイルを空白で区切り、CLI で検証されるファイルをリストします。または (ii) 検証されるファイルをテキストファイル (.txt file) でファイル名を各ラインに表示し、リストします。 そして、このテキストファイルを *InputFile* 引数として、true と設定された --listfile 186 オプションと共に返します。(*下のオプションのリストを参照してください*)。

サンプル

valdtd コマンドのサンプル:

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

- コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) は Windows と Mac のみで使用することができます。
* Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 検証と処理

▼ listfile

--listfile = true|false
true の場合、コマンドの *InputFile* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は false です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) --listfile オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ recurse

--recurse = true|false
ZIP アーカイブ内のファイルを選択するために使用されます。true の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: test.zip|zip\test.xml は test.xml という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。* および ? などのワイルドカード文字が使用されるかもしれません。ですから *.xml は、Zip フォルダー内のすべての .xml ファイルを選択します。オプションのデフォルト値は false です。
メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ script

--script = *FILE*
検証が完了した後、提出されたファイル内の Python スクリプトを実行します。1 つ以上のスクリプトを指定するため、オプションを複数回追加します。

▼ script-api-version

--api, --script-api-version = 1; 2; 2.1; 2.2; 2.3; 2.4; 2.4.1; 2.5; 2.6; 2.7; 2.8; 2.8.1; 2.8.2

スクリプトで使用される Python API バージョンを指定します。デフォルト値は現在 `2.8.2` である最新バージョンです。`1` および `2` の値の代わりに、それぞれ値 `1.0` および `2.0` を使用することができます。

- script-output

  --script-output = *FILE*
  FILE という名前のファイルにスクリプトの標準出力を書き込みます。

- script-param

  --script-param = *KEY:VALUE*
  Python スクリプト実行中にアクセスすることのできる、追加ユーザー指定パラメーター。1 つ以上のスクリプトパラメーターを指定するため、オプションを複数回追加します。

- カタログとグローバルリソース
  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = `true|false`
    グローバルリソース 49 を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - globalresourceconfig [gc]

    --gc | --globalresourceconfig = *VALUE*
    グローバルリソースのアクティブな構成 49 を指定します (グローバルリソース 49 を有効化します) 49。

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きの XML フォーマットを生成します。

- error-limit

  --error-limit = *N | unlimited*
  エラー制限を指定します。デフォルト値は100 です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は100 です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、 val any --h。(または、 help コマンドは引数と共に使用することができます。例: hel p val any。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は100 です。

4.1.4 valxsd (xsd)

valxsd | xsd コマンドは、1 つまたは複数のを W 3C XML スキーマ定義言語 (XSD) 1.0 または1.1 に従い、スキーマドキュメント (XSD ドキュメント) を検証します。XML スキーマ仕様 に対して検証されるのは XML スキーマ自身であり、に対する XML インスタンスドキュメントではない点に注意してください。

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

- *InputFile* 引数は検証する XML スキーマドキュメントです。 --xsd-version=1.0|1.1|detect 188 オプションは検証する XSD バージョンを指定します。デフォルトは 1.0 です。
- 複数のドキュメントを検証するには、以下を行います: (i)各ファイルを空白で区切り、CLI で検証されるファイルをリストする。または(ii) 検証されるファイルをテキストファイル(.txt file) でファイル名を各ラインに表示し、リストします。そして、このテキストファイルを *InputFile* 引数として、true と設定された --listfile 186 オプションと共に返します。(*下のオプションのリストを参照してください*)。

サンプル

valxsd コマンドのサンプル:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム(W indows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) は W indows と Mac のみで使用することができます。
＊Linux と Mac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のためにつまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 検証と処理

▼ ct-restrict-mode

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

複合型の制約のチェック方法を指定します。`1.0` の値は、XSD 1.1 検証モードであっても、XSD 1.0 仕様に従い複合型の制約をチェックします。`1.1` の値は、XSD 1.0 検証モードであっても、XSD 1.1 仕様に従い複合型の制約をチェックします。**`default`** の値は、現在の検証モード (1.0 または1.1) 内の仕様に従い複合型を検証します。デフォルトの値は **`default`** です。

listfile

--listfile = `true|false`
`true` の場合、コマンドの *`InputFile`* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

recurse

--recurse = `true|false`
ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *`InputFile`* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は`test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションの デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

schema-imports

--schema-imports = `load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only`
それぞれオプションの`namespace` 属性とオプションの`schemaLocation` 属性を持つ、`xs:import` 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL">`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: `load-preferring-schemalocation`.
振る舞いは以下の通りです:

- `load-by-schemalocation`: カタログマッピング[40] を考慮し、`schemaLocation` 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
- `load-preferring-schemalocation`: `schemaLocation` 属性が存在する場合、 カタログマッピング[40] を考慮して使用されます。`schemaLocation` 属性が存在しない場合、`namespace` 属性の値がカタログマッピング[40]を介して使用されます。スキーマこれはデフォルトの値です。
- `load-by-namespace`: カタログマッピング[40]を介して、`namespace` 属性の値がスキーマを検索するために使用されます。
- `load-combining-both`: `namespace` または`schemaLocation` 属性にカタログマッピング[40] がある場合、マッピング が使用されます。両方にカタログマッピング[40], がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション[188]) がどのマッピングが使用されるか決定します。カタログマッピング[40] が存在しない場合、`schemaLocation` 属性が使用されます。
- `license-namespace-only`: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
`xsi:schemaLocation` および`xsi:noNamespaceSchemaLocation` 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: `load-by-schemalocation`。

- `load-by-schemalocation` 値はXML インスタンスドキュメント内の`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性 の スキーマの場所のURL 317 使用します。これはデフォルト の値です。
- `load-by-namespace` 値は、`xsi:schemaLocation` の名前空間の部分 317 を、そして `xsi:noNamespaceSchemaLocation` の場合は空の文字列を取ります。カタログマッピング 40 を介してスキーマを検索します。
- `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が `ignore` の場合、`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(`--schemalocation-hints` または `--schema-imports` オプションが `load-combining-both` の値を有する場合、また、関連する名前空間とURL のペアが双方カタログマッピング 40 を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。`prefer-schemalocation` 値はURL マッピングを参照します)。) デフォルトは `prefer-schemalocation` です。

script

--script = *`FILE`*
検証が完了した後、提出されたファイル内のPython スクリプトを実行します。1 つ以上のスクリプトを指定するため、オプションを複数回追加します。

script-api-version

`--api, --script-api-version =` `1; 2; 2.1; 2.2; 2.3; 2.4; 2.4.1; 2.5; 2.6; 2.7; 2.8; 2.8.1; 2.8.2`
スクリプトで使用されるPython API バージョンを指定します。デフォルト値は現在 **`2.8.2`** である最新バージョンです。`1` および `2` の値の代わりに、それぞれ値 `1.0` および `2.0` を使用することができます。

script-output

--script-output = *`FILE`*
`FILE` という名前のファイルにスクリプトの標準出力を書き込みます。

script-param

--script-param = *`KEY:VALUE`*
Python スクリプト実行中にアクセスすることのできる、追加ユーザー指定パラメーター。1 つ以上のスクリプトパラメーターを指定するため、オプションを複数回追加します。

xinclude

--xinclude = `true|false`
XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は `false` です。 `false` の場合、XInclude の `include` 要素は無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

xml-mode

--xml-mode = `wf|id|valid`
XML インスタンスドキュメントのために使用される XML 処理モードを指定します: `wf`=整形式のチェック; `id`=ID/IDREF チェックと共に整形式のチェック; `valid`=検証。デフォルト値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
XML スキーマドキュメントのために使用される XML 処理モードを指定します。: `wf`=整形式のチェック; `id`=ID/IDREF チェックを持つ整形式; `valid`=検証。デフォルトの値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xsd-version

--xsd-version = `1.0|1.1|detect`
使用する W 3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

- カタログとグローバルリソース
  - catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

--user-catalog = *FILE*
ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース 49 を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

--gc | --globalresourceconfig = *VALUE*
グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

--gr | --globalresourcefile = *FILE*
グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

  --error-format = `text|shortxml|longxml`
  エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

- error-limit

  --error-limit = *N | unlimited*
  エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

- verbose

  --verbose = `true|false`
  `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  `1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.2 整形式チェック コマンド

整形式チェックコマンドはXMLドキュメントおよびDTD の整形式をチェックする際にも使用されます。これらのコマンドは下にリストされており、このセクションのサブセクションで詳しく述べられています:

- wfxml 72: XMLドキュメントの整形式をチェックします。
- wfdtd 75: DTD の整形式をチェックします。
- wfany 78: XMLドキュメントまたはDTD の整形式をチェックします。型は自動的に検出されます。

4.2.1 wfxml

wfxml コマンドは、XML 1.0 またはXML 1.1 仕様に従い1 つ以上のXMLドキュメントの整形式をチェックします。

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

- *InputFile* 引数は整形式をチェックするXMLドキュメントです。
- 複数のドキュメントをチェックする場合は、以下を行います: (i) 各ファイルを空白で区切り、CLI でチェックされるファイルをリストします。または(ii) チェックインするファイルをテキストファイル(.txt ファイル) でリストし、ファイル名を各ラインに表示します。そして、このテキストファイルを*InputFile* 引数として、true と設定された--listfile 186 オプションと共に返します。(*下のオプションのリストを参照してください*)。

サンプル

wfxml コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: `--option=value`。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。`--h, --help` オプションを使用してコマンドに関する情報を表示します。

- 検証と処理
  - dtd

    --dtd = *FILE*
    検証に使用される外部 DTD ドキュメントを指定します。XML ドキュメント内に外部 DTD への参照が存在する場合、CLI オプションが外部参照を上書きします。

  - listfile

    --listfile = `true|false`
    `true` の場合、コマンドの *InputFile* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - namespaces

    --namespaces = `true|false`
    名前空間対応処理を有効化します。これは、XML インスタンス内の間違った名前空間のため発生するエラーをチェックするために役に立ちます。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - recurse

    --recurse = `true|false`
    ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - streaming

    --streaming = `true|false`
    ストリーミング検証を有効化します。デフォルトは `true` です。ストリーミングモードでは、メモリに保管されるデータが最小化され、処理がより速くなります。欠点は、後に情報が必要になる可能性があります。例えば、XML インスタンスドキュメントのデータモデルが使用できない場合があります。この様なシチュエーションでは、ストリーミングモードは(`--streaming` に `false` の値を与え、)オフにされる必要があります。 `valxml-withxsd` コマンドを使用して、`--script` オプションを使用するとストリーミングを無効化することができます。`--streaming` オプションは、`--parallel-assessment` が `true` に設定されている場合、の場合無視されます。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- カタログとグローバルリソース

- catalog

  --catalog = *FILE*
  インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス(`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`)です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *FILE*
  ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン

  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きの XML フォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は `100` です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は100 です。

4.2.2 wfdtd

wfdtd コマンドは、XML 1.0 またはXML 1.1 仕様に従い1 つ以上のDTD ドキュメントの整形式をチェックします。

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

- *InputFile* 引数は整形式をチェックするDTD ドキュメントです。
- 複数のドキュメントをチェックする場合は、以下を行います: (i) 各ファイルを空白で区切り、CLI でチェックされるファイルをリストします。または(ii) チェックインするファイルをテキストファイル(.txt ファイル)でリストし、ファイル名を各ラインに表示します。そして、このテキストファイルを *InputFile* 引数として、true と設定された --listfile 186 オプションと共に返します。(*下のオプションのリストを参照してください*)。

サンプル

wfdtd コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、またはは、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1 つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 検証と処理

▼ listfile

--listfile = true|false
true の場合、コマンドの *InputFile* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値はfalse です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) --listfile オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ recurse

--recurse = true|false
ZIP アーカイブ内のファイルを選択するために使用されます。true の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: test.zip|zip\test.xml はtest.xml という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。* および ? などのワイルドカード文字が使用されるかもしれません。ですから*.xml は、Zip フォルダー内のすべての .xml ファイルを選択します。オプションの デフォルト値はfalse です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ カタログとグローバルリソース

▼ catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス(`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`)です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *FILE*
  ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きの XML フォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は `100` です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - log-output

--log-output = *FILE*
指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.2.3 wfany

wfany コマンドは対応する仕様に従いDTD およびXML ドキュメントの整形式をチェックします。 ドキュメントの型は自動的に検出されます。

```
raptorxml wfany [options] InputFile
```

- *InputFile* 引数はドキュメントの整形式のチェックです。
- コマンドの引数としては1 つのドキュメントのみしか提出できないことご注意してください。 提出されたドキュメントの型は自動的に検出されます。

サンプル

wfany コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) は Windows と Mac のみで使用することができます。
＊Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字が文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 処理

▼ listfile

--listfile = true|false
true の場合、コマンドの *InputFile* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は false です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) --listfile オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ カタログとグローバルリソース

▼ catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値は インストールされたルートカタログファイルへの絶対パス (<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

▼ user-catalog

--user-catalog = *FILE*
ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

▼ enable-globalresources

--enable-globalresources = true|false
グローバルリソース 49 を無効化します。デフォルト値は false です。

メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソース ファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は `100` です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - log-output

    --log-output = *FILE*
    指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

  - network-timeout

    --network-timeout = *VALUE*
    リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

  - verbose

    --verbose = `true|false`
    `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - verbose-output

--verbose-output = *FILE*
FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  `1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.3 XQuery コマンド

XQuery コマンド:

- xquery 82: XQuery ドキュメントの実行のため、オプションとして入力ドキュメントを持つことができます。
- xqueryupdate 88: XQuery アップデートを実行するため、XQuery ドキュメントと、オプションでアップデートする入力ドキュメントを使用します。
- valxquery 95: XQuery ドキュメントの検証のため
- valxqueryupdate 101: XQuery (アップデート) ドキュメントの検証のため

4.3.1 xquery

xquery コマンドは XQuery ファイルを単一の引数としてとり、出力ファイルを生成するためにオプションの入力ファイルを使用して実行します。入力と出力ファイルはオプションとして指定されます。

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

- 引数 ***XQuery-File*** は、実行される XQuery ファイルのパスと名前です。
- XQuery 1.0 または 3.0 以下を使用することができます。デフォルトで XQuery 3.0 が使用されます。

サンプル

xquery コマンドのサンプルです:

- **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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

* 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
* Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: `--option=value`。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**`--h, --help`** オプションを使用してコマンドに関する情報を表示します。

- XQuery 処理
  - indent-characters

    --indent-characters = *`VALUE`*
    インデントとして使用される文字列を指定します。

  - input

    --input = *`FILE`*
    変換されるXML ファイルのURL です。

  - omit-xml-declaration

    --omit-xml-declaration = `true|false`
    シリアル化 オプションはXML 宣言が出力から省略されるかどうかを指定します。`true` の場合、出力ドキュメント内にXML 宣言はありません。`false` の場合、XML 宣言は含まれます。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - output, xsltoutput

    output = *`FILE,`* xsltoutput = *`FILE`*
    プライマリ出力ファイルのURL。例えば、複数のファイルのHTML 出力の場合、プライマリ出力のファイルは、エントリポイントHTML ファイルの場所になります。イメージファイルなどの、追加出力ファイルは、`xslt-additional-output-files` として報告されます。`--output` または `--xsltoutput` オプションが指定されていない場合、出力は標準の出力に書き込まれます。

  - output-encoding

    --output-encoding = *`VALUE`*
    出力ドキュメント内のエンコード属性の値。有効な値はIANA 文字セットレジストリ内の名前です。デフォルト値は `UTF-8` です。

  - output-indent

    --output-indent = `true|false`
    `true` の場合、出力は階層的構造に従いインデントされます。`false` の場合、階層形式のインデントはありません。デフォルトは `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - output-method

    --output-method = `xml|html|xhtml|text`
    出力フォーマットを指定します。デフォルト値は `xml` です。

  - param [p]

    --p | --param = *`KEY:VALUE`*
    ⊟ ***XQuery***

外部パラメーターの値を指定します。内部パラメータは、`declare variable` を持ち、変数名、`external` キーワード、およびセミコロンが続く、XQuery ドキュメント内で宣言されています。例:

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

`external` キーワードである `$foo` は、外部パラメータとなり、その値は、ランタイム時に外部ソースからパスされます。外部パラメーターは、CLI コマンドにより値を与えられます。例:

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

上で説明されているように、*KEY* は外部パラメータの名前です。*VALUE* は、XPath 式として与えられた外部パラメーターの値です。の値です。 CLI で使用されたパラメーター名は、XQuery ドキュメント内で宣言される必要があります。複数の外部パラメータがCLI がでパスされた値である場合、それぞれに個別の `--param` オプションが必要になります。 XPath 式にスペースが含まれる場合二重引用符を使用する必要があります。

XSLT

グローバルスタイルシートのパラメーターを指定します。*KEY* はパラメータ名であり、*VALUE* は、パラメーターの値を与える XPath 式です。CLI で使用されるパラメーター名は、スタイルシート内で宣言される必要があります。複数のパラメータが使用される場合、`--param` スイッチが各パラメータ前に使用される必要があります。XPath 式内または式内の文字列でスペースが含まれる場合は、二重引用符が、XPath 式の周りで使用される必要があります。*例:*

```
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`

`true` の場合、警告を静的 XPath コンテキスト内で検知される型エラーにします。エラーは実行の失敗を引き起こしますが、警告は処理を継続することができます。デフォルトは、`false` です。

xquery-version

--xquery-version = `1|1.0|3|3.0|3.1`

XSLT プロセッサがXSLT 1.0 またはXSLT 3.0. を使用するか指定します。デフォルト値は 3.1 です。

XML スキーマとXML インスタンス

load-xml-with-psvi

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

入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: `true`。

schema-imports

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

それぞれオプションの `namespace` 属性とオプションの `schemaLocation` 属性を持つ、`xs:import` 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL">`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: `load-preferring-schemalocation`.
振る舞いは以下の通りです:

- `load-by-schemalocation`: カタログマッピング 40 を考慮し、`schemaLocation` 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
- `load-preferring-schemalocation`: `schemaLocation` 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。`schemaLocation` 属性が存在しない場合、`namespace` 属性の値がカタログマッピン

グ[40]を介して使用されます。スキーマこれはデフォルトの値です。

- load-by-namespace: カタログマッピング[40]を介して、namespace 属性の値がスキーマを検索するために使用されます。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング[40]がある場合、マッピングが使用されます。両方にカタログマッピング[40]，がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション[188]) がどのマッピングが使用されるか決定します。カタログマッピング[40] が存在しない場合、schemaLocation 属性が使用されます。
- license-namespace-only: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: load-by-schemalocation。

- load-by-schemalocation 値は XML インスタンスドキュメント内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所の URL [317] 使用します。これはデフォルトの値です。
- load-by-namespace 値は、xsi:schemaLocation の名前空間の部分[317]を、そして xsi:noNamespaceSchemaLocation の場合は空の文字列を取ります。カタログマッピング[40]を介してスキーマを検索します。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング[40]がある場合、マッピングが使用されます。両方にカタログマッピング[40]，がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション[188]) がどのマッピングが使用されるか決定します。名前空間または URL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が ignore の場合、xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(--schemalocation-hints または --schema-imports オプションが load-combining-both の値を有する場合、また、関連する名前空間と URL のペアが双方カタログマッピング[40]を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。prefer-schemalocation 値は URL マッピングを参照します)。) デフォルトは prefer-schemalocation です。

xinclude

--xinclude = `true|false`
XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は false です。 false の場合、XInclude の include 要素は無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

xml-mode

--xml-mode = `wf|id|valid`
XML インスタンスドキュメントのために使用される XML 処理モードを指定します: wf=整形式のチェック, id=ID/IDREF チェックと共に整形式のチェック, valid=検証。デフォルト値は wf です。valid の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
XML スキーマドキュメントのために使用される XML 処理モードを指定します。: wf=整形式のチェック, id=ID/IDREF チェッ

クを持つ整形式; valid=検証。デフォルトの値はwf です。valid の値はDTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = true|false
  検証エラーを警告として扱うかを指定します。警告として扱われる場合、エラーが検出されても、XSLT 変換などの追加処理は継続されます。デフォルトはfalse です。

- xsd

  --xsd = *FILE*
  1つまたは複数のXML スキーマドキュメント をXML インスタンスの検証に使用することを指定します。1 つ以上のスキーマドキュメントを指定するため、オプションを複数回追加します。

- xsd-version

  --xsd-version = 1.0|1.1|detect
  使用するW 3C スキーマ定義言語(XSD) のバージョンを指定します。デフォルト 1.0 はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン(1.0 または1.1) は、ドキュメントの<xs:schema> 要素のvc:minVersion 属性の値を読み込むことにより検出されます。@vc:minVersion 属性の値が1.1 の場合、スキーマはバージョン1.1 として検出されます。他の値に関しては、または、@vc:minVersion 属性が不在の場合、スキーマはバージョン1.0 として検出されます。

- カタログとグローバルリソース
  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス(<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用されるXML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = true|false
    グローバルリソース 49 を無効化します。デフォルト値はfalse です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - globalresourceconfig [gc]

    --gc | --globalresourceconfig = *VALUE*
    グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*

グローバルリソースファイル[49] を指定します。(グローバルリソース[49]を有効化します)[49]。

- 拡張子

これらのオプションは、(XMLSpy Enterprise エディションなど) Enterprise レベルのAltova 製品の多くで使用することができ、特別な拡張関数を扱うオプションを定義します。使用方法はこれらの製品のユーザーマニュアルで説明されています。

- chartext-disable

--chartext-disable = `true|false`
チャート拡張子を無効化します。デフォルト値は`false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- dotnetext-disable

--dotnetext-disable = `true|false`
.NET 拡張子を無効化します。デフォルト値は`false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- jvm-location

--jvm-location = *`FILE`*
`FILE` はJava 仮想マシン(W indows のDLL、Linux 上の共有されるオブジェクト) の場所を指定します。
XSLT/XQuery コード内で Java 拡張関数[412] を使用する場合、JVMが必要になります。デフォルトは`false` です。

- javaext-barcode-location

--javaext-barcode-location = *`FILE`*
バーコード拡張ファイル`AltovaBarcodeExtension.jar` を含むパスを指定します。.パスは次のフォームで与えられなければなりません

- ファイルURI の例: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"`
- バックスラッシュ付きのエスケープ文字を使用したW indows パスの例: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"`

- javaext-disable

--javaext-disable = `true|false`
Java 拡張子を無効化します。デフォルト値は`false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

--error-format = `text|shortxml|longxml`
エラー出力のフォーマットを指定します。デフォルト値は`text` です。他のオプションは`longxml` と共に詳細付きのXML フォーマットを生成します。

- error-limit

--error-limit = *`N | unlimited`*
エラー制限を指定します。デフォルト値は`100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、val any --h。(または、help コマンドは引数と共に使用することができます。例: help val any。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.3.2 xqueryupdate

xqueryupdate コマンドは XQuery ファイルを単一引数として取り、任意の入力ファイルと共に実行しアップデートされた出力ファイルを作成します。入力および出力ファイルはオプションとして指定されます。

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

- 引数 ***XQuery-File*** は、実行される XQuery ファイルのパスと名前です。
- XQuery Update 1.0 または 3.0 のどちらが使用されるかを指定することができます。デフォルトで XQuery Update 3.0 が使用されます。

サンプル

xqueryupdate コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
＊Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字がが文字列内で発生するとき引用符を使用してください。例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ XQuery Update 処理

▼ indent-characters

--indent-characters = *VALUE*
インデントとして使用される文字列を指定します。

▼ input

--input = *FILE*
変換される XML ファイルの URL です。

- omit-xml-declaration

  --omit-xml-declaration = `true|false`
  シリアル化 オプションはXML 宣言が出力から省略されるかどうかを指定します。`true` の場合、出力ドキュメント内にXML 宣言はありません。`false` の場合、XML 宣言は含まれます。デフォルト値は`false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- output, xsltoutput

  output = *`FILE`*, xsltoutput = *`FILE`*
  プライマリ出力ファイルのURL。例えば、複数のファイルのHTML 出力の場合、プライマリ出力のファイルは、エントリポイントHTML ファイルの場所になります。イメージファイルなどの、追加出力ファイルは、`xslt-additional-output-files` として報告されます。`--output` または`--xsltoutput` オプションが指定されていない場合、出力は標準の出力に書き込まれます。

- output-encoding

  --output-encoding = *`VALUE`*
  出力ドキュメント内のエンコード属性の値。有効な値はIANA 文字セットレジストリ内の名前です。デフォルト値は`UTF-8` です。

- output-indent

  --output-indent = `true|false`
  `true` の場合、出力は階層的構造に従いインデントされます。`false` の場合、階層形式のインデントはありません。デフォルトは`false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- output-method

  --output-method = `xml|html|xhtml|text`
  出力フォーマットを指定します。デフォルト値は`xml` です。

- param [p]

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

  XQuery

  外部パラメーターの値を指定します。内部パラメータは、`declare variable` を持ち、変数名、`external` キーワード、およびセミコロンが続く、XQuery ドキュメント内で宣言されています。例:

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

  `external` キーワードである`$foo` は、外部パラメータとなり、その値は、ランタイム時に外部ソースからパスされます。外部パラメーターは、CLI コマンドにより値を与えられます。例:

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

  上で説明されているように、*`KEY`* は外部パラメータの名前です。*`VALUE`* は、XPath 式として与えられた外部パラメーターの値です。の値です。 CLI で使用されたパラメーター名は、XQuery ドキュメント内で宣言される必要があります。複数の外部パラメータがCLI がでパスされた値である場合、それぞれに個別の`--param` オプションが必要になります。XPath 式にスペースが含まれる場合二重引用符を使用する必要があります。

  XSLT

  グローバルスタイルシートのパラメーターを指定します。*`KEY`* はパラメータ名であり、*`VALUE`* は、パラメーターの値を与えるXPath 式です。CLI で使用されるパラメーター名は、スタイルシート内で宣言される必要があります。複数のパラメータが使用される場合、`--param` スイッチが各パラメータ前に使用される必要があります。XPath 式内または式内の文字列でスペースが含まれる場合は、二重引用符が、XPath 式の周りで使用される必要があります。*例:*

```
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
  true の場合、警告を静的 XPath コンテキスト内で検知される型エラーにします。エラーは実行の失敗を引き起こしますが、警告は処理を継続することができます。デフォルトは、false です。

- xquery-update-version

  --xquery-update-version = 1|1.0|3|3.0|
  XQuery プロセッサが XQuery Update Facility 1.0 または XQuery Update Facility 3.0 を使用するかを指定します。デフォルト値は 3 です。

- keep-formatting

  --keep-formatting = true|false
  ターゲットドキュメントのフォーマットを最大範囲可能な限り保持します。デフォルト: true。

- updated-xml

  --updated-xml = discard|writeback|asmainresult
  更新された XML ファイルがどのように扱われるかを指定します。アップデートは以下のとおりです:

  - 破棄されたまたはファイルに書き込まれない (discard)
  - --input オプションにより指定されている入力 XML ファイルに書き込まれる (writeback)
  - 標準の場所または --output オプションにより指定されている場所に保存される (定義されている場合)

  デフォルト: discard。

- XML スキーマと XML インスタンス
  - load-xml-with-psvi

    --load-xml-with-psvi = true|false
    入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: true。

  - schema-imports

    --schema-imports = load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only
    それぞれオプションの namespace 属性とオプションの schemaLocation 属性を持つ、xs:import 要素の振る舞いを指定します: <import namespace="someNS" schemaLocation="someURL">。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: load-preferring-schemalocation.
    振る舞いは以下の通りです:

    - load-by-schemalocation: カタログマッピング 40 を考慮し、schemaLocation 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
    - load-preferring-schemalocation: schemaLocation 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。schemaLocation 属性が存在しない場合、namespace 属性の値がカタログマッピング 40 を介して使用されます。スキーマこれはデフォルトの値です。

- load-by-namespace: カタログマッピング[40]を介して、namespace 属性の値がスキーマを検索するために使用されます。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング[40]がある場合、マッピングが使用されます。両方にカタログマッピング[40]がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション[188])がどのマッピングが使用されるか決定します。カタログマッピング[40]が存在しない場合、schemaLocation 属性が使用されます。
- license-namespace-only: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: load-by-schemalocation。

- load-by-schemalocation 値は XML インスタンスドキュメント内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所の URL [317] 使用します。これはデフォルトの値です。
- load-by-namespace 値は、xsi:schemaLocation の名前空間の部分[317]を、そして xsi:noNamespaceSchemaLocation の場合は空の文字列を取ります。カタログマッピング[40]を介してスキーマを検索します。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング[40]がある場合、マッピングが使用されます。両方にカタログマッピング[40]がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション[188])がどのマッピングが使用されるか決定します。名前空間または URL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が ignore の場合、xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(--schemalocation-hints または --schema-imports オプションが load-combining-both の値を有する場合、また、関連する名前空間と URL のペアが双方カタログマッピング[40]を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します(名前空間 マッピング または URL マッピング。prefer-schemalocation 値は URL マッピングを参照します)。) デフォルトは prefer-schemalocation です。

xinclude

--xinclude = `true|false`
XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は false です。 false の場合、XInclude の include 要素は無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

xml-mode

--xml-mode = `wf|id|valid`
XML インスタンスドキュメントのために使用される XML 処理モードを指定します: wf=整形式のチェック; id=ID/IDREF チェックと共に整形式のチェック; valid=検証。デフォルト値は wf です。valid の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
XML スキーマドキュメントのために使用される XML 処理モードを指定します。: wf=整形式のチェック; id=ID/IDREF チェックを持つ整形式; valid=検証。デフォルトの値は wf です。valid の値は DTD の参照の処理中に各スキーマドキュメント

がロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = `true|false`
  検証エラーを警告として扱うかを指定します。警告として扱われる場合、エラーが検出されても、XSLT 変換などの追加処理は継続されます。デフォルトは `false` です。

- xsd

  --xsd = *FILE*
  1 つまたは複数の XML スキーマドキュメント を XML インスタンスの検証に使用することを指定します。1 つ以上のスキーマドキュメントを指定するため、オプションを複数回追加します。

- xsd-version

  --xsd-version = `1.0|1.1|detect`
  使用する W3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

- カタログとグローバルリソース

  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = `true|false`
    グローバルリソース 49 を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - globalresourceconfig [gc]

    --gc | --globalresourceconfig = *VALUE*
    グローバルリソースのアクティブな構成 49 を指定します (グローバルリソース 49 を有効化します) 49。

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49。

拡張子

これらのオプションは、(XMLSpy Enterprise エディションなど) Enterprise レベルの Altova 製品の多くで使用することができ、特別な拡張関数を扱うオプションを定義します。使用方法はこれらの製品のユーザーマニュアルで説明されています。

chartext-disable

--chartext-disable = `true|false`
チャート拡張子を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

dotnetext-disable

--dotnetext-disable = `true|false`
.NET 拡張子を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

jvm-location

--jvm-location = *`FILE`*
`FILE` はJava 仮想マシン (Windows のDLL、Linux 上の共有されるオブジェクト) の場所を指定します。
XSLT/XQuery コード内で Java 拡張関数 412 を使用する場合、JVMが必要になります。デフォルトは `false` です。

javaext-barcode-location

--javaext-barcode-location = *`FILE`*
バーコード拡張ファイル `AltovaBarcodeExtension.jar` を含むパスを指定します。.パスは次のフォームで与えられなければなりません

- ファイルURI の例: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"`
- バックスラッシュ付きのエスケープ文字を使用したWindows パスの例: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"`

javaext-disable

--javaext-disable = `true|false`
Java 拡張子を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

error-format

--error-format = `text|shortxml|longxml`
エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXML フォーマットを生成します。

error-limit

--error-limit = *`N | unlimited`*
エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

(C) 2015-2021 Altova GmbH

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、val any --h。(または、help コマンドは引数と共に使用することができます。例: help val any。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.3.3 valxquery

valxquery コマンドはXQuery ファイルを単一の引数として取り、検証します。

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

- ***XQuery-File*** 引数は、実行される XQuery ファイルのパスと名前です。

サンプル

valxquery コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1 つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ XQuery 処理

▼ omit-xml-declaration

--omit-xml-declaration = true|false
シリアル化 オプションはXML 宣言が出力から省略されるかどうかを指定します。true の場合、出力ドキュメント内にXML 宣言はありません。false の場合、XML 宣言は含まれます。デフォルト値はfalse です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ xquery-version

--xquery-version = 1|1.0|3|3.0|3.1
XSLT プロセッサがXSLT 1.0 またはXSLT 3.0. を使用するか指定します。デフォルト値は 3.1 です。

▼ XML スキーマとXML インスタンス

▼ load-xml-with-psvi

--load-xml-with-psvi = true|false
入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: true。

schema-imports

--schema-imports = `load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only`
それぞれオプションの namespace 属性とオプションの schemaLocation 属性を持つ、xs:import 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL">`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: load-preferring-schemalocation.
振る舞いは以下の通りです:

- load-by-schemalocation: カタログマッピング[40] を考慮し、schemaLocation 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
- load-preferring-schemalocation: schemaLocation 属性が存在する場合、 カタログマッピング[40] を考慮して使用されます。schemaLocation 属性が存在しない場合、namespace 属性の値がカタログマッピング[40]を介して使用されます。スキーマこれはデフォルトの値です。
- load-by-namespace: カタログマッピング[40]を介して、namespace 属性の値がスキーマを検索するために使用されます。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング[40]がある場合、マッピング が使用されます。両方にカタログマッピング[40], がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション[188]) がどのマッピングが使用されるか決定します。カタログマッピング[40] が存在しない場合、schemaLocation 属性が使用されます。
- license-namespace-only: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: load-by-schemalocation。

- load-by-schemalocation 値はXML インスタンスドキュメント内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所のURL [317] 使用します。これはデフォルトの値です。
- load-by-namespace 値は、xsi:schemaLocation の名前空間の部分[317]を、そして xsi:noNamespaceSchemaLocation の場合は空の文字列を取ります。カタログマッピング[40]を介してスキーマを検索します。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング[40]がある場合、マッピング が使用されます。両方にカタログマッピング[40], がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション[188]) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が ignore の場合、xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(--schemalocation-hints または --schema-imports オプションが load-combining-both の値を有する場合、また、関連する名前空間とURL のペアが双方カタログマッピング[40]を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。prefer-schemalocation 値はURL マッピングを参照します)。) デフォルトは prefer-schemalocation です。

xinclude

--xinclude = `true|false`
XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は `false` です。 `false` の場合、XInclude の `include` 要素は無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xml-mode

--xml-mode = `wf|id|valid`
XML インスタンスドキュメントのために使用される XML 処理モードを指定します: `wf`=整形式のチェック; `id`=ID/IDREF チェックと共に整形式のチェック; `valid`=検証。デフォルト値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
XML スキーマドキュメントのために使用される XML 処理モードを指定します。: `wf`=整形式のチェック; `id`=ID/IDREF チェックを持つ整形式; `valid`=検証。デフォルトの値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xsd-version

--xsd-version = `1.0|1.1|detect`
使用する W3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

- カタログとグローバルリソース
  - catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

--user-catalog = *FILE*
ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース 49 を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

--gc | --globalresourceconfig = *VALUE*
グローバルリソースのアクティブな構成 49 を指定します (グローバルリソース 49 を有効化します) 49。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル[49] を指定します。(グローバルリソース[49] を有効化します)[49]。

- 拡張子

  これらのオプションは、(XMLSpy Enterprise エディションなど) Enterprise レベルのAltova 製品の多くで使用することができ、特別な拡張関数を扱うオプションを定義します。使用方法はこれらの製品のユーザーマニュアルで説明されています。

  - chartext-disable

    --chartext-disable = `true|false`
    チャート拡張子を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - dotnetext-disable

    --dotnetext-disable = `true|false`
    .NET 拡張子を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - jvm-location

    --jvm-location = *FILE*
    FILE はJava 仮想マシン(Windows のDLL、Linux 上の共有されるオブジェクト) の場所を指定します。
    XSLT/XQuery コード内で Java 拡張関数[412] を使用する場合、JVMが必要になります。デフォルトは `false` です。

  - javaext-barcode-location

    --javaext-barcode-location = *FILE*
    バーコード拡張ファイル `AltovaBarcodeExtension.jar` を含むパスを指定します。.パスは次のフォームで与えられなければなりません

    - ファイルURI の例: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"`
    - バックスラッシュ付きのエスケープ文字を使用したWindows パスの例: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"`

  - javaext-disable

    --javaext-disable = `true|false`
    Java 拡張子を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン

  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXML フォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*

エラー制限を指定します。デフォルト値は100 です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は100 です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、val any --h。(または、help コマンドは引数と共に使用することができます。例: help val any。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は100 です。

4.3.4 valxqueryupdate

valxqueryupdate コマンドはXQuery ファイルを単一の引数として取り、検証します。

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

- ***XQuery-File*** 引数は、実行されるXQuery ファイルのパスと名前です。

サンプル

valxqueryupdate コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、またはフォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ XQuery 処理

▼ omit-xml-declaration

--omit-xml-declaration = true|false
シリアル化 オプションはXML 宣言が出力から省略されるかどうかを指定します。true の場合、出力ドキュメント内にXML 宣言はありません。false の場合、XML 宣言は含まれます。デフォルト値はfalse です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

▼ xquery-update-version

--xquery-update-version = 1|1.0|3|3.0|
XQuery プロセッサがXQuery Update Facility 1.0 またはXQuery Update Facility 3.0 を使用するかを指定しま

す。デフォルト値は 3 です。

- XML スキーマとXML インスタンス
  - load-xml-with-psvi

    --load-xml-with-psvi = `true|false`
    入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: `true`。

  - schema-imports

    --schema-imports = `load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only`
    それぞれオプションの `namespace` 属性とオプションの `schemaLocation` 属性を持つ、`xs:import` 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL"/>`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: `load-preferring-schemalocation`.
    振る舞いは以下の通りです:

    - `load-by-schemalocation`: カタログマッピング 40 を考慮し、`schemaLocation` 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
    - `load-preferring-schemalocation`: `schemaLocation` 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。`schemaLocation` 属性が存在しない場合、`namespace` 属性の値がカタログマッピング 40 を介して使用されます。スキーマこれはデフォルト の値です。
    - `load-by-namespace`: カタログマッピング 40 を介して、`namespace` 属性の値がスキーマを検索するために使用されます。
    - `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。カタログマッピング 40 が存在しない場合、`schemaLocation` 属性が使用されます。
    - `license-namespace-only`: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

  - schemalocation-hints

    --schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
    `xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: `load-by-schemalocation`。

    - `load-by-schemalocation` 値はXML インスタンスドキュメント内の `xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性 の スキーマの場所のURL 317 使用します。これはデフォルト の値です。
    - `load-by-namespace` 値は、`xsi:schemaLocation` の名前空間の部分 317 を、そして `xsi:noNamespaceSchemaLocation` の場合は空の文字列を取ります。カタログマッピング 40 を介してスキーマを検索します。
    - `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
    - オプションの値が `ignore` の場合、`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性は両方とも無視されます。

- schema-mapping

  --schema-mapping = `prefer-schemalocation | prefer-namespace`
  スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(`--schemalocation-hints` または `--schema-imports` オプションが `load-combining-both` の値を有する場合、また、関連する名前空間とURL のパートが双方カタログマッピング 40 を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。`prefer-schemalocation` 値は URL マッピングを参照します)。) デフォルトは `prefer-schemalocation` です。

- xinclude

  --xinclude = `true|false`
  XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は `false` です。 `false` の場合、XInclude の `include` 要素は無視されます。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xml-mode

  --xml-mode = `wf|id|valid`
  XML インスタンスドキュメントのために使用される XML 処理モードを指定します: `wf`=整形式のチェック; `id`=ID/IDREF チェックと共に整形式のチェック; `valid`=検証。デフォルト値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-mode-for-schemas

  --xml-mode-for-schemas = `wf|id|valid`
  XML スキーマドキュメントのために使用される XML 処理モードを指定します。: `wf`=整形式のチェック; `id`=ID/IDREF チェックを持つ整形式; `valid`=検証。デフォルトの値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xsd-version

  --xsd-version = `1.0|1.1|detect`
  使用する W3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

- カタログとグローバルリソース

  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値は インストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース[49]を無効化します。デフォルト値は`false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成[49] を指定します(グローバルリソース[49]を有効化します)[49]。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル[49] を指定します。(グローバルリソース[49]を有効化します)[49]。

- 拡張子

  これらのオプションは、(XMLSpy Enterprise エディションなど) Enterprise レベルのAltova 製品の多くで使用することができ、特別な拡張関数を扱うオプションを定義します。使用方法はこれらの製品のユーザーマニュアルで説明されています。

  - chartext-disable

    --chartext-disable = `true|false`
    チャート拡張子を無効化します。デフォルト値は`false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - dotnetext-disable

    --dotnetext-disable = `true|false`
    .NET 拡張子を無効化します。デフォルト値は`false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - jvm-location

    --jvm-location = *FILE*
    FILE はJava 仮想マシン(W indows のDLL、Linux 上の共有されるオブジェクト) の場所を指定します。
    XSLT/XQuery コード内で Java 拡張関数[412] を使用する場合、JVMが必要になります。デフォルトは`false` です。

  - javaext-barcode-location

    --javaext-barcode-location = *FILE*
    バーコード拡張ファイル`AltovaBarcodeExtension.jar` を含むパスを指定します。.パスは次のフォームで与えられなければなりません

    - ファイルURI の例: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"`
    - バックスラッシュ付きのエスケープ文字を使用したW indows パスの例: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"`

  - javaext-disable

    --javaext-disable = `true|false`
    Java 拡張子を無効化します。デフォルト値は`false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

  --error-format = `text|shortxml|longxml`
  エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

- error-limit

  --error-limit = *N | unlimited*
  エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

- verbose

  --verbose = `true|false`
  `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  `1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.4 XSLT コマンド

XSLT コマンドは以下のとおりです:

- xslt [107]: XSLT ドキュメントと共にXML ドキュメントを変換するためのコマンド
- valxslt [114]: XSLT ドキュメントの検証のためのコマンド

4.4.1 xslt

xslt コマンドはXSLT ファイルを単一引数として取り、この引数を入力 XML ファイルを出力ファイルに変換するために使用します。入力および出力ファイルはオプション [192] として指定されます。

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

- *XSLT-File* 引数は変換に使用するXSLT ファイルのパスと名前です。
- 入力XML ファイル(--input [192]) または、名前のついたテンプレートエントリポイント (--template-entry-point [192]) が必要とされます。
- JSON データを変換するにはXPath 3.1 の **json-doc($path)** 関数を介して JSON データをロードし、**xslt** コマンドの **--initial-match-selection** オプションを使用します。下のサンプルの最後のアイテムを確認してください。
- --output [192] オプションが指定されていない場合、出力は標準の出力に書き込まれます。XSLT 1.0、2.0、または3.0 を使用することができます。デフォルトではXSLT 3.0 が使用されます。

サンプル

xslt コマンドのサンプル:

- **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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: \\"。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成

する場合、以下のような名前を与えます: `"C:\My Directory\\"`。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: `--option=value`。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。`--h, --help` オプションを使用してコマンドに関する情報を表示します。

- XSLT 処理
  - indent-characters

    --indent-characters = *VALUE*
    インデントとして使用される文字列を指定します。

  - function-param

    --function-param = *VALUE*
    最初の関数へパスされる関数を指定します。1 つ以上の関数を指定するには、オプションを複数回使用してください。順序が重要なことに注意してください。

  - global-context-item

    --global-context-item = *VALUE*
    グローバル編集の評価のために使用されたコンテキスト アイテムを指定します。

  - initial-function

    --initial-function = *VALUE*
    変換のエントリポイントとして実行される関数の名前。

  - initial-match-selection

    --initial-match-selection = *VALUE*
    最初の一致の選択の値 (シーケンス) を指定します。

  - initial-mode, template-mode

    --initial-mode, --template-mode = *VALUE*
    変換のために使用されるテンプレート モードを指定します。

  - initial-template, template-entry-point

    --initial-template, --template-entry-point = *VALUE*
    変換のエントリ ポイントである XSLT スタイルシート内の名前の付けられたテンプレートに名前を与えます。

  - input

    --input = *FILE*
    変換される XML ファイルの URL です。

  - output, xsltoutput

    output = *FILE*, xsltoutput = *FILE*
    プライマリ出力ファイルの URL。例えば、複数のファイルの HTML 出力の場合、プライマリ出力のファイルは、エントリポイント

HTML ファイルの場所になります。イメージファイルなどの、追加出力ファイルは、`xslt-additional-output-files` として報告されます。`--output` または `--xsltoutput` オプションが指定されていない場合、出力は標準の出力に書き込まれます。

- param [p]

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

  - ***XQuery***

    外部パラメーターの値を指定します。内部パラメータは、`declare variable` を持ち、変数名、`external` キーワード、およびセミコロンが続く、XQuery ドキュメント内で宣言されています。例:

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

    `external` キーワードである `$foo` は、外部パラメータとなり、その値は、ランタイム時に外部ソースからパスされます。外部パラメーターは、CLI コマンドにより値を与えられます。例:

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

    上で説明されているように、*KEY* は外部パラメータの名前です。*VALUE* は、XPath 式として与えられた外部パラメーターの値です。の値です。 CLI で使用されたパラメーター名は、XQuery ドキュメント内で宣言される必要があります。複数の外部パラメータが CLI ででパスされた値である場合、それぞれに個別の `--param` オプションが必要になります。 XPath 式にスペースが含まれる場合二重引用符を使用する必要があります。

  - ***XSLT***

    グローバルスタイルシートのパラメーターを指定します。*KEY* はパラメータ名であり、*VALUE* は、パラメーターの値を与える XPath 式です。CLI で使用されるパラメーター名は、スタイルシート内で宣言される必要があります。複数のパラメータが使用される場合、`--param` スイッチが各パラメータ前に使用される必要があります。XPath 式内または式内の文字列でスペースが含まれる場合は、二重引用符が、XPath 式の周りで使用される必要があります。*例:*

    ```
    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`

  ストリーミングシリアル化を有効化します。デフォルト値は `true` です。

  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- template-param

  --template-param = *KEY:VALUE*

  最初のテンプレートにのみパスされるパラメーターを指定します (子孫テンプレートの呼び出しにはパスされません)。複数のパラメーターを指定するには、各パラメーターのためにオプションを使用します。

- tunnel-param

  --tunnel-param = *KEY:VALUE*

  最初のテンプレートと子孫テンプレートの呼び出しへパスされるパラメーターを指定します。複数のパラメーターを指定するには、各パラメーターのためにオプションを使用します。

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

  `--xpath-static-type-errors-as-warnings =` `true|false`

  `true` の場合、警告を静的 XPath コンテキスト内で検知される型エラーにします。エラーは実行の失敗を引き起こしますが、警告は処理を継続することができます。デフォルトは、`false` です。

- xslt-version

--xslt-version = `1|1.0|2|2.0|3|3.0|3.1`
XSLT プロセッサがXSLT 1.0、XSLT 2.0、またはXSLT 3.0. を使用するか指定します。デフォルト値は `3` です。

XML スキーマとXML インスタンス

load-xml-with-psvi

--load-xml-with-psvi = `true|false`
入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: `true`。

schema-imports

--schema-imports = `load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only`
それぞれオプションの `namespace` 属性とオプションの `schemaLocation` 属性を持つ、`xs:import` 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL">`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: `load-preferring-schemalocation`.
振る舞いは以下の通りです:

- `load-by-schemalocation`: カタログマッピング 40 を考慮し、`schemaLocation` 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
- `load-preferring-schemalocation`: `schemaLocation` 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。`schemaLocation` 属性が存在しない場合、`namespace` 属性の値がカタログマッピング 40 を介して使用されます。スキーマこれはデフォルトの値です。
- `load-by-namespace`: カタログマッピング 40 を介して、`namespace` 属性の値がスキーマを検索するために使用されます。
- `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。カタログマッピング 40 が存在しない場合、`schemaLocation` 属性が使用されます。
- `license-namespace-only`: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: `load-by-schemalocation`。

- `load-by-schemalocation` 値はXML インスタンスドキュメント内の `xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性 の スキーマの場所のURL 317 使用します。これはデフォルトの値です。
- `load-by-namespace` 値は、`xsi:schemaLocation` の名前空間の部分 317 を、そして `xsi:noNamespaceSchemaLocation` の場合は空の文字列を取ります。カタログマッピング 40 を介してスキーマを検索します。
- `load-combining-both`: `namespace` または `schemaLocation` 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、`--schema-mapping` オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が `ignore` の場合、`xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性は両方とも無視されます。

- schema-mapping

  --schema-mapping = `prefer-schemalocation | prefer-namespace`
  スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(`--schemalocation-hints` または `--schema-imports` オプションが `load-combining-both` の値を有する場合、また、関連する名前空間とURL のパートが双方カタログマッピング 40 を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。`prefer-schemalocation` 値は URL マッピングを参照します)。) デフォルトは `prefer-schemalocation` です。

- xinclude

  --xinclude = `true|false`
  XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は `false` です。 `false` の場合、XInclude の `include` 要素は無視されます。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xml-mode

  --xml-mode = `wf|id|valid`
  XML インスタンスドキュメントのために使用される XML 処理モードを指定します: `wf`=整形式のチェック; `id`=ID/IDREF チェックと共に整形式のチェック; `valid`=検証。デフォルト値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-mode-for-schemas

  --xml-mode-for-schemas = `wf|id|valid`
  XML スキーマドキュメントのために使用される XML 処理モードを指定します。: `wf`=整形式のチェック; `id`=ID/IDREF チェックを持つ整形式; `valid`=検証。デフォルトの値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = `true|false`
  検証エラーを警告として扱うかを指定します。警告として扱われる場合、エラーが検出されても、XSLT 変換などの追加処理は継続されます。デフォルトは `false` です。

- xsd

  --xsd = *`FILE`*
  1 つまたは複数の XML スキーマドキュメント を XML インスタンスの検証に使用することを指定します。1 つ以上のスキーマドキュメントを指定するため、オプションを複数回追加します。

- xsd-version

  --xsd-version = `1.0|1.1|detect`
  使用する W3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

- カタログとグローバルリソース
  - catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス(`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`)です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *FILE*
  ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- 拡張子

  これらのオプションは、(XMLSpy Enterprise エディションなど) Enterprise レベルの Altova 製品の多くで使用することができ、特別な拡張関数を扱うオプションを定義します。使用方法はこれらの製品のユーザーマニュアルで説明されています。

  - chartext-disable

    --chartext-disable = `true|false`
    チャート拡張子を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - dotnetext-disable

    --dotnetext-disable = `true|false`
    .NET 拡張子を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - jvm-location

    --jvm-location = *FILE*
    FILE はJava 仮想マシン(Windows のDLL、Linux 上の共有されるオブジェクト) の場所を指定します。
    XSLT/XQuery コード内で Java 拡張関数 412 を使用する場合、JVMが必要になります。デフォルトは `false` です。

  - javaext-barcode-location

    --javaext-barcode-location = *FILE*
    バーコード拡張ファイル `AltovaBarcodeExtension.jar` を含むパスを指定します。.パスは次のフォームで与えられなければなりません

- ファイルURI の例: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"`
- バックスラッシュ付きのエスケープ文字を使用したWindows パスの例: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"`

javaext-disable

--javaext-disable = `true|false`
Java 拡張子を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

error-format

--error-format = `text|shortxml|longxml`
エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

error-limit

--error-limit = *`N | unlimited`*
エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

info-limit

--info-limit = *`N | unlimited`*
範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

help

--help
コマンドのヘルプテキストを表示します。例えば、`valany --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help valany`。)

log-output

--log-output = *`FILE`*
指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

network-timeout

--network-timeout = *`VALUE`*
リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

verbose

--verbose = `true|false`
`true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

verbose-output

--verbose-output = *FILE*
FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は100 です。

4.4.2 valxslt

valxslt コマンドはXSLT ファイルを単一の引数として取り、検証します。

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

- ***XSLT-File*** 引数はXSLT ファイルが検証されるためのパスと名前です。検証は、XSLT 1.0、2.0、または3.0 仕様に従い行われます。デフォルトで使用される使用はXSLT 3.0 です。

サンプル

valxslt コマンドのサンプルです:

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

- コマンドライン上の大文字と小文字の区別とスラッシュ

  Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
  Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

  * 小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
  * Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

- Windows システム上でのバックスラッシュ、スペース、および特別文字

  Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

- コマンドライン上の大文字と小文字の区別とスラッシュ

  Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
  Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

  * 小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
  * Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

- Windows システム上でのバックスラッシュ、スペース、および特別文字

  Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

- XSLT 処理
  - initial-mode, template-mode

    --initial-mode, --template-mode = *VALUE*
    変換のために使用されるテンプレートモードを指定します。

  - initial-template, template-entry-point

    --initial-template, --template-entry-point = *VALUE*
    変換のエントリポイントであるXSLT スタイルシート内の名前の付けられたテンプレートに名前を与えます。

  - xslt-version

    --xslt-version = 1|1.0|2|2.0|3|3.0|3.1
    XSLT プロセッサがXSLT 1.0、XSLT 2.0、またはXSLT 3.0. を使用するか指定します。デフォルト値は3 です。

- XML スキーマとXML インスタンス
  - load-xml-with-psvi

    --load-xml-with-psvi = true|false
    入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: true。

  - schema-imports

    --schema-imports = load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only
    それぞれオプションのnamespace 属性とオプションのschemaLocation 属性を持つ、xs:import 要素の振る舞いを指定します: <import namespace="someNS" schemaLocation="someURL">。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: load-preferring-schemalocation.
    振る舞いは以下の通りです:

    - load-by-schemalocation: カタログマッピング 40 を考慮し、schemaLocation 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。

- load-preferring-schemalocation: schemaLocation 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。schemaLocation 属性が存在しない場合、namespace 属性の値がカタログマッピング 40 を介して使用されます。スキーマこれはデフォルトの値です。
- load-by-namespace: カタログマッピング 40 を介して、namespace 属性の値がスキーマを検索するために使用されます。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。カタログマッピング 40 が存在しない場合、schemaLocation 属性が使用されます。
- license-namespace-only: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: load-by-schemalocation。

- load-by-schemalocation 値はXML インスタンスドキュメント内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所のURL 317 使用します。これはデフォルトの値です。
- load-by-namespace 値は、xsi:schemaLocation の名前空間の部分 317 を、そして xsi:noNamespaceSchemaLocation の場合は空の文字列を取ります。カタログマッピング 40 を介してスキーマを検索します。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング 40 がある場合、マッピング が使用されます。両方にカタログマッピング 40 , がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値が ignore の場合、xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(--schemalocation-hints または --schema-imports オプションが load-combining-both の値を有する場合、また、関連する名前空間とURL のパートが双方カタログマッピング 40 を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。prefer-schemalocation 値はURL マッピングを参照します)。) デフォルトは prefer-schemalocation です。

xinclude

--xinclude = `true|false`
XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は false です。 false の場合、XInclude の include 要素は無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

xml-mode

--xml-mode = `wf|id|valid`
XML インスタンスドキュメントのために使用される XML 処理モードを指定します: wf=整形式のチェック, id=ID/IDREF チェックと共に整形式のチェック, valid=検証。デフォルト値は wf です。valid の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
XML スキーマドキュメントのために使用される XML 処理モードを指定します。: `wf`=整形式のチェック; `id`=ID/IDREF チェックを持つ整形式; `valid`=検証。デフォルトの値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xsd-version

--xsd-version = `1.0|1.1|detect`
使用する W3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト `1.0` はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (`1.0` または `1.1`) は、ドキュメントの `<xs:schema>` 要素の `vc:minVersion` 属性の値を読み込むことにより検出されます。`@vc:minVersion` 属性の値が 1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

カタログとグローバルリソース

- catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値は インストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

--user-catalog = *FILE*
ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース 49 を無効化します。デフォルト値は `false` です。
メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

--gc | --globalresourceconfig = *VALUE*
グローバルリソースのアクティブな構成 49 を指定します (グローバルリソース 49 を有効化します) 49。

- globalresourcefile [gr]

--gr | --globalresourcefile = *FILE*
グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49。

拡張子

これらのオプションは、(XMLSpy Enterprise エディションなど) Enterprise レベルの Altova 製品の多くで使用することができ、特別な拡張関数を扱うオプションを定義します。使用方法はこれらの製品のユーザーマニュアルで説明されています。

- chartext-disable

--chartext-disable = `true|false`

チャート拡張子を無効化します。デフォルト値は false です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- dotnetext-disable

  --dotnetext-disable = true|false
  .NET 拡張子を無効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- jvm-location

  --jvm-location = *FILE*
  FILE はJava 仮想マシン(Windows のDLL、Linux 上の共有されるオブジェクト) の場所を指定します。
  XSLT/XQuery コード内で Java 拡張関数 412 を使用する場合、JVMが必要になります。デフォルトは false です。

- javaext-barcode-location

  --javaext-barcode-location = *FILE*
  バーコード拡張ファイル AltovaBarcodeExtension.jar を含むパスを指定します。.パスは次のフォームで与えられなければなりません:

  - ファイルURI の例: --javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"
  - バックスラッシュ付きのエスケープ文字を使用したWindows パスの例: --javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"

- javaext-disable

  --javaext-disable = true|false
  Java 拡張子を無効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

  --error-format = text|shortxml|longxml
  エラー出力のフォーマットを指定します。デフォルト値は text です。他のオプションは longxml と共に詳細付きのXML フォーマットを生成します。

- error-limit

  --error-limit = *N | unlimited*
  エラー制限を指定します。デフォルト値は 100 です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

- help

  --help

コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

- verbose

  --verbose = `true|false`
  `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  `1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.5 JSON/Avro コマンド

JSON コマンドは、JSON スキーマおよびインスタンスドキュメントの有効性と整形式をチェックするために使用することができます。これらのコマンドは、下にリストされており、このセクションのサブセクションに詳細が説明されています:

- avroextractschema [120]: Avro バイナリファイルから Avro スキーマを抽出します。
- valavro [129]: 1 つまたは複数の Avro バイナリ内のデータを、各バイナリの対応する Avro スキーマに対して検証します。
- valavrojson [132]: Avro スキーマに対して、1 つまたは複数の JSON データファイルを検証します。
- valavroschema [135]: Avro スキーマ仕様に対して、Avro スキーマを検証します。
- valjsonschema [138]: JSON スキーマドキュメントの有効性をチェックします。
- valjson [141]: JSON ドキュメントの有効性をチェックします。
- wfjson [145]: JSON ドキュメントの整形式をチェックします。

4.5.1 avroextractschema

Avro バイナリファイルは、データブロックの構造を定義する Avro スキーマにより優先される Avro データブロックを含みます。**avroextractschema** コマンドは、Avro バイナリから、Avro スキーマを抽出し、Avro スキーマを JSON としてシリアル化します。

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

- *AvroBinaryFile* 引数は、Avro スキーマが抽出される Avro バイナリファイルを指定します。
- `--output` オプションは、抽出された Avro スキーマの箇所を指定します。

サンプル

avroextractschema コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

* 小文字 (`raptorxml` と `raptorxmlserver`) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (`RaptorXML`) は Windows と Mac のみで使用することができます。
* Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のためにつまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: `--option=value`。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**`--h, --help`** オプションを使用してコマンドに関する情報を表示します。

- 処理
  - output, **`avrooutput`**

    --output = *FILE*, --avrooutput = *FILE*
    Avro 出力ファイルの場所を設定します。

  - recurse

    --recurse = `true|false`
    ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は`test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- カタログとグローバルリソース
  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = `true|false`
    グローバルリソース 49 を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - globalresourceconfig [gc]

    --gc | --globalresourceconfig = *VALUE*
    グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*

グローバルリソースファイル[49] を指定します。(グローバルリソース[49]を有効化します)[49]。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は`text` です。他のオプションは`longxml` と共に詳細付きのXMLフォーマットを生成します。

  - error-limit

    --error-limit = *`N | unlimited`*
    エラー制限を指定します。デフォルト値は`100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *`N | unlimited`*
    範囲`1-65535` または`unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は`100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - log-output

    --log-output = *`FILE`*
    指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

  - network-timeout

    --network-timeout = *`VALUE`*
    リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

  - verbose

    --verbose = `true|false`
    `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は`false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - verbose-output

    --verbose-output = *`FILE`*
    `FILE` に詳細出力を書き込みます。

  - version

    --version
    RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

▼ warning-limit

--warning-limit = *N | unlimited*
1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.5.2 avrotojson

avrotojson コマンドは、Avro バイナリファイル内のデータブロックを抽出し、データを JSON として、JSON データファイルにシリアル化します。

```
raptorxml avrotojson [options] --output=JSONFile AvroBinaryFile
```

- *AvroBinaryFile* 引数は、JSON 出力のためのデータを抽出する Avro バイナリです。
- --output オプションは、生成された JSON ファイルの箇所を指定します。

サンプル

avrotojason コマンドのサンプルです:

- **raptorxml** avrotojson --output=c:\MyAvroData.json c:\MyAvroBinary.avro

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

＊小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
＊Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字が文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 処理

- recurse

  --recurse = `true|false`
  ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *`InputFile`* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- カタログとグローバルリソース
  - catalog

    --catalog = *`FILE`*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *`FILE`*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = `true|false`
    グローバルリソース 49 を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - globalresourceconfig [gc]

    --gc | --globalresourceconfig = *`VALUE`*
    グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *`FILE`*
    グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きの XML フォーマットを生成します。

  - error-limit

    --error-limit = *`N | unlimited`*
    エラー制限を指定します。デフォルト値は `100` です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、val any --h。(または、help コマンドは引数と共に使用することができます。例: help val any。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、--version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.5.3 jsontoavro

jsontoavro コマンドは、1 つまたは複数の JSON データファイルを Avro バイナリファイルに変換します。JSON ドキュメントに従い有効な、 Avro スキーマドキュメントを指定する必要があります。Avro スキーマファイルと JSON データファイルは、Avro バイナリファイルにシリアル化されます。

```
raptorxml jsontoavro [options] --output=AvroBinary --schema=AvroSchema JSONFiles
```

- *JSONFiles* 引数は、--schema オプションにより指定されている Avro スキーマに従い有効である1 つまたは複数の JSON データファイルを指定します。
- --output オプションは、生成された Avro バイナリファイルの箇所を指定します。
- 複数の JSON ファイルをに変更するには、以下を行います: (i) CLI 上で変換するファイルをスペースにより区切りリストします。(ii) テキストファイル(.txt ファイル) 形式で変換するファイルを、1 行につき1 つのファイル名をリストします。そして、テキストファイルを *JSONFiles* 引数として提供し、--listfile 186 オプションを true に設定します(*下にリストされるオプションを参照してください*)。
- --codec オプション(*下を参照*) は、Avro 圧縮コーデックを指定し、null または deflate の値を取ります。デフォルトは、null です。

サンプル

jsontoavro コマンドのサンプルです:

- **raptorxml** jsontoavro --output=c:\MyAvroBinary.avro --schema=c:\MyAvroSchema.avsc c:\MyAvroData.json
- **raptorxml** jsontoavro --output=c:\MyAvroBinary.avro --schema=c:\MyAvroSchema.avsc c:\MyAvroData-01.json c:\MyAvroData-02.json
- **raptorxml** jsontoavro --output=c:\MyAvroBinary.avro --schema=c:\MyAvroSchema.avsc --listfile=true c:\MyAvroData.txt

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1 つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 処理

▼ listfile

--listfile = `true|false`
`true` の場合、コマンドの *InputFile* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。（代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。） `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- recurse

--recurse = `true|false`
ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- codec

--codec = `null|deflate`
使用する Avro 圧縮コーデックを指定します。デフォルトは、`null` です。

- カタログとグローバルリソース
  - catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値は インストールされたルートカタログファイルへの絶対パス（`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`）です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

--user-catalog = *FILE*
ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース 49 を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

--gc | --globalresourceconfig = *VALUE*
グローバルリソースのアクティブな構成 49 を指定します（グローバルリソース 49 を有効化します） 49 。

- globalresourcefile [gr]

--gr | --globalresourcefile = *FILE*
グローバルリソースファイル 49 を指定します。（グローバルリソース 49 を有効化します） 49 。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

error-format

--error-format = `text|shortxml|longxml`
エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

error-limit

--error-limit = *N | unlimited*
エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

info-limit

--info-limit = *N | unlimited*
範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

help

--help
コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

log-output

--log-output = *FILE*
指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

network-timeout

--network-timeout = *VALUE*
リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

verbose

--verbose = `true|false`
`true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

verbose-output

--verbose-output = *FILE*
FILE に詳細出力を書き込みます。

version

--version
RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

warning-limit

--warning-limit = *N | unlimited*

1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.5.4 valavro (avro)

valavro | avro コマンドは、各バイナリファイル内の対応する Avro スキーマに対して、1 つまたは複数の Avro バイナリファイル内のデータブロックを検証します。

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

- *AvroBinaryFile* 引数は、検証する 1 つまたは複数の Avro バイナリファイルを指定します。具体的には、各 Avro バイナリファイル内のデータブロックは、そのバイナリファイル内の Avro スキーマに対して検証されます。
- 複数の Avro バイナリを検証するには、以下を行います: (i) CLI 上で変換するファイルをスペースにより区切りリストします。(ii) テキストファイル (.txt ファイル) 形式で変換するファイルを、1 行につき 1 つのファイル名をリストします。そして、テキストファイルを *AvroBinaryFile* 引数として提供し、--listfile 186 オプションを true に設定します (*下にリストされるオプションを参照してください*)。

サンプル

valavro コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

* 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
* Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は

TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

- 処理
  - listfile

    --listfile = true|false
    true の場合、コマンドの *InputFile* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は false です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) --listfile オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - recurse

    --recurse = true|false
    ZIP アーカイブ内のファイルを選択するために使用されます。true の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: test.zip|zip\test.xml は test.xml という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。* および ? などのワイルドカード文字が使用されるかもしれません。ですから *.xml は、Zip フォルダー内のすべての .xml ファイルを選択します。オプションのデフォルト値は false です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- カタログとグローバルリソース
  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = true|false
    グローバルリソース 49 を無効化します。デフォルト値は false です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - globalresourceconfig [gc]

    --gc | --globalresourceconfig = *VALUE*
    グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

error-format

--error-format = `text|shortxml|longxml`
エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

error-limit

--error-limit = *N | unlimited*
エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

info-limit

--info-limit = *N | unlimited*
範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

help

--help
コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

log-output

--log-output = *FILE*
指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

network-timeout

--network-timeout = *VALUE*
リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

verbose

--verbose = `true|false`
`true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

verbose-output

--verbose-output = *FILE*
FILE に詳細出力を書き込みます。

version

--version
RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

warning-limit

--warning-limit = *N | unlimited*
`1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.5.5 valavrojson (avrojson)

valavrojson | avrojson コマンド は、JSON ドキュメントを Avro スキーマに対して検証します。

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

- *JSONFile* 引数は、検証する JSON ドキュメントを指定します。
- **--avroschema** オプションは、Avro スキーマがどの JSON ドキュメントに対して検証されるかを指定します。複数の
- JSON ファイルを検証するには、以下を行います: CLI 上で変換するファイルをスペースにより区切りリストします。(ii) テキストファイル (.txt ファイル) 形式で変換するファイルを、1 行につき 1 つのファイル名をリストします。そして、テキストファイルを *JSONFile* 引数として提供し、--listfile 186 オプションを true に設定します (*下にリストされるオプションを参照してください*)。

サンプル

valavrojson コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

* 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
* Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字が文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 処理

- listfile

  --listfile = `true|false`
  `true` の場合、コマンドの *`InputFile`* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- recurse

  --recurse = `true|false`
  ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *`InputFile`* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

カタログとグローバルリソース

- catalog

  --catalog = *`FILE`*
  インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *`FILE`*
  ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *`VALUE`*
  グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *`FILE`*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

--error-format = text|shortxml|longxml
エラー出力のフォーマットを指定します。デフォルト値は text です。他のオプションは longxml と共に詳細付きの XML フォーマットを生成します。

error-limit

--error-limit = *N | unlimited*
エラー制限を指定します。デフォルト値は 100 です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

info-limit

--info-limit = *N | unlimited*
範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

help

--help
コマンドのヘルプテキストを表示します。例えば、 val any --h。(または、 help コマンドは引数と共に使用することができます。例: help val any。)

log-output

--log-output = *FILE*
指定されたファイル URL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

network-timeout

--network-timeout = *VALUE*
リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

verbose

--verbose = true|false
true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

verbose-output

--verbose-output = *FILE*
FILE に詳細出力を書き込みます。

version

--version
RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、 --version をコマンドの前に置きます。

warning-limit

--warning-limit = *N | unlimited*
1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.5.6 valavroschema (avroschema)

valavroschema | avroschema コマンドは、1つまたは複数のAvro スキーマドキュメントをAvro スキーマ仕様に対して検証します。

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

- *AvroSchema* 引数は、検証されるAvro スキーマドキュメントです。
- 複数のAvro スキーマを検証するには、以下を行います: (i)CLI 上で変換するファイルをスペースにより区切りリストします。(ii) テキストファイル(.txt ファイル) 形式で変換するファイルを、1行につき1つのファイル名をリストします。そして、テキストファイルを *AvroSchema* 引数として提供し、--listfile 186 オプションを true に設定します (*下にリストされるオプションを参照してください*)。

サンプル

valavroschema コマンドのサンプルです:

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

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 処理

- listfile

  --listfile = `true|false`
  `true` の場合、コマンドの *`InputFile`* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- recurse

  --recurse = `true|false`
  ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *`InputFile`* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

カタログとグローバルリソース

- catalog

  --catalog = *`FILE`*
  インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

- user-catalog

  --user-catalog = *`FILE`*
  ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

- enable-globalresources

  --enable-globalresources = `true|false`
  グローバルリソース 49 を無効化します。デフォルト値は `false` です。
  メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *`VALUE`*
  グローバルリソースのアクティブな構成 49 を指定します (グローバルリソース 49 を有効化します) 49。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *`FILE`*
  グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

--error-format = text|shortxml|longxml
エラー出力のフォーマットを指定します。デフォルト値は text です。他のオプションは longxml と共に詳細付きのXMLフォーマットを生成します。

- error-limit

  --error-limit = *N | unlimited*
  エラー制限を指定します。デフォルト値は 100 です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、 val any --h。(または、 help コマンドは引数と共に使用することができます。例: help val any。)

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、 --version をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は 100 です。

4.5.7 valjsonschema (jsonschema)

valjsonschema | jsonschema コマンドは、1つまたは複数のJSON スキーマドキュメント を、JSON Schema Draft 4、Draft 6、またはDraft 7 仕様に従い検証します。

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

- *InputFile* 引数は、検証するJSON スキーマドキュメントです。
- 複数のドキュメントを検証するには以下を行います: (i) CLI 上に検証するファイルを各ファイルをスペースで区切り、リストします、または(ii) テキストファイル(.txt ファイル) 内の検証するファイルを、各ラインにファイル名を1つもつ、テキストファイルを --listfile [186] オプションを true に設定して、*InputFile* 引数として与えます (*下にリストされるオプションを参照*)。

サンプル

valjsonschema コマンドのサンプルです:

- **raptorxml** valjsonschema c:\MyJSONSchema.json
- **raptorxml** jsonschema c:\MyJSONSchema-01.json c:\MyJSONSchema-02.json
- **raptorxml** jsonschema --listfile=true c:\FileList.txt

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 検証と処理

▼ listfile

--listfile = true|false

true の場合、コマンドの *InputFile* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は false です。（代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。） --listfile オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- recurse

  --recurse = true|false
  ZIP アーカイブ内のファイルを選択するために使用されます。true の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: test.zip|zip\test.xml はtest.xml という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。* および ? などのワイルドカード文字が使用されるかもしれません。ですから *.xml は、Zip フォルダー内のすべての .xml ファイルを選択します。オプションのデフォルト値は false です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- JSON 検証オプション
  - additional-schema

    --additional-schema = *FILE*
    追加スキーマドキュメントのURI を指定します。追加スキーマはメインスキーマによりロードされ、追加スキーマ **id** または **$id** プロパティによりメインスキーマから参照することができます。

  - disable-format-checks

    --disable-format-checks = true|false
    書式属性により指定されたセマンティック検証を無効化します。デフォルトの値は false です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - jsonschema-version

    --jsonschema-version = draft04|draft06|draft07|latest|detect
    使用するJSON スキーマ仕様ドラフトバージョンのバージョンを指定します。デフォルトは detect です。

- カタログとグローバルリソース
  - catalog

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス（<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml）です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *FILE*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

    --enable-globalresources = true|false

グローバルリソース[49]を無効化します。デフォルト値は false です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成[49] を指定します(グローバルリソース[49]を有効化します)[49]。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル[49] を指定します。(グローバルリソース[49]を有効化します)[49]。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = text|shortxml|longxml
    エラー出力のフォーマットを指定します。デフォルト値は text です。他のオプションは longxml と共に詳細付きのXMLフォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は 100 です。 1 から 999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲 1-65535 または unlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は 100 です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、 val any --h。(または、 help コマンドは引数と共に使用することができます。例: help val any。)

  - log-output

    --log-output = *FILE*
    指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

  - network-timeout

    --network-timeout = *VALUE*
    リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

  - verbose

    --verbose = true|false
    true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  `1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.5.8 valjson (json)

valjson | json コマンドは、1 つまたは複数の JSON インスタンスドキュメントを、`--schema` オプションにより与えられた、JSON スキーマに従い検証します。

```
raptorxml valjson | json [options] --jsonschema=File InputFile
```

- *InputFile* 引数は、検証する JSON インスタンスドキュメントです。
- 複数のドキュメントを検証するには以下を行います: (i) CLI 上に検証するファイルを各ファイルをスペースで区切り、リストします、または (ii) テキストファイル (`.txt` ファイル) 内の検証するファイルを、各ラインにファイル名を 1 つもつ、テキストファイルを `--listfile` 186 オプションを `true` に設定して、*InputFile* 引数として与えます (*下にリストされるオプションを参照*)。

サンプル

valjson コマンドのサンプルです:

- **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`

- コマンドライン上の大文字と小文字の区別とスラッシュ

  Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
  Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

  * 小文字 (`raptorxml` と `raptorxmlserver`) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (`RaptorXML`) は Windows と Mac のみで使用することができます。
  * Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

- Windows システム上でのバックスラッシュ、スペース、および特別文字

  Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文

字が文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ（例えば、"**C:\My directory**"）は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

（使用できる場合）オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

- 検証と処理
  - schema, **jsonschema**

    --schema = *FILE*, --jsonschema = *FILE*
    JSON インスタンスドキュメントの検証のために使用する、JSON スキーマドキュメントを指定します。

  - listfile

    --listfile = true|false
    true の場合、コマンドの *InputFile* 引数を各ラインに1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は false です。（代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。）--listfile オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - additional-schema

    --additional-schema = *FILE*
    追加スキーマドキュメントのURI を指定します。追加スキーマはメインスキーマによりロードされ、追加スキーマ **id** または **$id** プロパティによりメインスキーマから参照することができます。

  - recurse

    --recurse = true|false
    ZIP アーカイブ内のファイルを選択するために使用されます。true の場合、コマンドの *InputFile* 引数は指定されたファイルをサブディレクトリでも選択します。例: test.zip|zip\test.xml はtest.xml という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。* および ? などのワイルドカード文字が使用されるかもしれません。ですから *.xml は、Zip フォルダー内のすべての .xml ファイルを選択します。オプションのデフォルト値は false です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - json5

    --json5 = true|false
    JSON5 サポートを有効化します。デフォルトの値は false です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - jsonc

    --jsonc = true|false

JSON 内のコメントのためのサポートを有効化します。デフォルトの値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- **json-lines**

  --json-lines = `true|false`
  JSON ライン (すなわち1つのJSON 値が各ラインに表示される) のためのサポートが有効化されます。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- **jsonschema-version**

  --jsonschema-version = `draft04|draft06|draft07|latest|detect`
  使用するJSON スキーマ仕様ドラフトバージョンのバージョンを指定します。デフォルトは `detect` です。

- **disable-format-checks**

  --disable-format-checks = `true|false`
  書式属性により指定されたセマンティック検証を無効化します。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- **カタログとグローバルリソース**
  - **catalog**

    --catalog = *FILE*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - **user-catalog**

    --user-catalog = *FILE*
    ルートカタログに追加して使用されるXML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - **enable-globalresources**

    --enable-globalresources = `true|false`
    グローバルリソース 49 を無効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

  - **globalresourceconfig [gc]**

    --gc | --globalresourceconfig = *VALUE*
    グローバルリソースのアクティブな構成 49 を指定します (グローバルリソース 49 を有効化します) 49 。

  - **globalresourcefile [gr]**

    --gr | --globalresourcefile = *FILE*
    グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

メッセージ、エラー、ヘルプ、タイムアウト、バージョン

error-format

--error-format = `text|shortxml|longxml`
エラー出力のフォーマットを指定します。デフォルト値は `text` です。他のオプションは `longxml` と共に詳細付きのXMLフォーマットを生成します。

error-limit

--error-limit = *N | unlimited*
エラー制限を指定します。デフォルト値は `100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

info-limit

--info-limit = *N | unlimited*
範囲 `1-65535` または `unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は `100` です。

help

--help
コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

log-output

--log-output = *FILE*
指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

network-timeout

--network-timeout = *VALUE*
リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

verbose

--verbose = `true|false`
`true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

verbose-output

--verbose-output = *FILE*
FILE に詳細出力を書き込みます。

version

--version
RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

warning-limit

--warning-limit = *N | unlimited*
`1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

(C) 2015-2021 Altova GmbH

4.5.9 wfjson

wfjson コマンド は、1 つまたは複数のJSON スキーマドキュメントを、整形式のためのECMA-404 仕様 に従い検証します。

```
raptorxml wfjson [options] InputFile
```

- *InputFile* 引数は、整形式をチェックするための(スキーマ または インスタンス) JSON ドキュメントです。
- 複数のドキュメントを検証するには以下を行います: (i) CLI 上に検証するファイルを各ファイルをスペースで区切り、リストします、または(ii) テキストファイル(.txt ファイル) 内の検証するファイルを、各ラインにファイル名を1 つもつ、テキストファイルを --listfile 186 オプションを true に設定して、*InputFile* 引数として与えます (*下にリストされるオプションを参照*)。

サンプル

wfjson コマンドのサンプルです:

- **raptorxml** wfjson c:\MyJSONFile.json
- **raptorxml** wfjson c:\MyJSONFile-01.json c:\MyJSONFile-02.json
- **raptorxml** wfjson --listfile=true c:\FileList.txt

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1 つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 検証と処理

- json5

  --json5 = `true|false`
  JSON5 サポートを有効化します。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- jsonc

  --jsonc = `true|false`
  JSON 内のコメントのためのサポートを有効化します。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- json-lines

  --json-lines = `true|false`
  JSON ライン (すなわち 1 つの JSON 値が各ラインに表示される) のためのサポートが有効化されます。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- listfile

  --listfile = `true|false`
  `true` の場合、コマンドの *`InputFile`* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- recurse

  --recurse = `true|false`
  ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *`InputFile`* 引数は指定されたファイルをサブディレクトリでも選択します。例: `test.zip|zip\test.xml` は `test.xml` という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。`*` および `?` などのワイルドカード文字が使用されるかもしれません。ですから `*.xml` は、Zip フォルダー内のすべての `.xml` ファイルを選択します。オプションのデフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- カタログとグローバルリソース
  - catalog

    --catalog = *`FILE`*
    インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

  - user-catalog

    --user-catalog = *`FILE`*
    ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

  - enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース[49]を無効化します。デフォルト値は`false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- globalresourceconfig [gc]

  --gc | --globalresourceconfig = *VALUE*
  グローバルリソースのアクティブな構成[49] を指定します(グローバルリソース[49]を有効化します)[49]。

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  グローバルリソースファイル[49] を指定します。(グローバルリソース[49]を有効化します)[49]。

- メッセージ、エラー、ヘルプ、タイムアウト、バージョン
  - error-format

    --error-format = `text|shortxml|longxml`
    エラー出力のフォーマットを指定します。デフォルト値は`text` です。他のオプションは`longxml` と共に詳細付きのXMLフォーマットを生成します。

  - error-limit

    --error-limit = *N | unlimited*
    エラー制限を指定します。デフォルト値は`100` です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

  - info-limit

    --info-limit = *N | unlimited*
    範囲`1-65535` または`unlimited` 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は`100` です。

  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - log-output

    --log-output = *FILE*
    指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

  - network-timeout

    --network-timeout = *VALUE*
    リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: `40`。

  - verbose

    --verbose = `true|false`
    `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は`false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

- version

  --version
  RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

- warning-limit

  --warning-limit = *N | unlimited*
  `1-65535` 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

(C) 2015-2021 Altova GmbH

4.6 XML 署名コマンド

XML 署名コマンドはドキュメントを署名するために使用することができ、署名済みのXMLドキュメントを検証することができます。これらのコマンドは下にリストされており、このセクションのサブセクションで詳しく説明されています:

- xmlsignature-sign 149: 入力ドキュメントからXML 署名出力ドキュメントを生成します。
- xmlsignature-verify 153: XML 署名ドキュメントを検証します。
- xmlsignature-update 149: (変更された) XMLドキュメントの署名を更新します。
- xmlsignature-remove 149: XMLドキュメントの署名を削除します。

4.6.1 xmlsignature-sign

xmlsignature-sign | xsign コマンドはXMLドキュメントを入力として取りXML 署名出力ドキュメントを指定された署名オプションを使用して作成します。

```
raptorxml xmlsignature-sign [options] --output=File --signature-type=Value --
signature-canonicalization-method=Value --certname=Value|hmackey=Value InputFile
```

- *InputFile* 引数は署名するXMLドキュメントです。
- **--output** オプションはXML 署名を含むドキュメントの場所を指定します。

サンプル

xmlsignature-sign コマンドのサンプルです:

- **raptorxml** xsign --output=c:\SignedFile.xml --signature-type=enveloped --signature-canonicalization-method=xml-c14n11 --hmackey=secretpassword c:\SomeUnsigned.xml

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、またはフォルダー名、企業名、または個人、または製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用すること

ができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

- 共通のオプション
  - output

    output = *FILE*
    新規のXML 署名と共に作成される出力ドキュメントのURL です。

  - verbose

    --verbose = true|false
    true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
    メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- XML 署名オプション
  - absolute-reference-uri

    --absolute-reference-uri = true|false
    署名済みドキュメントのURI が絶対 (true) または相対 (false) で読み取られるかを指定します。デフォルトは false です。
    メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

  - certname, **certificate-name**

    --certname, --certificate-name = *VALUE*
    署名のために使用される証明書の名前です。

    Windows
    選択された **--certificate-store** からの証明書の**サブジェクト**名です。

    (PowerShell の下)証明書のリストのサンプル

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

    サンプル: **--certificate-name==certificate1**

    Linux/MacOS
    --certname はエンコードされる PEM の名前を指定します。X.509v3 c証明書には秘密キーが存在します。このようなファイルには通常拡張子 **.pem** がつきます。

    サンプル: **--certificate-name==/path/to/certificate1.pem**

  - certstore, **certificate-store**

    --certstore, --certificate-store = *VALUE*

--certificate-name を使用して証明書が保管される場所です。

Windows
cert://CurrentUser に保管される証明書ストアの名前です。(PowerShell の下) **% ls cert://CurrentUser/** を使用して使用することのできる証明書ストアをリストすることができます:

```
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
```

サンプル: **--certificate-store==MyCertStore**

Linux/MacOS
--certstore オプションは現在サポートされていません。

digest, digest-method

--digest, --digest-method = sha1|sha256|sha384|sha512
入力 XML ファイルのダイジェスト値を計算するために使用されるアルゴリズム。使用可能な値: sha1|sha256|sha384|sha512.

hmackey, hmac-secret-key

--hmackey, --hmac-secret-key = *VALUE*
HMAC 共有される秘密キー。最小6文字の長さが必要です。

サンプル: **--hmackey=secretpassword**

hmaclen, hmac-output-length

--hmaclen, --hmac-output-length = *LENGTH*
LENGTH ビットに HMAC アルゴリズムの出力を 切り捨てます。

- 8 の倍数
- 80 より大きい
- 元になるハッシュアルゴリズムの出力の長さの半分よりも大きい必要があります

keyinfo, append-keyinfo

--keyinfo, --append-keyinfo = `true|false`
署名に **KeyInfo** 要素を含むかを決定します。デフォルトは **false** です。

sigc14nmeth, signature-canonicalization-method

--sigc14nmeth, --signature-canonicalization-method = *VALUE*
SignedInfo 要素に適用する標準化アルゴリズムを指定します。値は以下の内の1つになります:

- `REC-xml-c14n-20010315`
- `xml-c14n11`
- `xml-exc-c14n#`

sigmeth, signature-method

--sigmeth, --signature-method = *VALUE*
署名を生成するために使用されるアルゴリズムを指定します。

証明書が使用される場合
証明書が指定される場合、`--signature-method` は任意で、このパラメータのための値は証明書から計算されます。指定されると、証明書により使用されるアルゴリズムに一致する必要があります。

サンプル: **--signature-method=rsa-sha256**

--hmac-secret-key が使用される場合
`--hmac-secret-key` が使用される場合、このオプションは必須です。HMAC アルゴリズムにサポートされる値である必要があります:

- `hmac-sha256`
- `hmac-sha386`
- `hmac-sha512`
- hmac-sha1 (仕様では使用が奨励されていません)

サンプル: **--signature-method=hmac-sha256**

sigtype, signature-type

--sigtype, --signature-type = `detached | enveloping | enveloped`
生成される署名の型を指定します。

transforms

(C) 2015-2021 Altova GmbH

--transforms = *VALUE*
入力ドキュメントに適用される XML 署名変換を指定します。このオプションは複数回指定することができ、指定の順序は重要です。最初に指定される変換は入力ドキュメントを受信します。最後に指定される変換はダイジェスト値の計算の直前に使用されます。

サポートされる値:

- Canonical XML 1.0 (コメント無し) のための `REC-xml-c14n-20010315`
- Canonical XML 1.1 (コメント無し) のための `xml-c14n11`
- Exclusive XML Canonicalization 1.0 (コメント無し) のための `xml-exc-c14n#`
- Canonical XML 1.0 (コメント有り) のための `REC-xml-c14n-20010315#WithComments`
- Canonical XML 1.1 (コメント有り) のための `xml-c14n11#WithComments`
- Exclusive XML Canonicalization 1.0 (コメント有り) のための `xml-exc-c14n#WithComments`
- `base64`
- `strip-whitespaces` Altova 拡張子

サンプル: **`--transforms=xml-c14n11`**

- write-default-attributes

  --write-default-attributes = `true|false`
  署名済みドキュメント内のDTD からのデフォルトの値を含むかを指定します。

- *ヘルプとバージョンオプション*
  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - version

    --version
    RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

4.6.2 xmlsignature-verify

`xmlsignature-verify | xverify` コマンドは署名済み入力ファイルを検証します。

```
raptorxml xmlsignature-verify [options] InputFile
```

- *`InputFile`* 引数は検証される署名済み XML ドキュメントです。
- 検証に成功した場合、**`result="OK"`** メッセージが表示されます。それ以外の場合、**`result="Failed"`** メッセージが表示されます。

サンプル

xmlsignature-verify コマンドのサンプルです:

- **raptorxml** xverify c:\SignedFile.xml

コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、またはフォルダー名、企業名、または個人、または製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

共通のオプション

verbose

--verbose = true|false
true の値は、検証中の追加情報の出力を有効化します。デフォルト値は false です。
メモ: ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

XML 署名オプション

certname, **certificate-name**

--certname, --certificate-name = *VALUE*
署名のために使用される証明書の名前です。

Windows
選択された **--certificate-store** からの証明書の**サブジェクト**名です。

(PowerShell の下)証明書のリストのサンプル

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
```

```
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

サンプル: **--certificate-name==certificate1**

Linux/MacOS
--certname はエンコードされる PEM の名前を指定します。X.509v3 c証明書には秘密キーが存在します。このようなファイルには通常拡張子 **.pem** がつきます。

サンプル: **--certificate-name==/path/to/certificate1.pem**

certstore, **certificate-store**

--certstore, --certificate-store = *VALUE*
--certificate-name を使用して証明書が保管される場所です。

Windows
cert://CurrentUser に保管される証明書ストアの名前です。(PowerShell の下) **% ls cert://CurrentUser/** を使用して使用することのできる証明書ストアをリストすることができます:

```
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
```

サンプル: **--certificate-store==MyCertStore**

Linux/MacOS
--certstore オプションは現在サポートされていません。

hmackey, hmac-secret-key

--hmackey, --hmac-secret-key = *VALUE*
HMAC 共有される秘密キー。最小6文字の長さが必要です。

サンプル: **--hmackey=secretpassword**

- ヘルプとバージョンオプション
  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - version

    --version
    RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

4.6.3 xmlsignature-update

xmlsignature-update | xupdate コマンドは署名済みの入力ファイル内のXML 署名を更新します。ドキュメントが変更されると、 XML 署名も変更されます。それ以外の場合は、更新された署名は前の署名と同じです。

```
raptorxml xmlsignature-update [options] --output=File SignedFile
```

- *SignedFile* 引数は更新する署名済みのXMLドキュメントです。
- `hmac-secret-key` オプション、または(ii) `certificate-name` と`certificate-store` オプションが指定される必要があります。
- `certificate-name` と`certificate-store` オプションが指定される場合、XMLドキュメントを署名するために以前に使用されたものに一致する必要があります。(`certificate-store` オプションは現在 Linux とmacOS 上ではサポートされていないことに注意してください)。

サンプル

xmlsignature-update コマンドのサンプルです:

- **raptorxml** `xupdate --output=c:\UpdatedSignedFile.xml --certname=certificate1 --certstore=MyCertStore c:\SomeSignedFile.xml`
- **raptorxml** `xupdate --output=c:\UpdatedSignedFile.xml --hmackey=SecretPassword c:\SomeSignedFile.xml`

- コマンドライン上の大文字と小文字の区別とスラッシュ

  Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
  Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

  * 小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はWindows とMac のみで使用することができます。
  * Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

- Windows システム上でのバックスラッシュ、スペース、および特別文字

  Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で)スペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ(例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの

開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: `\\"`。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: `"C:\My Directory\\"`。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: `--option=value`。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。`--h, --help` オプションを使用してコマンドに関する情報を表示します。

- 共通のオプション
  - output

    output = *FILE*
    新規のXML 署名と共に作成される出力ドキュメントのURL です。

  - verbose

    --verbose = `true|false`
    `true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
    メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- XML 署名オプション
  - certname, `certificate-name`

    --certname, --certificate-name = *VALUE*
    署名のために使用される証明書の名前です。

    Windows
    選択された `--certificate-store` からの証明書の**サブジェクト**名です。

    (PowerShell の下)証明書のリストのサンプル
    ```
    % ls cert://CurrentUser/My
    PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
    Thumbprint Subject
    ---------- -------
    C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
    ... CN=...
    ```

    サンプル: `--certificate-name==certificate1`

    Linux/MacOS
    `--certname` はエンコードされる PEM の名前を指定します。X.509v3 c証明書には秘密キーが存在します。このようなファイルには通常拡張子 `.pem` がつきます。

    サンプル: `--certificate-name==/path/to/certificate1.pem`

  - certstore, `certificate-store`

--certstore, --certificate-store = *VALUE*
--certificate-name を使用して証明書が保管される場所です。

Windows
cert://CurrentUser に保管される証明書ストアの名前です。(PowerShell の下) **% ls cert://CurrentUser/** を使用して使用することのできる証明書ストアをリストすることができます:

```
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
```

サンプル: **--certificate-store==MyCertStore**

Linux/MacOS
--certstore オプションは現在サポートされていません。

hmackey, hmac-secret-key

--hmackey, --hmac-secret-key = *VALUE*
HMAC 共有される秘密キー。最小6文字の長さが必要です。

サンプル: **--hmackey=secretpassword**

ヘルプとバージョンオプション

help

--help
コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `hel p val any`。)

version

--version
RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

4.6.4 xmlsignature-remove

xmlsignature-remove | xremove コマンドは、署名済みの入力ファイルのXML 署名を削除し、指定する出力の場所に署名が削除された結果を保存します。

```
raptorxml xmlsignature-remove [options] --output=File SignedFile
```

- XML 署名を削除する署名済みのXMLドキュメントの *SignedFile* 引数です。
- `--output` オプションは署名の削除された生成されたXMLドキュメントの場所を指定します。

サンプル

xmlsignature-remove コマンドのサンプルです:

- **raptorxml** `xremove --output=c:\UnsignedFile.xml c:\SignedFile.xml`

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ 共通のオプション

▼ output

output = *FILE*
XML 署名が削除されている形で作成された出力ドキュメントのURL。

▼ verbose

--verbose = `true|false`
`true` の値は、検証中の追加情報の出力を有効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- ヘルプとバージョンオプション
  - help

    --help
    コマンドのヘルプテキストを表示します。例えば、`val any --h`。(または、`help` コマンドは引数と共に使用することができます。例: `help val any`。)

  - version

    --version
    RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

4.7 一般的なコマンド

このセクションには次の一般的なコマンドの説明が含まれています:

- valany [161]: 型により送信されたドキュメントを検証します
- script [162]: Python スクリプトを実行します
- help [163]: 名前を持つコマンドに関する情報を表示します

4.7.1 valany

valany コマンドは、ドキュメントの型をベースにしてドキュメントを検証する一般的なコマンドです。入力ドキュメントの型は、自動的に検出され、対応するけんしょうが対応する仕様に従い実行されます。*InputFile* 引数は、検証されるドキュメントです。コマンドの引数として1つのドキュメントのみを提出することができることに注意してください。

```
raptorxml valany [options] InputFile
```

valany コマンドは、以下の検証の型をカバーします。 対応する個々の検証コマンドのためのオプションです。対応するオプションのリストは、対応する検証コマンドを確認してください。

- valdtd (dtd) [62]
- valxsd (xsd) [66]
- valxml-withdtd (xml) [53]
- valxml-withxsd (xsi) [56]
- valxslt [114]
- valxquery [95]
- valavrojson (avrojson) [132]

サンプル

- **raptorxml** valany c:\Test.xsd

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム (W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字が文字列内で発生するとき引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

対応するオプションのリストを確認するには、それぞれの検証コマンドの詳細を参照してください。個々の検証コマンドが複数の入力ドキュメントを受け入れますが、**valany** コマンドは、1つの入力ドキュメントのみを受け入れることに注意してください。 `--listfile` オプション等のオプションは、**valany** に適用されません。

4.7.2 script

script コマンドは、RaptorXML Python API を使用するPython 3.7 スクリプトを実行します。

```
raptorxml script [options] PythonScriptFile
```

File 引数は、実行するPython スクリプトへのパスです。追加のオプションも使用することができます。追加オプションのリストを取得するには、以下のコマンドを実行してください：

```
raptorxml script [-h | --help]
```

サンプル

- **raptorxml** script c:\MyPythonScript.py
- **raptorxml** script -h
- **raptorxml** script # *スクリプトファイル無し。対話型 Python シェルが開始されます。*
- **raptorxml** script -m pip # *ロードし、pip モジュールを実行します。下のオプションのセクションを参照。*

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML**（および管理コマンドのための**RaptorXMLServer**）
Windows とUnix（Linux, Mac）上の **raptorxml**（および管理コマンドのための**raptorxmlserver**）

＊小文字（raptorxml とraptorxmlserver）はすべてのプラットフォーム（Windows、Linux、およびMac）で使用することができますが、小文字（RaptorXML）はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム（例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ（例えば、"**C:\My directory**"）は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

オプション

script コマンドの後のコマンドと引数は、直接 Python インタープリターに転送されます。使用することのできるオプションの完全なリストは、Python ドキュメントのページ https://docs.python.org/3.7/using/cmdline.html を参照してください。

4.7.3 help

構文と説明

help コマンドは、ヘルプを必要とするコマンドの名前である単一の引数 (Command) を取ります。コマンドの構文、オプション、および他の関連する情報を表示します。Command 引数が指定されていない場合、説明と共に実行可能ファイルのすべてのコマンドがリストされます。**help** コマンドは実行可能ファイルから呼び出すこともできます: **raptorxml** または **raptorxmlserver**。

```
raptorxml help Command
raptorxmlserver help Command
```

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム (W indows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

例

help コマンドのサンプルは**licenserver** コマンドに関する情報を表示します (このコマンドは両方の実行可能ファイルで使用することができます):

```
raptorxml help licenseserver
raptorxmlserver help licenseserver
```

--help オプション

ヘルプ情報が必要とされるコマンドの --help オプションを使用してヘルプ情報を使用することもできます。下の2 つのコマンドは同じ結果を生成します:

```
raptorxml licenseserver --help
```

上記のコマンドは**licenseserver** コマンドの --help オプションを使用します。

```
raptorxml help licenseserver
```

help コマンドは引数として licenseserver を取ります。

両方のコマンドは**licenseserver** コマンドに関するヘルプ情報を表示します。

4.8 ローカライズコマンド

RaptorXML アプリケーションのローカライズされたバージョンを希望する言語で作成することができます。既にローカライズされた5 つの言語 (英語、ドイツ語、スペイン語、フランス語、および日本語) は、<ProgramFilesFolder>\Altova\RaptorXMLServer2021\bin\フォルダー内にあり使用することができます。ですから、これらの4 つの言語のバージョンを作成する必要はありません。

ローカライズされたバージョンを他の言語で作成するには以下を行います:

1. リソース文字列を含むXML ファイルを生成します。 exportresourcestrings 164 コマンドを使用して行います。生成されたファイル内のソース文字列は、コマンドで使用される引数に従い、サポートされる言語のいずれになります: 英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、または日本語 (ja)。
2. 生成されたXML ファイルの言語からターゲット言語にソース文字列を翻訳します。ソース文字列はXML ファイル内の <string> 要素のコンテンツです。{option} または{product} などの中かっこ内の変数は翻訳しないでください。
3. Altova サポート に連絡を取り、ローカライズされたRaptorXML DLL ファイルを翻訳されたXML ファイルから生成してください。
4. Altova サポートからローカライズされたDLL ファイルを受け取ると、DLL を <ProgramFilesFolder>¥Altova¥RaptorXMLServer2021¥bin¥ フォルダー内に保存します。DLL ファイルは、RaptorXMLServer_lc.dll の書式の名前を与えられます。名前の_lc 部分は言語コードを含みます。例えば、RaptorXMLServer_de.dll の場合、de の部分がドイツ語 (Deutsch) の言語コードです。
5. setdeflang 166 コマンドを実行し、使用するRaptorXML アプリケーションとしてローカライズされたDLL ファイルを設定します。setdeflang 166 コマンドの引数は、 DLL名の一部である言語コードを使用します。

メモ Altova RaptorXML Server は以下の言語へのサポートが搭載されています: 英語、ドイツ語、スペイン語、フランス語、および日本語。ですから、これらの言語のローカライズされたバージョンを作成する必要はありません。これらの5 つの言語のいずれかをデフォルトの言語に設定する場合は、CLI の setdeflang 166 コマンドを使用してください。

4.8.1 exportresourcestrings

構文と説明

exportresourcestrings コマンドは特定の言語内のRaptorXML Server アプリケーションのソース文字列を含むXML ファイルを出力します。使用することのできる言語は英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、と日本語 (ja) です。

```
raptorxml exportresourcestrings [options] LanguageCode XMLOutputFile
raptorxmlserver exportresourcestrings [options] LanguageCode XMLOutputFile
```

- ***LanguageCode*** 引数は、出力 XML ファイル内のソース言語の言語を与えます。これは *エクスポート言語* です。許可されているエクスポート言語は以下の通りです (それぞれの言語コードがかっこで囲まれています) : 英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、と日本語 (ja)。
- ***XMLOutputFile*** 引数は出力 XML ファイルのパスと名前を指定します
- **exportresourcestrings**コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

ローカリゼーションの作成方法は以下のとおりです。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
* Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

例

exportresourcestrings コマンドのサンプル:

```
raptorxml exportresourcestrings de c:\Strings.xml
raptorxmlserver exportresourcestrings de c:\Strings.xml
```

- 上の最初のコマンドはRaptorXML Server のドイツ語リソース文字列を含む Strings.xml と呼ばれるファイルを c:\ に作成します。
- 2番目のコマンドはサーバー実行可能ファイルを呼び出し最初のコマンドと同様に作動します。

RaptorXML Server のローカライズされたバージョンの作成

選択された言語で RaptorXML Server のローカライズされたバージョンを作成することができます。5つのローカライズされたバージョン (英語、ドイツ語、スペイン語、フランス語、および日本語) は C:\Program Files (x86)\Altova\RaptorXMLServer2021\bin フォルダー内で使用することができるため、作成する必要はありません。

ローカライズされたバージョンを以下のように作成します:

1. exportresourcestrings コマンドを使用してリソース文字列を含む XML ファイルを作成します (*上記のコマンド構文を参照してください*)。この XML ファイル内のリソース文字列は、サポートされる以下の5つの言語のひとつです: コマンドと共に使用される LanguageCode 引数に従い英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、または日本語 (ja)。
2. 5つのサポートされる言語の1つからのリソース文字列からターゲット言語に翻訳します。リソース文字列は XML ファイル内の <string> 要素のコンテンツです。{option} または {product} などの、中かっこ内の変数は翻訳しないでください。
3. ローカライズされた RaptorXML Server DLL ファイルを翻訳された XML ファイルから生成するために、Altova サポート に連絡してください。
4. Altova サポート からローカライズされた DLL ファイルを受信すると、DLL を C:\Program Files (x86)\Altova\<%APPFOLDER\bin フォルダー内に保存します。DLL ファイルはフォーム RaptorXML2021_lc.dll の名前を持ちます。名前の _lc 部分には、言語コードが含まれています。例えば、 RaptorXML2021_de.dll では、de の部分は、ドイツ語 (Deutsch) のための言語コードです。
5. setdeflang コマンドを実行し、使用する RaptorXML Server アプリケーションと同様にローカライズされた DLL ファイルを設定します。setdeflang コマンドの引数のために、DLL 名の一部である言語コードを使用してください。

メモ Altova RaptorXML Server は5つの言語のためのサポートを提供しています: 英語、ドイツ語、スペイン語、フランス語、および日本語。これらの言語のローカライズされたバージョンを作成する必要はありません。これらの言語をデフォルトの言語として設定するには、RaptorXML Server の setdeflang コマンドを使用します。

4.8.2 setdeflang

構文と説明

setdeflang コマンド (**sdl** と略される) はRaptorXML Server. のデフォルトの言語を設定します。英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、および日本語 (ja) を使用することができます。コマンドは必須の *LanguageCode* 引数を取ります。

```
raptorxml setdeflang [options] LanguageCode
raptorxmlserver setdeflang [options] LanguageCode
```

- *LanguageCode* 引数が必須で、RaptorXML Server のデフォルトの言語を設定します。対応する値は以下の通りです :en, de, es, fr, ja.
- The **setdeflang** コマンドは、以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.
- **--h, --help** オプションを使用してコマンドの情報を表示します。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム (W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

例

setdeflang (**sdl**) コマンドのサンプル:

```
raptorxml sdl de
raptorxml setdeflang es
raptorxmlserver setdeflang es
```

- 最初のコマンド RaptorXML Server のデフォルトの言語をドイツ語に設定します。
- 2番目のコマンドはRaptorXML Server のデフォルトの言語をスペイン語に設定します。
- 3番目のコマンドは2番目のコマンドと同じですが、サーバー実行可能ファイルにより実行されます。

オプション

--h, --help オプションを使用してコマンドに関する情報を表示します。

4.9 ライセンスコマンド

このセクションはRaptorXML Server にライセンスを供与するために使用されるコマンドについて説明しています:

- licenseserver 167 はネットワーク上のAltova LicenseServer にRaptorXML Server を登録します。
- assignlicense 168 はLicenseServer (Windows のみ)にライセンスファイルをアップロードします。
- verifylicense 169 はRaptorXML Server のライセンス供与 (Windows のみ)を検証します。

メモ これらのコマンドを 管理コマンドのためのサーバー実行可能ファイル 171 を介して実行することができます。

Altova LicenseServer を使用して Altove 製品にライセンスを供与するための情報に関しては、Altova LicenseServer ドキュメントを参照してください。

4.9.1 licenseserver

構文と説明

licenseserver コマンドはRaptorXML Server をServer-Or-IP-Address 引数により指定されているAltova LicenseServer を使用して登録します。licenseserver コマンド実行されるには、2つのサーバー(RaptorXML Server と LicenseServer) が同じネットワーク上で作動しておりLicenseServer が作動している必要があります。RaptorXML Server を LicenseServer と登録する管理特権が必要になります。

raptorxml licenseserver [options] *Server-Or-IP-Address*
raptorxmlserver licenseserver [options] *Server-Or-IP-Address*

- *Server-Or-IP-Address* 引数は、LicenseServer マシンの名前、または、IP アドレスを取ります。
- **licenseserver** コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

RaptorXML Server がLicenseServer へ登録されると、この件に関連するメッセージを受け取ります。メッセージはLicenseServer のURL も表示します。LicenseServer に移動して、RaptorXML Server にライセンスを割り当てることもできます。ライセンスに関する詳細は、LicenseServer ドキュメントを確認してください(https://www.altova.com/manual/ja/AltovaLicenseServer/)。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

例

licenseserver コマンドのサンプル:

```
raptorxml licenseserver DOC.altova.com
raptorxml licenseserver localhost
raptorxml licenseserver 127.0.0.1
raptorxmlserver licenseserver 127.0.0.1
```

上記のコマンドは、それぞれ `DOC.altova.com` という名前のマシンとユーザーのマシン (`localhost` と `127.0.0.1`) を Altova LicenseServer を作動するマシンとして指定します。双方の場合、指定されているマシン上の LicenseServer を使用してコマンドは RaptorXML Server を登録します。最後のコマンドは、コマンドを実行するためにサーバー実行可能ファイルを呼び出します。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために - または 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ json [j]

--j, --json = `true|false`

値は `true|false` です。`true` の場合、マシン解読可能な JSON オブジェクトとして登録の結果を出力します。

4.9.2 assignlicense (Windows only)

構文と説明

assignlicense コマンドは Windows 上のみで使用することができます (Linux または Mac システム上では使用することができません)。RaptorXML Server が登録されている Altova LicenseServer にライセンスファイルをアップロードし、RaptorXML Server にライセンスを割り当てます (licenseserver コマンドを参照してください)。ライセンスのパスを引数として取ります。コマンドによりライセンスの有効性をテストすることができます。

```
raptorxml assignlicense [options] FILE
raptorxmlserver assignlicense [options] FILE
```

- *FILE* 引数はライセンスファイルのパスを取ります。
- `--test-only` オプションは LicenseServer にライセンスをアップロードし、ライセンスを検証しますが、RaptorXML Server にライセンスを割り当てません。
- **assignlicense** コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

ライセンスに関する詳細は、LicenseServer ドキュメントを確認してください (https://www.altova.com/manual/ja/AltovaLicenseServer/)。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の **RaptorXML** (および管理コマンドのための **RaptorXMLServer**)
Windows と Unix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための **raptorxmlserver**)

＊小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字 (RaptorXML) は Windows と Mac のみで使用することができます。
＊Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

▼ Windows システム上でのバックスラッシュ、スペース、および特別文字

Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字がが文字列内で発生すると引用符を使用してください！例えば、"My File"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"C:\My directory\") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: \\"。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: "C:\My Directory\\"。

例

assignlicense コマンドのサンプル:

```
raptorxml assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxmlserver assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxml assignlicense --test-only=true C:\licensepool\mylicensekey.altova_licenses
```

- 上記の最初のコマンドは LicenseServer に指定されたライセンスをアップロードし、RaptorXML Server に割り当てます。
- 2番目のコマンドはサーバー実行可能ファイルを呼び出し、サーバー実行可能ファイルを呼び出し、最初のコマンドと同じように作動します。
- 最後のコマンドは LicenseServer に指定されたライセンスアップロードし、RaptorXML Server に割り当てることなく検証します。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: --option=value。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ test-only [t]

--t, --test-only = true|false
値は true|false です。true の場合、LicenseServer にライセンスファイルがアップロードされ、検証されますが、割り当てられることはありません。

4.9.3 verifylicense (Windows only)

構文と説明

verifylicense コマンドは、使用中の製品がライセンス供与を受けているかチェックします。更に、--license-key オプションにより特定のライセンスキーが製品に既に割り当て済みかをチェックします。このコマンドは Windows 上でのみサポートされています。Linux または Mac システム上ではサポートされていません。

```
raptorxml verifylicense [options]
raptorxmlserver verifylicense [options]
```

- RaptorXML Server に割り当てられた特定のライセンスが、`--license-key` オプションの値としてライセンスキーを与えるかをチェックします。
- **verifylicense** コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

ライセンスに関する詳細は、LicenseServer ドキュメントを確認してください (https://www.altova.com/manual/ja/AltovaLicenseServer/)。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

例

verifylicense コマンドのサンプル:

```
raptorxml verifylicenseserver
raptorxml verifylicenseserver --license-key=ABCD123-ABCD123-ABCD123-ABCD123-ABCD123-
ABCD123
raptorxmlserver verifylicenseserver --license-key=ABCD123-ABCD123-ABCD123-ABCD123-
ABCD123-ABCD123
```

- 最初のコマンドはRaptorXML Server がライセンス供与されているかをチェックします。
- 2番目のコマンドはRaptorXML Server が`--license-key` オプションで指定されているライセンスキーと共にライセンス供与されているかをチェックします。
- 3番目のコマンドは2番目のコマンドと同じですが、サーバー実行可能ファイルにより実行されます。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1 つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ license-key [l]

--l, --license-key = *Value*
RaptorXML Server がこのオプションの値としてライセンスキーと共にライセンス供与されているか確認します。

4.10 管理コマンド

(サービスとしてインストール、または、ライセンスコマンドなどの) 管理コマンドは(RaptorXMLServer という名前を持つ) RaptorXML Server のサーバー実行可能ファイルに対して発行されます。実行可能ファイルはデフォルトで以下で見つけることができます:

Windows	`<ProgramFilesFolder>\Altova\RaptorXMLServer2021\bin\`**`RaptorXMLServer.exe`**
Linux	`/opt/Altova/RaptorXMLServer2021/bin/`**`raptorxmlserver`**
Mac	`/usr/local/Altova/RaptorXMLServer2021/bin/`**`raptorxmlserver`**

使用法

コマンドの構文は以下の通りです:

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

- `--help` (短縮形 `--h`) は与えられたコマンドのヘルプテキストを表示します。コマンドに名前がない場合、実行可能ファイルの全てのコマンドがコマンドの短い説明と共にリストされます。
- `--version` はRaptorXML Server バージョン番号を表示しています。
- `<command>` は実行するコマンドです。対応するコマンドと共にリストされ説明されています。*(下のリスト参照)*。
- `[options]` はコマンドのオプションです。対応するコマンドと共にリストされ説明されています。
- `[arguments]` はコマンドの引数です。対応するコマンドと共にリストされ説明されています。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**`RaptorXML`** (および管理コマンドのための**`RaptorXMLServer`**)
Windows とUnix (Linux, Mac) 上の **`raptorxml`** (および管理コマンドのための**`raptorxmlserver`**)

* 小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はWindows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

管理コマンド

サーバー実行可能ファイルのコマンドは管理機能を提供します。以下にリストされており、このセクションのサブセクションで詳しく説明されています:

- install 172
- uninstall 172
- start 173
- setdeflang 174
- licenseserver 175
- assignlicense 176
- verifylicense 177
- createconfig 178
- exportresourcestrings 179
- debug 180
- help 181

4.10.1 install

構文と説明

install コマンドはRaptorXML Server をサービスとしてサーバーマシンにインストールします。

```
raptorxmlserver install [options]
```

- RaptorXML Server をサービスとしてインストールしても、サービスは自動的に開始されないことに注意してください。サービスを開始するために、**start** コマンドを使用します。
- RaptorXML Server をサービスとしてアンインストールするには、**uninstall** コマンドを使用します。
- **--h, --help** オプションを使用してコマンドの情報を表示します。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

サンプル

install コマンドのサンプル:

```
raptorxmlserver install
```

4.10.2 uninstall

構文と説明

uninstall コマンドは、サーバーマシン上でRaptorXML Server をサービスとしてアンインストールします。

```
raptorxmlserver uninstall [options]
```

RaptorXML Server をサービスとして再度インストールするには、**install** コマンドを使用します。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

例

uninstall コマンドのサンプル:

```
raptorxmlserver uninstall
```

4.10.3 start

構文と説明

start コマンドはRaptorXML Server をサービスとしてサーバーマシン上で開始します。

```
raptorxmlserver start [options]
```

- If RaptorXML Server はサービスとしてインストールされませんinstall コマンドとして(起動する前に)最初にインストールしてください。
- To uninstall RaptorXML Server をサービスとしてアンインストールするには、**uninstall** コマンドを使用します。
- **--h, --help** オプションを使用してコマンドの情報を表示します。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で)スペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

サンプル

start コマンドのサンプル:

```
raptorxmlserver start
```

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のためにコ つまたは2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ config [c]

--c, --config = *File*
構成ファイルへのパスを指定します。

▼ fork

--fork = true|false

Unix サーバー上でクラシック `init` を使用する場合、分岐を使用する機能を提供します。デフォルトは `false` です。

- port

  `--port =` *`PortNumber`*
  RaptorXML Server のデバッグインスタンスのポート番号です。

4.10.4 setdeflang

構文と説明

`setdeflang` コマンド (**`sdl`** と略される) は RaptorXML Server. のデフォルトの言語を設定します。英語 (`en`)、ドイツ語 (`de`)、スペイン語 (`es`)、フランス語 (`fr`)、および日本語 (`ja`) を使用することができます。コマンドは必須の *`LanguageCode`* 引数を取ります。

```
raptorxml setdeflang [options] LanguageCode
raptorxmlserver setdeflang [options] LanguageCode
```

- *`LanguageCode`* 引数が必須で、RaptorXML Server のデフォルトの言語を設定します。対応する値は以下の通りです :`en`, `de`, `es`, `fr`, `ja`.
- The **`setdeflang`** コマンドは、以下の実行可能ファイルから呼び出すことができます: **`raptorxml`** または **`raptorxmlserver`**.
- **`--h, --help`** オプションを使用してコマンドの情報を表示します。

- コマンドライン上の大文字と小文字の区別とスラッシュ

  W indows 上の**`RaptorXML`** (および管理コマンドのための**`RaptorXMLServer`**)
  W indows とUnix (Linux, Mac) 上の **`raptorxml`** (および管理コマンドのための**`raptorxmlserver`**)

  * 小文字 (`raptorxml` と `raptorxmlserver`) はすべてのプラットフォーム (W indows、Linux、および Mac) で使用することができますが、小文字(`RaptorXML`) は W indows と Mac のみで使用することができます。
  * Linux と Mac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

例

`setdeflang` (**`sdl`**) コマンドのサンプル:

```
raptorxml sdl de
raptorxml setdeflang es
raptorxmlserver setdeflang es
```

- 最初のコマンド RaptorXML Server のデフォルトの言語をドイツ語に設定します。
- 2番目のコマンドは RaptorXML Server のデフォルトの言語をスペイン語に設定します。
- 3番目のコマンドは2番目のコマンドと同じですが、サーバー実行可能ファイルにより実行されます。

オプション

`--h, --help` オプションを使用してコマンドに関する情報を表示します。

4.10.5 licenseserver

構文と説明

licenseserver コマンドはRaptorXML Server をServer-Or-IP-Address 引数により指定されているAltova LicenseServer を使用して登録します。licenseserver コマンド実行されるには、2つのサーバー(RaptorXML Server と LicenseServer) が同じネットワーク上で作動しておりLicenseServer が作動している必要があります。RaptorXML Server を LicenseServer と登録する管理特権が必要になります。

```
raptorxml licenseserver [options] Server-Or-IP-Address
raptorxmlserver licenseserver [options] Server-Or-IP-Address
```

- *Server-Or-IP-Address* 引数は、LicenseServer マシンの名前、または、IP アドレスを取ります。
- **licenseserver** コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

RaptorXML Server がLicenseServer へ登録されると、この件に関連するメッセージを受け取ります。メッセージはLicenseServer のURL も表示します。LicenseServer に移動して、RaptorXML Server にライセンスを割り当てることもできます。ライセンスに関する詳細は、LicenseServer ドキュメントを確認してください(https://www.altova.com/manual/ja/AltovaLicenseServer/)。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

▼ W indows システム上でのバックスラッシュ、スペース、および特別文字

W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

例

licenseserver コマンドのサンプル:

```
raptorxml licenseserver DOC.altova.com
raptorxml licenseserver localhost
raptorxml licenseserver 127.0.0.1
raptorxmlserver licenseserver 127.0.0.1
```

上記のコマンドは、それぞれDOC.altova.com という名前のマシンとユーザーのマシン(localhost と127.0.0.1) をAltova LicenseServer を作動するマシンとして指定します。双方の場合、指定されているマシン上のLicenseServer を使用してコマンドはRaptorXML Server を登録します。最後のコマンドは、コマンドを実行するためにサーバー実行可能ファイルを呼び出します。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用すること

ができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

- json [j]

  --j, --json = true|false
  値はtrue|false です。true の場合、マシン解読可能な JSON オブジェクトとして登録の結果を出力します。

4.10.6 assignlicense (Windows only)

構文と説明

assignlicense コマンドはW indows 上のみで使用することができます (Linux またはMac システム上では使用することができません)。RaptorXML Server が登録されているAltova LicenseServer にライセンスファイルをアップロードし、RaptorXML Server にライセンスを割り当てます (licenseserver コマンドを参照してください)。ライセンスのパスを引数として取ります。コマンドによりライセンスの有効性をテストすることができます。

```
raptorxml assignlicense [options] FILE
raptorxmlserver assignlicense [options] FILE
```

- *FILE* 引数はライセンスファイルのパスを取ります。
- --test-only オプションはLicenseServer にライセンスをアップロードし、ライセンスを検証しますが、RaptorXML Server にライセンスを割り当てません。
- **assignlicense** コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

ライセンスに関する詳細は、LicenseServer ドキュメントを確認してください (https://www.altova.com/manual/ja/AltovaLicenseServer/)。

- コマンドライン上の大文字と小文字の区別とスラッシュ

  W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
  W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

  * 小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム (W indows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はW indows とMac のみで使用することができます。
  * Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

- W indows システム上でのバックスラッシュ、スペース、および特別文字

  W indows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

例

assignlicense コマンドのサンプル:

```
raptorxml assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxmlserver assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxml assignlicense --test-only=true C:\licensepool\mylicensekey.altova_licenses
```

- 上記の最初のコマンドはLicenseServer に指定されたライセンスをアップロードし、RaptorXML Server に割り当てます。
- 2番目のコマンドはサーバー実行可能ファイルを呼び出し、サーバー実行可能ファイルを呼び出し、最初のコマンドと同じように作動します。
- 最後のコマンドはLicenseServer に指定されたライセンスアップロードし、RaptorXML Server に割り当てることなく検証します。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

▼ test-only [t]

--t, --test-only = `true|false`
値は`true|false` です。`true` の場合、LicenseServer にライセンスファイルがアップロードされ、検証されますが、割り当てられることはありません。

4.10.7 verifylicense (Windows only)

構文と説明

verifylicense コマンドは、使用中の製品がライセンス供与を受けているかチェックします。更に、`--license-key` オプションにより特定のライセンスキーが製品に既に割り当て済みかをチェックします。このコマンドはW indows 上でのみサポートされています。Linux または Mac システム上ではサポートされていません。

```
raptorxml verifylicense [options]
raptorxmlserver verifylicense [options]
```

- RaptorXML Server に割り当てられた特定のライセンスが、`--license-key` オプションの値としてライセンスキーを与えるかをチェックします。
- **verifylicense** コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

ライセンスに関する詳細は、LicenseServer ドキュメントを確認してください (https://www.altova.com/manual/ja/AltovaLicenseServer/)。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

W indows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
W indows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(W indows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はW indows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、W indows 上ではバックスラッシュを使用します。

例

verifylicense コマンドのサンプル:

```
raptorxml verifylicenseserver
raptorxml verifylicenseserver --license-key=ABCD123-ABCD123-ABCD123-ABCD123-ABCD123-
ABCD123
raptorxmlserver verifylicenseserver --license-key=ABCD123-ABCD123-ABCD123-ABCD123-
ABCD123-ABCD123
```

- 最初のコマンドはRaptorXML Server がライセンス供与されているかをチェックします。
- 2番目のコマンドはRaptorXML Server が`--license-key` オプションで指定されているライセンスキーと共にライセンス供与されているかをチェックします。
- 3番目のコマンドは2番目のコマンドと同じですが、サーバー実行可能ファイルにより実行されます。

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために1つまたは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は `TRUE` です。**`--h, --help`** オプションを使用してコマンドに関する情報を表示します。

▼ license-key [l]

--l, --license-key = *`Value`*
RaptorXML Server がこのオプションの値としてライセンスキーと共にライセンス供与されているか確認します。

4.10.8 createconfig

構文と説明

createconfig コマンドはデフォルトの値を持つサーバー構成ファイルを上書きします。

```
raptorxmlserver createconfig [options]
```

- **--lang** オプションはサーバー構成ファイルのデフォルト言語を指定します。

サーバー構成ファイルに関する情報は、サーバーの構成 203 を参照してください。

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

＊小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はWindows とMac のみで使用することができます。
＊Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

サンプル

createconfig コマンドのサンプルです:

```
raptorxml createconfig
raptorxml createconfig --lang=de
```

オプション

- lang

  --lang = en|de|es|fr|ja
  サーバー構成ファイルのデフォルトの言語を指定します。次のオプションを使用することができます: 英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、日本語 (ja)。オプションが指定されていない場合、英語がデフォルトの言語として選択されます。

--h, --help オプションを使用してコマンドに関する情報を表示します。

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のためにコマンドは2つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで2つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

4.10.9 exportresourcestrings

構文と説明

exportresourcestrings コマンドは特定の言語内のRaptorXML Server アプリケーションのソース文字列を含むXML ファイルを出力します。使用することのできる言語は英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、と日本語 (ja) です。

```
raptorxml exportresourcestrings [options] LanguageCode XMLOutputFile
raptorxmlserver exportresourcestrings [options] LanguageCode XMLOutputFile
```

- ***LanguageCode*** 引数は、出力 XML ファイル内のソース言語の言語を与えます。これは *エクスポート言語* です。許可されているエクスポート言語は以下の通りです (それぞれの言語コードがかっこで囲まれています) : 英語 (en)、ドイツ語 (de)、スペイン語 (es)、フランス語 (fr)、と日本語 (ja)。
- ***XMLOutputFile*** 引数は出力 XML ファイルのパスと名前を指定します
- **exportresourcestrings**コマンドを以下の実行可能ファイルから呼び出すことができます: **raptorxml** または **raptorxmlserver**.

ローカリゼーションの作成方法は以下のとおりです。

- コマンドライン上の大文字と小文字の区別とスラッシュ

  Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
  Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

  * 小文字 (raptorxml とraptorxmlserver) はすべてのプラットフォーム (Windows、Linux、およびMac) で使用することができますが、小文字(RaptorXML) はWindows とMac のみで使用することができます。
  * Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

- Windows システム上でのバックスラッシュ、スペース、および特別文字

  Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内でスペース、または特別文字がが文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ

(例えば、"C:\My directory\") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイルパスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

例

exportresourcestrings コマンドのサンプル:

```
raptorxml exportresourcestrings de c:\Strings.xml
raptorxmlserver exportresourcestrings de c:\Strings.xml
```

- 上の最初のコマンド はRaptorXML Server のドイツ語リソース文字列を含む `Strings.xml` と呼ばれるファイルを `c:\` に作成します。
- 2番目のコマンドはサーバー実行可能ファイルを呼び出し最初のコマンドと同様に作動します。

RaptorXML Server のローカライズされたバージョンの作成

選択された言語で RaptorXML Server のローカライズされたバージョンを作成することができます。5つのローカライズされたバージョン(英語、ドイツ語、スペイン語、フランス語、および日本語) は `C:\Program Files (x86)\Altova\RaptorXMLServer2021\bin` フォルダー内で使用することができるため、作成する必要はありません。

ローカライズされたバージョンを以下のように作成します:

1. `exportresourcestrings` コマンドを使用してリソース文字列を含む XML ファイルを作成します(*上記のコマンド構文を参照してください*)。この XML ファイル内のリソース文字列は、サポートされる以下の5つの言語のひとつです: コマンドと共に使用される LanguageCode 引数に従い英語 (`en`)、ドイツ語 (`de`)、スペイン語 (`es`)、フランス語 (`fr`)、または日本語 (`ja`)。
2. 5つのサポートされる言語の1つからのリソース文字列からターゲット言語に翻訳します。リソース文字列は XML ファイル内の `<string>` 要素のコンテンツです。`{option}` または `{product}` などの、中かっこ内の変数は翻訳しないでください。
3. ローカライズされた RaptorXML Server DLL ファイルを翻訳された XML ファイルから生成するために、Altova サポート に連絡してください。
4. Altova サポート からローカライズされた DLL ファイルを受信すると、DLL を `C:\Program Files (x86)\Altova\<%APPFOLDER\bin` フォルダー内に保存します。DLL ファイルはフォーム `RaptorXML2021_lc.dll` の名前を持ちます。名前の `_lc` 部分には、言語コードが含まれています。例えば、 `RaptorXML2021_de.dll` では、`de` の部分は、ドイツ語 (Deutsch) のための言語コードです。
5. `setdeflang` コマンドを実行し、使用する RaptorXML Server アプリケーションと同様にローカライズされた DLL ファイルを設定します。`setdeflang` コマンドの引数のために、DLL 名の一部である言語コードを使用してください。

メモ Altova RaptorXML Server は5つの言語のためのサポートを提供しています: 英語、ドイツ語、スペイン語、フランス語、および日本語。これらの言語のローカライズされたバージョンを作成する必要はありません。これらの言語をデフォルトの言語として設定するには、RaptorXML Server の `setdeflang` コマンドを使用します。

4.10.10 debug

構文と説明

debug コマンドはデバッグのための RaptorXML Server をサービスとしてではなく開始します。このモードでの RaptorXML Server を停止するには、「Ctrl+C」を押します。

```
raptorxmlserver debug [options]
```

- コマンドライン上の大文字と小文字の区別とスラッシュ

  Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
  Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

  * 小文字 (raptorxml と raptorxmlserver) はすべてのプラットフォーム (Windows、Linux、および Mac) で使用することができますが、小文字(RaptorXML) は Windows と Mac のみで使用することができます。
  * Linux と Mac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

- Windows システム上でのバックスラッシュ、スペース、および特別文字

  Windows システム (例えば、ファイル、または、フォルダー名、企業名、または、個人、または、製品名内で) スペース、または特別文字が文字列内で発生すると引用符を使用してください! 例えば、"**My File**"。しかしながら、二重引用符の後のバックスラッシュ (例えば、"**C:\My directory**") は正確に読み取られない場合があります。これはバックスラッシュ文字がエスケープシーケンスの開始を示すためであり、エスケープシーケンス ¥" は二重引用符文字を示すためです。この文字のシーケンスをエスケープする場合は、以下のように前にバックスラッシュを次のように使用します: **\\"**。 要約すると スペースまたはバックスラッシュを含むファイル パスを作成する場合、以下のような名前を与えます: **"C:\My Directory\\"**。

サンプル

debug コマンドのサンプル:

```
raptorxmlserver debug
```

オプション

(使用できる場合) オプションは短いおよび長い書式でリストされています。短いおよび長い書式のために 1 つまたは 2 つのダッシュを使用することができます。オプションは値を取る、または、取らない場合があります。値をとる場合、以下のようになります: **--option=value**。値は引用符無しで 2 つの場合で指定することができます: (i) 値文字列にスペースが含まれる場合。または、(ii) 引用符が必要とするオプションの説明内で明示的に指示されている場合。オプションがブール値の値を取り、ブールの値が指定されていない場合、オプションのデフォルトの値は TRUE です。**--h, --help** オプションを使用してコマンドに関する情報を表示します。

- config [c]

  --c, --config = *File*
  構成ファイルへのパスを指定します。

- port

  --port = *PortNumber*
  RaptorXML Server のデバッグインスタンスのポート番号です。

4.10.11 help

構文と説明

help コマンドは、ヘルプを必要とするコマンドの名前である単一の引数 (Command) を取ります。コマンドの構文、オプション、および他の関連する情報を表示します。Command 引数が指定されていない場合、説明と共に実行可能ファイルのすべてのコマンドがリストされます。**help** コマンドは実行可能ファイルから呼び出すこともできます: **raptorxml** または **raptorxmlserver**。

```
raptorxml help Command
raptorxmlserver help Command
```

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はWindows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

例

help コマンドのサンプルは**licenserver** コマンドに関する情報を表示します (このコマンドは両方の実行可能ファイルで使用することができます):

```
raptorxml help licenseserver
raptorxmlserver help licenseserver
```

--help オプション

ヘルプ情報が必要とされるコマンドの`--help` オプションを使用してヘルプ情報を使用することもできます。下の2 つのコマンドは同じ結果を生成します:

```
raptorxml licenseserver --help
```

上記のコマンドは**licenseserver** コマンドの`--help` オプションを使用します。

```
raptorxml help licenseserver
```

help コマンドは引数として`licenseserver` を取ります。

両方のコマンドは**licenseserver** コマンドに関するヘルプ情報を表示します。

4.10.12 version

構文と説明

version コマンドはRaptorXML Server のバージョンを表示します。実行可能ファイルから呼び出すことも可能です: **raptorxml** または**raptorxmlserver**。

```
raptorxml version
raptorxmlserver version
```

▼ コマンドライン上の大文字と小文字の区別とスラッシュ

Windows 上の**RaptorXML** (および管理コマンドのための**RaptorXMLServer**)
Windows とUnix (Linux, Mac) 上の **raptorxml** (および管理コマンドのための**raptorxmlserver**)

* 小文字 (`raptorxml` と`raptorxmlserver`) はすべてのプラットフォーム(Windows、Linux、およびMac) で使用することができますが、小文字(`RaptorXML`) はWindows とMac のみで使用することができます。
* Linux とMac 上ではスラッシュを使用し、Windows 上ではバックスラッシュを使用します。

例

`version` コマンドのサンプル:

```
raptorxml version
raptorxmlserver version
```

4.11 オプション

このセクションは全てのオプションの説明を含んでおり、機能別にグループ化されています。 各コマンドと共に使用されるオプションを検索するには、対応するコマンドの詳細を参照してください。

- カタログ、グローバルリソース、ZIP ファイル 184
- メッセージ、エラー、ヘルプ 185
- 処理 186
- XML 187
- XSD 188
- XQuery 190
- XSLT 192
- JSON/Avro 194
- XML 署名 194

4.11.1 カタログ、グローバルリソース、ZIP ファイル

▼ catalog

--catalog = *FILE*
インストールされたルートカタログファイルではないルートカタログファイルへの絶対パスを指定します。デフォルト値はインストールされたルートカタログファイルへの絶対パス (`<installation-folder>\Altova\RaptorXMLServer2021\etc\Rootcatalog.xml`) です。カタログとの作業の詳細に関しては、XML カタログ 40 のセクションを参照してください。

▼ user-catalog

--user-catalog = *FILE*
ルートカタログに追加して使用される XML カタログへの絶対パスを指定します。のセクションを参照してください。カタログとの作業についての追加情報は、XML カタログ 40 を参照してください。

▼ enable-globalresources

--enable-globalresources = `true|false`
グローバルリソース 49 を無効化します。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

▼ globalresourceconfig [gc]

--gc | --globalresourceconfig = *VALUE*
グローバルリソースのアクティブな構成 49 を指定します(グローバルリソース 49 を有効化します) 49 。

▼ globalresourcefile [gr]

--gr | --globalresourcefile = *FILE*
グローバルリソースファイル 49 を指定します。(グローバルリソース 49 を有効化します) 49 。

▼ recurse

--recurse = `true|false`
ZIP アーカイブ内のファイルを選択するために使用されます。`true` の場合、コマンドの *InputFile* 引数は指定されたファイルをサ

ブディレクトリでも選択します。例: test.zip|zip\test.xml はtest.xml という名前のファイル名を、Zip フォルダーの全てのフォルダーのレベルで選択します。ZIP ファイルへのレファレンスは引用符を用いて与えられている必要があります。 * および ? などのワイルドカード文字が使用されるかもしれません。ですから*.xml は、Zip フォルダー内のすべての .xml ファイルを選択します。オプションのデフォルト値はfalse です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

4.11.2 メッセージ、エラー、ヘルプ、タイムアウト、バージョン

- error-format

  --error-format = text|shortxml|longxml
  エラー出力のフォーマットを指定します。デフォルト値はtext です。他のオプションはlongxml と共に詳細付きのXML フォーマットを生成します。

- error-limit

  --error-limit = *N | unlimited*
  エラー制限を指定します。デフォルト値は100 です。 1 から999 の値は許可されています。検証中のプロセッサの使用を制限する際に役に立ちます。エラーの制限に達すると、検証は停止されます。

- help

  --help
  コマンドのヘルプテキストを表示します。例えば、val any --h。(または、help コマンドは引数と共に使用することができます。例: help val any。)

- info-limit

  --info-limit = *N | unlimited*
  範囲 1-65535 またはunlimited 内の情報メッセージの制限を指定します。情報制限が達成されると処理は継続されますが更なるメッセージはレポートされません。デフォルトの値は100 です。

- log-output

  --log-output = *FILE*
  指定されたファイルURL にログ出力を書き込みます。CLI が書き込みアクセス許可があることを確認してください。

- network-timeout

  --network-timeout = *VALUE*
  リモート I/O オペレーションのタイムアウトを秒で指定します。デフォルト: 40。

- verbose

  --verbose = true|false
  true の値は、検証中の追加情報の出力を有効化します。デフォルト値はfalse です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、true に設定されています。

- verbose-output

  --verbose-output = *FILE*
  FILE に詳細出力を書き込みます。

▼ version

--version
RaptorXML Server のバージョンを表示します。. コマンドと共に使用される場合、`--version` をコマンドの前に置きます。

▼ warning-limit

--warning-limit = *N | unlimited*
1-65535 範囲内で警告のリミットを指定します。処理は、このリミットに到達しても継続されますが、更なる警告はレポートされません。デフォルトの値は `100` です。

4.11.3 処理

▼ listfile

--listfile = `true|false`
`true` の場合、コマンドの *InputFile* 引数を各ラインに 1 つのファイル名を含むテキストファイルとして扱います。デフォルト値は `false` です。(代替としてはスペースを区切りとして使用し CLI 上にファイルをリストすることです。しかしながら、CLI には最高文字数の制限があることに注意してください。) `--listfile` オプションは引数のみに適用することができ、オプションには適用することができないことに注意してください。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

▼ parallel-assessment [pa]

--pa | --parallel-assessment = `true|false`
`true` に設定されている場合、パラレルにスキーマの有効性の評価が実行されます。これは、あるレベルに 128 個以上の要素が存在する場合、複数のスレッドを使用してこれらの要素が処理されることを意味します。ですから、大きな XML ファイルは、このオプションが有効化されている場合より早く処理されることができます。パラレル評価は、階層的なレベルごとに実行されますが、単一のインフォセットでは、複数のレベルで実行されることもできます。パラレル評価はストリーミングモードでは実行することができません。このため、`--streaming` オプションは `--parallel-assessment` が `true` に設定されている場合、無視されます。また、メモリ使用は `--parallel-assessment` オプションが使用される場合高いことに注意してください。 デフォルトの設定は `false` です。オプションの短いフォームは `--pa` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

▼ script

--script = *FILE*
検証が完了した後、提出されたファイル内の Python スクリプトを実行します。1 つ以上のスクリプトを指定するため、オプションを複数回追加します。

▼ script-api-version

`--api, --script-api-version =` `1; 2; 2.1; 2.2; 2.3; 2.4; 2.4.1; 2.5; 2.6; 2.7; 2.8; 2.8.1; 2.8.2`
スクリプトで使用される Python API バージョンを指定します。デフォルト値は現在 **`2.8.2`** である最新バージョンです。`1` および `2` の値の代わりに、それぞれ値 `1.0` および `2.0` を使用することができます。

▼ script-param

--script-param = *KEY:VALUE*
Python スクリプト実行中にアクセスすることのできる、追加ユーザー指定パラメーター。1 つ以上のスクリプトパラメーターを指定するため、オプションを複数回追加します。

▼ streaming

--streaming = `true|false`
ストリーミング検証を有効化します。デフォルトは `true` です。ストリーミングモードでは、メモリに保管されるデータが最小化され、処理がより速くなります。欠点は、後に情報が必要になる可能性があります。例えば、XML インスタンスドキュメントのデータモデルが使用できない場合があります。この様なシチュエーションでは、ストリーミングモードは `--streaming` に `false` の値を与え、)オフにされる必要があります。 `valxml-withxsd` コマンドを使用して、`--script` オプションを使用するとストリーミングを無効化することができます。`--streaming` オプションは、`--parallel-assessment` が `true` に設定されている場合、の場合無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xml-validation-error-as-warning

--xml-validation-error-as-warning = `true|false`
検証エラーを警告として扱うかを指定します。警告として扱われる場合、エラーが検出されても、XSLT 変換などの追加処理は継続されます。デフォルトは `false` です。

4.11.4 XML

- assessment-mode

--assessment-mode = `lax|strict`
XSD 仕様で定義されているようにスキーマの有効性評価モードを指定します。デフォルト値は `strict` です。 XML インスタンスドキュメントはこのオプションで指定されたモードに従い検証されます。

- dtd

--dtd = *`FILE`*
検証に使用される外部 DTD ドキュメントを指定します。XML ドキュメント内に外部 DTD への参照が存在する場合、CLI オプションが外部参照を上書きします。

- load-xml-with-psvi

--load-xml-with-psvi = `true|false`
入力 XML ファイルを有効化し、スキーマ検証後の情報を生成します。デフォルト: `true`。

- namespaces

--namespaces = `true|false`
名前空間対応処理を有効化します。これは、XML インスタンス内の間違った名前空間のため発生するエラーをチェックするために役に立ちます。デフォルト値は `false` です。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xinclude

--xinclude = `true|false`
XML Inclusions (XInclude) へのサポートを有効化します。デフォルト値は `false` です。 `false` の場合、 XInclude の `include` 要素は無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- xml-mode

--xml-mode = `wf|id|valid`
XML インスタンスドキュメントのために使用される XML 処理モードを指定します: `wf`=整形式のチェック; `id`=ID/IDREF チェックと共に整形式のチェック; `valid`=検証。デフォルト値は `wf` です。`valid` の値は DTD の参照の処理中に各スキーマドキュメントが

ロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = `true|false`
  検証エラーを警告として扱うかを指定します。警告として扱われる場合、エラーが検出されても、XSLT 変換などの追加処理は継続されます。デフォルトは `false` です。

- xsd

  --xsd = *`FILE`*
  1 つまたは複数の XML スキーマドキュメント を XML インスタンスの検証に使用することを指定します。1 つ以上のスキーマドキュメントを指定するため、オプションを複数回追加します。

4.11.5 XSD

- assessment-mode

  --assessment-mode = `lax|strict`
  XSD 仕様で定義されているようにスキーマの有効性評価モードを指定します。デフォルト値は `strict` です。 XML インスタンスドキュメントはこのオプションで指定されたモードに従い検証されます。

- ct-restrict-mode

  --ct-restrict-mode = `1.0|1.1|default`
  複合型の制約のチェック方法を指定します。**`1.0`** の値は、XSD 1.1 検証モードであっても、XSD 1.0 仕様に従い複合型の制約をチェックします。**`1.1`** の値は、XSD 1.0 検証モードであっても、XSD 1.1 仕様に従い複合型の制約をチェックします。**`default`** の値は、現在の検証モード (1.0 または 1.1) 内の仕様に従い複合型を検証します。デフォルトの値は **`default`** です。

- namespaces

  --namespaces = `true|false`
  名前空間対応処理を有効化します。これは、XML インスタンス内の間違った名前空間のため発生するエラーをチェックするために役に立ちます。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

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

  --report-import-namespace-mismatch-as-warning = `true|false`
  `xs:import` を使用してエラーから警告にスキーマをインポートする際、名前空間またはターゲット名前空間の不一致エラーをダウングレードします。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- schema-imports

  --schema-imports = `load-by-schemalocation | load-preferring-schemalocation | load-by-namespace | load-combining-both | license-namespace-only`
  それぞれオプションの `namespace` 属性とオプションの `schemaLocation` 属性を持つ、`xs:import` 要素の振る舞いを指定します: `<import namespace="someNS" schemaLocation="someURL"/>`。オプションはスキーマドキュメントをロードするかまたは名前空間にライセンスを与えるかを指定します。スキーマドキュメントがロードされる場合、どの情報が検索するために使用されるか指定されます。、デフォルト: `load-preferring-schemalocation`.
  振る舞いは以下の通りです:

- load-by-schemalocation: カタログマッピング 40 を考慮し、schemaLocation 属性の値がスキーマを検索するために使用されます。名前空間属性が存在する場合、名前空間はインポートされます (ライセンスが与えられます)。
- load-preferring-schemalocation: schemaLocation 属性が存在する場合、 カタログマッピング 40 を考慮して使用されます。schemaLocation 属性が存在しない場合、namespace 属性の値がカタログマッピング 40 を介して使用されます。スキーマこれはデフォルトの値です。
- load-by-namespace: カタログマッピング 40 を介して、namespace 属性の値がスキーマを検索するために使用されます。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング 40 がある場合、マッピングが使用されます。両方にカタログマッピング 40 , がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。カタログマッピング 40 が存在しない場合、schemaLocation 属性が使用されます。
- license-namespace-only: 名前空間はインポートされます。スキーマドキュメントはインポートされません。

schemalocation-hints

--schemalocation-hints = `load-by-schemalocation | load-by-namespace | load-combining-both | ignore`
xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性の振る舞いを指定します。: スキーマドキュメントをロードするか、またその場合、どの情報が検索に使用されるか。デフォルト: load-by-schemalocation。

- load-by-schemalocation 値はXML インスタンスドキュメント内のxsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 のスキーマの場所のURL 317 使用します。これはデフォルトの値です。
- load-by-namespace 値は、xsi:schemaLocation の名前空間の部分 317 を、そして xsi:noNamespaceSchemaLocation の場合は空の文字列を取ります。カタログマッピング 40 を介してスキーマを検索します。
- load-combining-both: namespace または schemaLocation 属性にカタログマッピング 40 がある場合、マッピングが使用されます。両方にカタログマッピング 40 , がある場合、--schema-mapping オプションの値が使用されます。(XML/XSD オプション 188) がどのマッピングが使用されるか決定します。名前空間またはURL がカタログマッピングを持たない場合、URL が使用されます。
- オプションの値がignore の場合、xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

schema-mapping

--schema-mapping = `prefer-schemalocation | prefer-namespace`
スキーマロケーションと名前空間がスキーマドキュメントの検索に使用される場合、カタログの検索中優先されるスキーマロケーションと名前空間を指定します。(--schemalocation-hints または--schema-imports オプションが load-combining-both の値を有する場合、また、関連する名前空間とURL のパートが双方カタログマッピング 40 を有する場合 、このオプションの値が使用される 2 つのマッピングを指定します (名前空間 マッピング または URL マッピング。prefer-schemalocation 値は URL マッピングを参照します)。) デフォルトは prefer-schemalocation です。

xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
XML スキーマドキュメントのために使用される XML 処理モードを指定します。: wf=整形式のチェック, id=ID/IDREF チェックを持つ整形式, valid=検証。デフォルトの値は wf です。valid の値はDTD の参照の処理中に各スキーマドキュメントがロードされることを必要とします。DTD が存在しない場合エラーは発生しません。

xsd-version

--xsd-version = `1.0|1.1|detect`
使用するW 3C スキーマ定義言語 (XSD) のバージョンを指定します。デフォルト 1.0 はです。また、このオプションはスキーマが、1.1-互換性ではなく1.0-互換性を有するかを検出する際に役に立ちます。検出オプションは、Altova 固有の機能です。XML スキーマドキュメントのバージョン (1.0 または 1.1) は、ドキュメントの <xs:schema> 要素の vc:minVersion 属性の値を読み込むことにより検出されます。@vc:minVersion 属性の値が1.1 の場合、スキーマはバージョン 1.1 として検出されます。他の値

に関しては、または、`@vc:minVersion` 属性が不在の場合、スキーマはバージョン 1.0 として検出されます。

4.11.6 XQuery

- indent-characters

  --indent-characters = *VALUE*
  インデントとして使用される文字列を指定します。

- input

  --input = *FILE*
  変換される XML ファイルの URL です。

- keep-formatting

  --keep-formatting = `true|false`
  ターゲットドキュメントのフォーマットを最大範囲可能な限り保持します。デフォルト: `true`。

- omit-xml-declaration

  --omit-xml-declaration = `true|false`
  シリアル化 オプションは XML 宣言が出力から省略されるかどうかを指定します。`true` の場合、出力ドキュメント内に XML 宣言はありません。`false` の場合、XML 宣言は含まれます。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- output, xsltoutput

  output = *FILE*, xsltoutput = *FILE*
  プライマリ出力ファイルの URL。例えば、複数のファイルの HTML 出力の場合、プライマリ出力のファイルは、エントリポイント HTML ファイルの場所になります。イメージファイルなどの、追加出力ファイルは、`xslt-additional-output-files` として報告されます。`--output` または `--xsltoutput` オプションが指定されていない場合、出力は標準の出力に書き込まれます。

- output-encoding

  --output-encoding = *VALUE*
  出力ドキュメント内のエンコード属性の値。有効な値は IANA 文字セットレジストリ内の名前です。デフォルト値は `UTF-8` です。

- output-indent

  --output-indent = `true|false`
  `true` の場合、出力は階層的構造に従いインデントされます。`false` の場合、階層形式のインデントはありません。デフォルトは `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- output-method

  --output-method = `xml|html|xhtml|text`
  出力フォーマットを指定します。デフォルト値は `xml` です。

- param [p]

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

外部パラメーターの値を指定します。内部パラメータは、`declare variable` を持ち、変数名、`external` キーワード、およびセミコロンが続く、XQuery ドキュメント内で宣言されています。例:

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

`external` キーワードである `$foo` は、外部パラメータとなり、その値は、ランタイム時に外部ソースからパスされます。外部パラメーターは、CLI コマンドにより値を与えられます。例:

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

上で説明されているように、*KEY* は外部パラメータの名前です。*VALUE* は、XPath 式として与えられた外部パラメーターの値です。の値です。 CLI で使用されたパラメーター名は、XQuery ドキュメント内で宣言される必要があります。複数の外部パラメータがCLI がでパスされた値である場合、それぞれに個別の `--param` オプションが必要になります。 XPath 式にスペースが含まれる場合二重引用符を使用する必要があります。

XSLT

グローバルスタイルシートのパラメーターを指定します。*KEY* はパラメータ名であり、*VALUE* は、パラメーターの値を与える XPath 式です。CLI で使用されるパラメーター名は、スタイルシート内で宣言される必要があります。複数のパラメータが使用される場合、`--param` スイッチが各パラメータ前に使用される必要があります。XPath 式内または式内の文字列でスペースが含まれる場合は、二重引用符が、XPath 式の周りで使用される必要があります。*例:*

```
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
```

updated-xml

--updated-xml = `discard|writeback|asmainresult`
更新された XML ファイルがどのように扱われるかを指定します。アップデートは以下のとおりです:

- 破棄されたまたはファイルに書き込まれない (`discard`)
- `--input` オプションにより指定されている入力 XML ファイルに書き込まれる (`writeback`)
- 標準の場所または `--output` オプションにより指定されている場所に保存される (定義されている場合)

デフォルト: `discard`。

xpath-static-type-errors-as-warnings

`--xpath-static-type-errors-as-warnings =` `true|false`
`true` の場合、警告を静的 XPath コンテキスト内で検知される型エラーにします。エラーは実行の失敗を引き起こしますが、警告は処理を継続することができます。デフォルトは、`false` です。

xquery-update-version

--xquery-update-version = `1|1.0|3|3.0|`
XQuery プロセッサがXQuery Update Facility 1.0 またはXQuery Update Facility 3.0 を使用するかを指定します。デフォルト値は `3` です。

xquery-version

--xquery-version = `1|1.0|3|3.0|3.1`
XSLT プロセッサがXSLT 1.0 またはXSLT 3.0. を使用するか指定します。デフォルト値は 3.1 です。

4.11.7 XSLT

- chartext-disable

  --chartext-disable = `true|false`
  チャート拡張子を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- dotnetext-disable

  --dotnetext-disable = `true|false`
  .NET 拡張子を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- indent-characters

  --indent-characters = *VALUE*
  インデントとして使用される文字列を指定します。

- input

  --input = *FILE*
  変換される XML ファイルの URL です。

- javaext-barcode-location

  --javaext-barcode-location = *FILE*
  バーコード拡張ファイル `AltovaBarcodeExtension.jar` を含むパスを指定します。. パスは次のフォームで与えられなければなりません:

  - ファイル URI の例: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2015/etc/jar/"`
  - バックスラッシュ付きのエスケープ文字を使用した Windows パスの例: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2015\\etc\\jar\\"`

- javaext-disable

  --javaext-disable = `true|false`
  Java 拡張子を無効化します。デフォルト値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- output, xsltoutput

  output = *FILE*, xsltoutput = *FILE*
  プライマリ出力ファイルの URL。例えば、複数のファイルの HTML 出力の場合、プライマリ出力のファイルは、エントリポイント HTML ファイルの場所になります。イメージファイルなどの、追加出力ファイルは、`xslt-additional-output-files` として報告されます。`--output` または `--xsltoutput` オプションが指定されていない場合、出力は標準の出力に書き込まれます。

- param [p]

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

  - ***XQuery***

    外部パラメーターの値を指定します。内部パラメータは、`declare variable` を持ち、変数名、`external` キーワード、およびセミコロンが続く、XQuery ドキュメント内で宣言されています。例:

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

    `external` キーワードである `$foo` は、外部パラメータとなり、その値は、ランタイム時に外部ソースからパスされます。外部パラ

メーターは、CLI コマンドにより値を与えられます。例:

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

上で説明されているように、*KEY* は外部パラメータの名前です。*VALUE* は、XPath 式として与えられた外部パラメーターの値です。の値です。 CLI で使用されたパラメーター名は、XQuery ドキュメント内で宣言される必要があります。複数の外部パラメータがCLI がでパスされた値である場合、それぞれに個別の `--param` オプションが必要になります。 XPath 式にスペースが含まれる場合二重引用符を使用する必要があります。

XSLT

グローバルスタイルシートのパラメーターを指定します。*KEY* はパラメータ名であり、*VALUE* は、パラメーターの値を与える XPath 式です。CLI で使用されるパラメーター名は、スタイルシート内で宣言される必要があります。複数のパラメータが使用される場合、`--param` スイッチが各パラメータ前に使用される必要があります。XPath 式内または式内の文字列でスペースが含まれる場合は、二重引用符が、XPath 式の周りで使用される必要があります。*例:*

```
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`
ストリーミング検証を有効化します。デフォルトは `true` です。ストリーミングモードでは、メモリに保管されるデータが最小化され、処理がより速くなります。欠点は、後に情報が必要になる可能性があります。例えば、XML インスタンスドキュメントのデータモデルが使用できない場合があります。この様なシチュエーションでは、ストリーミングモードは `--streaming` に `false` の値を与え、)オフにされる必要があります。 `valxml-withxsd` コマンドを使用して、`--script` オプションを使用するとストリーミングを無効化することができます。`--streaming` オプションは、`--parallel-assessment` が `true` に設定されている場合、の場合無視されます。
メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

initial-template, template-entry-point

--initial-template, --template-entry-point = *VALUE*
変換のエントリポイントである XSLT スタイルシート内の名前の付けられたテンプレートに名前を与えます。

initial-mode, template-mode

--initial-mode, --template-mode = *VALUE*
変換のために使用されるテンプレートモードを指定します。

xpath-static-type-errors-as-warnings

`--xpath-static-type-errors-as-warnings =` `true|false`
`true` の場合、警告を静的 XPath コンテキスト内で検知される型エラーにします。エラーは実行の失敗を引き起こしますが、警告は処理を継続することができます。デフォルトは、`false` です。

xslt-version

--xslt-version = `1|1.0|2|2.0|3|3.0|3.1`
XSLT プロセッサがXSLT 1.0、XSLT 2.0、またはXSLT 3.0. を使用するか指定します。デフォルト値は `3` です。

4.11.8 JSON/Avro

- additional-schema

  --additional-schema = *FILE*
  追加スキーマドキュメントのURI を指定します。追加スキーマはメインスキーマによりロードされ、追加スキーマ **id** または **$id** プロパティによりメインスキーマから参照することができます。

- disable-format-checks

  --disable-format-checks = `true|false`
  書式属性により指定されたセマンティック検証を無効化します。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- jsonc

  --jsonc = `true|false`
  JSON 内のコメントのためのサポートを有効化します。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- json-lines

  --json-lines = `true|false`
  JSON ライン (すなわち1 つのJSON 値が各ラインに表示される) のためのサポートが有効化されます。デフォルトの値は `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

4.11.9 XML 署名

- absolute-reference-uri

  --absolute-reference-uri = `true|false`
  署名済みドキュメントのURI が絶対 (`true`) または相対 (`false`) で読み取られるかを指定します。デフォルトは `false` です。
  メモ ブール値のオプションの値は、オプションに対しての値が設定されていない場合、`true` に設定されています。

- certname, **certificate-name**

  --certname, --certificate-name = *VALUE*
  署名のために使用される証明書の名前です。

  Windows
  選択された **--certificate-store** からの証明書の**サブジェクト**名です。

  (PowerShell の下)証明書のリストのサンプル
  ```
  % ls cert://CurrentUser/My
  PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
  Thumbprint Subject
  ---------- -------
  C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
  ... CN=...
  ```

  サンプル: **--certificate-name==certificate1**

Linux/MacOS
--certname はエンコードされる PEM の名前を指定します。X.509v3 c証明書には秘密キーが存在します。このようなファイルには通常拡張子 **.pem** がつきます。

サンプル: **--certificate-name==/path/to/certificate1.pem**

certstore, **certificate-store**

--certstore, --certificate-store = *VALUE*
--certificate-name を使用して証明書が保管される場所です。

Windows
cert://CurrentUser に保管される証明書ストアの名前です。(PowerShell の下) **% ls cert://CurrentUser/** を使用して使用することのできる証明書ストアをリストすることができます:

```
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
```

サンプル: **--certificate-store==MyCertStore**

Linux/MacOS
--certstore オプションは現在サポートされていません。

digest, digest-method

--digest, --digest-method = sha1|sha256|sha384|sha512
入力 XML ファイルのダイジェスト値を計算するために使用されるアルゴリズム。使用可能な値: sha1|sha256|sha384|sha512.

hmackey, hmac-secret-key

--hmackey, --hmac-secret-key = *VALUE*
HMAC 共有される秘密キー。最小6文字の長さが必要です。

サンプル: **--hmackey=secretpassword**

hmaclen, hmac-output-length

--hmaclen, --hmac-output-length = *LENGTH*
LENGTH ビットに HMAC アルゴリズムの出力を 切り捨てます。

- 8 の倍数
- 80 より大きい
- 元になるハッシュアルゴリズムの出力の長さの半分よりも大きい必要があります

keyinfo, append-keyinfo

--keyinfo, --append-keyinfo = true|false
署名に **KeyInfo** 要素を含むかを決定します。デフォルトは **false** です。

sigc14nmeth, signature-canonicalization-method

--sigc14nmeth, --signature-canonicalization-method = *VALUE*
SignedInfo 要素に適用する標準化アルゴリズムを指定します。値は以下の内の1つになります:

- REC-xml-c14n-20010315
- xml-c14n11
- xml-exc-c14n#

sigmeth, signature-method

--sigmeth, --signature-method = *VALUE*
署名を生成するために使用されるアルゴリズムを指定します。

証明書が使用される場合
証明書が指定される場合、--signature-method は任意で、このパラメータのための値は証明書から計算されます。指定されると、証明書により使用されるアルゴリズムに一致する必要があります。

サンプル: **--signature-method=rsa-sha256**

--hmac-secret-key が使用される場合
--hmac-secret-key が使用される場合、このオプションは必須です。HMAC アルゴリズムにサポートされる値である必要があります:

- hmac-sha256
- hmac-sha386
- hmac-sha512
- hmac-sha1 (仕様では使用が奨励されていません)

サンプル: **--signature-method=hmac-sha256**

sigtype, signature-type

--sigtype, --signature-type = `detached | enveloping | enveloped`
生成される署名の型を指定します。

transforms

--transforms = *VALUE*
入力ドキュメントに適用される XML 署名変換を指定します。このオプションは複数回指定することができ、指定の順序は重要です。最初に指定される変換は入力ドキュメントを受信します。最後に指定される変換はダイジェスト値の計算の直前に使用されます。

サポートされる値:

- Canonical XML 1.0 (コメント無し) のための `REC-xml-c14n-20010315`
- Canonical XML 1.1 (コメント無し) のための `xml-c14n11`
- Exclusive XML Canonicalization 1.0 (コメント無し) のための `xml-exc-c14n#`
- Canonical XML 1.0 (コメント有り) のための `REC-xml-c14n-20010315#WithComments`
- Canonical XML 1.1 (コメント有り) のための `xml-c14n11#WithComments`
- Exclusive XML Canonicalization 1.0 (コメント有り) のための `xml-exc-c14n#WithComments`
- `base64`
- `strip-whitespaces` Altova 拡張子

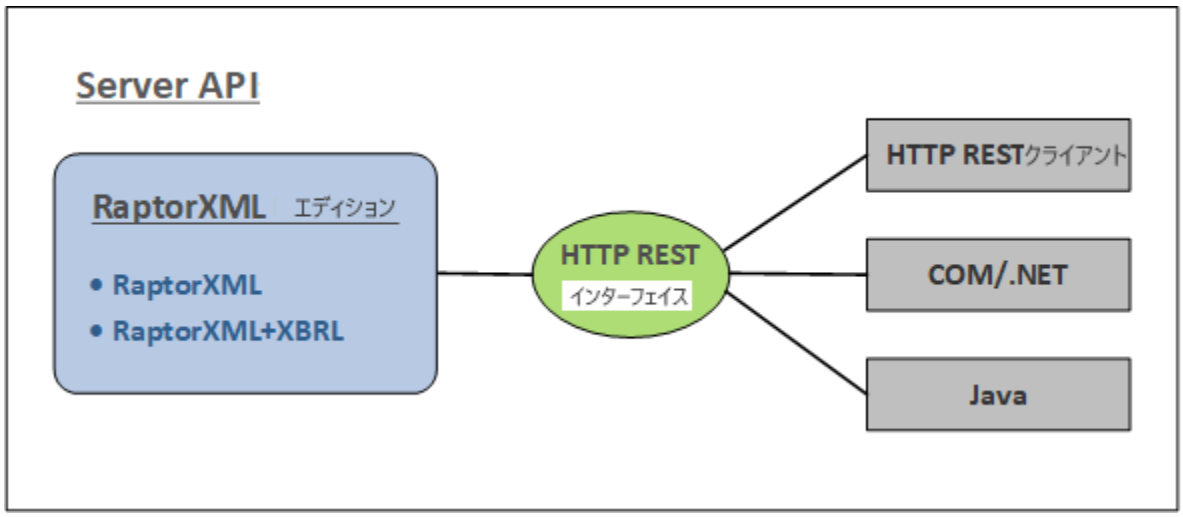
サンプル: **`--transforms=xml-c14n11`**

write-default-attributes

--write-default-attributes = `true|false`
署名済みドキュメント内の DTD からのデフォルトの値を含むかを指定します。

5 サーバー API: HTTP、COM/.NET、Java

RaptorXML Server はクライエントによりサーバーに検証ジョブを送信するHTTP REST インターフェイスを定義します。クライアントは HTTP REST インターフェイスに直接アクセス、または、ハイレベルのCOM/.NET およびJava Server API を使用します。これらの API はCOM/.NET とJava クラスを使用するHTTP REST リクエストの作成と送信を管理する簡単な方法を提供します。下の図は RaptorXML サーバーと通信する使用することのできるHTTP REST クライアントメソッドを表しています。



3つのサーバーAPI が存在し、HTTP REST インターフェイスを介して、以下の図に示されるようにRaptorXML と通信することができます。

- HTTP REST クライアントインターフェイス 200
- COM/.NET API 230
- Java API 238

メモ API には、コマンドラインインターフェイス(CLI) 52 と同じ機能が搭載されています。これには、検証とドキュメントの変換が含まれます。データの読み取り、抽出、分析などの高度の機能を使用する場合、エンジン API を使用してください。これらのAPI は要素の個数、および、ドキュメント内の要素の位置などの追加情報を提供します。

使用方法

RaptorXML Server はクライアントによりろーけるネットワークでアクセスできるマシン上にインストールされている必要があります。RaptorXML Server サービスが開始されると、クライアントはサーバーに接続し、コマンドを生成することができます。次のアクセスメソッドはサーバーAPI としてラベル付けされ、リモートのRaptorXML サーバーと通信する方法を提供します。

API のこのグループのカテゴリーは、HTTP からアクセスができるため サーバーAPI デバイス名の下で、 RaptorXML Server がサーバー上にインストールされています。

- *HTTP REST クライアントインターフェイス* 200: JSON フォーマットでクライアントリクエストは行われます。各リクエストは、出力ファイルが保存されるサーバー上のジョブディレクトリに割り当てられます。クライアントへのサーバーレスポンスは、ジョブに関する関連した情報が含まれています。詳細に関しては、次を参照してください: HTTP REST クライアントインターフェイス 200.
- *COM/.NET API* 230 と*Java API* 238: COM/.NET プログラム言語 230 内のスクリプトとJava 238 内のスクリプトは、RaptorXML サーバーAPI 241 のオブジェクトを使用して、RaptorXML Server の機能にアクセスすることができます。詳細に関しては、対応するサブセクションを参照してください。

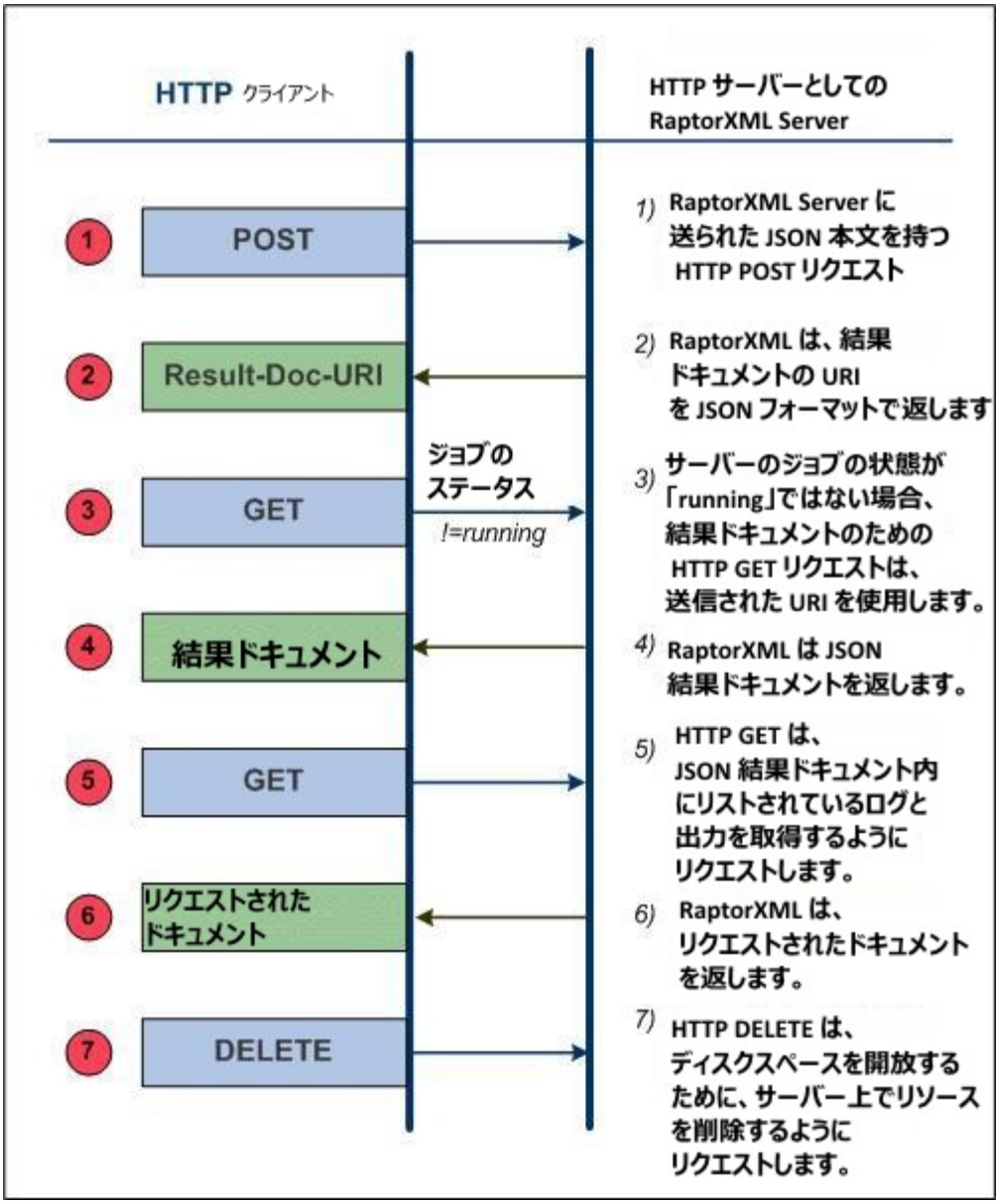
 (C) 2015-2021 Altova GmbH

ライセンス

RaptorXML Server はインストールされているマシン上でライセンス供与されています。RaptorXML Server のへ接続はHTTP を介して行われます。

5.1 HTTP REST クライアントインターフェイス

RaptorXML Server は、HTTP（または HTTPS [208]）を介して送信された検証ジョブを受け付けます。ジョブの説明と結果は、JSON フォーマットで交換されます。基本的なワークフローは下の図で表示されています。



HTTP REST インターフェイスに関連するセキュリティの問題

HTTP REST インターフェイスは、デフォルトでは、クライアントにより指定された HTTP プロトコール

を使用してアクセスすることのできる)すべての箇所に結果ドキュメントを書き込むことができます。ですから、RaptorXML Server を構成する際、このセキュリティのアスペクトを考慮することは重要です。

セキュリティが危害を受けるまたはインターフェイスが誤用される問題がある場合、結果ドキュメントが、サーバー上の専用の出力ディレクトリに書き込まれるようにサーバーを構成することができます。これには、サーバー構成ファイルの server.unrestricted-filesystem-access [205] のオプションの設定を false に指定します。この様にアクセスが制限されていると、クライアントは結果ドキュメントを専用の出力ディレクトリから GET リクエストを使用してダウンロードすることができます。または、管理者が結果ドキュメントファイルをサーバーからターゲットの場所にコピー/ダウンロード することができます。

このセクション

クライアントリクエストを送信する前に、RaptorXML Server が正確に構成され、開始されている必要があります。設定方法はサーバーセットアップ [201] のセクションに説明されています。クライアントリクエストの送信方法はクライアントリクエスト [211] のセクションで説明されています。

5.1.1 サーバーセットアップ

RaptorXML はインストールされているマシン上でライセンス供与されている必要があります。このインストールは HTTP REST インターフェイス [200] からアクセスすることができます。 RaptorXML Server を正しくセットアップするには、以下を行います。RaptorXML Server が既に正確にインストール [19] および ライセンス供与 [19] されていると仮定します。

1. RaptorXML Server は、HTTP または HTTPS を介して正確にアクセスされるためには、サービスまたはアプリケーションとして開始 [201] される必要があります。この方法は、オペレーションシステムにより異なり以下で説明されています: Windows [202]、Linux [202]、macOS [203]。
2. サーバーへの接続をテスト [203] するために初期サーバー構成 [204] を使用します。(初期サーバー構成 [204] はインストール時のデフォルトの構成です。) http://localhost:8087/v1/version などのシンプル HTTP GET リクエストを使用して接続をテストすることもできます。(リクエストはブラウザーウィンドウのアドレスバーに入力することもできます。) サービスが作動してる場合、バージョンリクエストなどの HTTP テストリクエストの反応を取得する必要があります。
3. サーバー構成ファイル [203] server_config.xml を確認してください。ファイル内の設定 [205] を変更する場合は、サーバー構成ファイルを編集して変更を保存します。HTTPS は、デフォルトで無効化されているため、構成ファイル [203] 内で有効化される必要があります。
4. サーバー構成ファイル [203] を編集するには、RaptorXML Server をサービスとして再起動して、新しい構成設定が適用されるようにします。ます。接続を再度確認して、サービスが作動しており、アクセスすることができることを確認してください。

メモ　サーバー開始エラー、使用される サーバー構成ファイル、およびライセンスエラーはシステムログに報告されます。サーバーに関する問題がある場合は、システムログ [205] を参照してください。

HTTPS に関する詳細は、HTTPS 設定 [208] のセクションを参照してください、

5.1.1.1 サーバーの開始

このセクション:

- サーバー実行可能ファイルの場所 [202]
- Windows 上でサービスとして RaptorXML を開始する [202]

- Linux 上でサービスとして RaptorXML を開始する 202
- macOS 上でサービスとして RaptorXML を開始する 203

サーバー実行可能ファイルの場所

RaptorXML Server 実行可能ファイルはデフォルトではフォルダにインストールされます:

```
<ProgramFilesFolder>\Altova\RaptorXMLServer2021\bin\RaptorXML.exe
```

RaptorXML Server をサービスとして開始するために、実行ファイルを使用することができます。

Windows 上でサービスとして開始する

インストールのプロセスは、RaptorXML Server をサービスとして Windows 上に登録します。ですが、RaptorXML Server をサービスとして開始する必要があります。これを以下の方法で行うことができます:

- システムトレイ内のアイコンとして使用することのできる、Altova ServiceController を介して開始。アイコンが使用不可能の場合、Altova ServiceController を開始して、**スタート** メニューから、[**すべてのプログラム** | Altova | Altova **LicenseServer | Altova ServiceController**] を選択して、アイコンをシステムトレイに追加します。
- Windows サービス管理コンソールを使用して開始: [**コントロール パネル | すべての コントロール パネル項目 | 管理ツール | サービス**]。
- 管理者の権利と共に開始されるコマンドプロンプトを使用して開始。任意のディレクトリで次のコマンドを使用します: **`net start "Altova RaptorXML Server"`**
- コマンドプロンプトウィンドウ内の RaptorXML Server 実行可能ファイルを使用して開始: **`RaptorXMLServer.exe debug`**。これは、サーバーを開始し、コマンドプロンプトウィンドウに直接サーバーアクティビティに関する情報が表示されます。サーバーアクティビティに関する情報は、サーバー構成ファイル 203 の `http.log-screen` 205 設定によりオンオフすることができます。サーバーを中断するには、[**Ctrl+Break**](または[**Ctrl+Pause**])を押します。サーバーが上記のようにサービスとしてではなく、この方法で開始されると、コマンドラインコンソールが中断される、またはユーザーがログオフするとサーバーは中断されます。

Linux 上でサービスとして開始する

以下のコマンドを使用して RaptorXML Server をサービスとして開始します:

[< 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`

RaptorXML Server を停止する場合は、以下を使用します:

[< 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

macOS 上でサービスとして開始する

以下のコマンドを使用して RaptorXML Server をサービスとして開始します:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.RaptorXMLServer2021.plist
```

RaptorXML Server を停止する場合は、以下を使用します:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.RaptorXMLServer2021.plist
```

5.1.1.2 接続のテスト

このセクション:

- 接続をテストするためのリクエストの取得 204
- サーバーの反応と JSON データ構造リスト 204

接続をテストするためのリクエストの取得

RaptorXML Server が開始されると、GET リクエストを使用して接続のテストを行います。(ブラウザーウィンドウのアドレスバーにこのリクエストを入力することもできます)。

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

メモ　RaptorXML Server のインターフェイスとポート番号は、次のセクション サーバー構成 203 で説明されている、サーバー構成ファイル server_config.xml 内で指定されています。

サーバーの反応と JSON データ構造リスト

サーバーが作動しており、正確に構成されている場合、リクエストが失敗することはありません。RaptorXML Server は、バージョンの情報を JSON データ構造として返します。(*下のリスト参照*)。

```
{
    "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"
}
```

メモ　サーバー構成ファイル 203 を編集して、サーバー構成を変更する場合、接続を再度テストしてください。

5.1.1.3 サーバーの構成

このセクション:

- サーバー構成ファイル: 初期設定 204
- サーバー構成ファイル: 初期設定の変更、初期設定に戻す 204
- サーバー構成ファイル: リストと設定 204
- サーバー構成ファイル: 設定の説明 205
- サーバーアドレスの構成 207

サーバー構成ファイル: 初期設定

RaptorXML Server は、 `server_config.xml` と呼ばれる構成ファイルを使用して構成されています。このファイルはデフォルトで以下の場所にあります:

```
C:\Program Files (x86)\Altova\RaptorXMLServer2021\etc\server_config.xml
```

RaptorXML Server のための初期構成は、以下を定義します:

- ポート番号 8087 がサーバーのポート番号です。
- サーバーがローカル接続 (`localhost`) のみをリッスンします。
- サーバーが出力を以下に書き込む `C:\ProgramData\Altova\RaptorXMLServer2021¥Output¥`。

他のデフォルトの設定は `server_config.xml` の内のリスト 204 で表示されています。

サーバー構成ファイル: 初期設定の変更、初期設定に戻す

初期設定を変更するには、 サーバー構成ファイル `server_config.xml` (下のリスト参照 204) を変更して、 RaptorXML Server をサーバーとして再起動します。

元のサーバー構成ファイルを再作成するには、(元の設定でサーバーが構成されるように)、`createconfig` コマンドを実行してください:

```
RaptorXMLServer.exe createconfig
```

このコマンドを実行するには、初期設定ファイルが再作成され、 `server_config.xml` ファイルを上書きします。`createconfig` コマンドは、サーバー構成を初期設定にリセットする際に役に立ちます。

サーバー構成ファイル: リストと設定

サーバー構成ファイル、**`server_config.xml`**、は初期設定と共に下にリストされています。使用可能な設定は下のリストで説明されています。

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\RaptorXMLServer2021\output\</server.output-root-dir>
  <server.script-root-dir>C:\Program
Files\Altova\RaptorXMLServer2021\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\RaptorXMLServer2021\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\RaptorXMLServer2021\Log\access.log</http.access-file>
    <http.error-file>C:\ProgramData\Altova\RaptorXMLServer2021\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\RaptorXMLServer2021\etc\cert\key.pem</https.private-key>
  <https.certificate>C:\Program
Files\Altova\RaptorXMLServer2021\etc\cert\cert.pem</https.certificate>
  <!--<https.certificate-chain>/path/to/chain.pem</https.certificate-chain>-->

</config>
```

設定

language
任意の `language` 要素内の、サーバーメッセージの言語を設定します。デフォルト値は `en`（英語）です。許可される値は `en|de|es|fr|ja`（英語、ドイツ語、スペイン語、フランス語、および日本語）です。RaptorXML のローカライズの方法に関してはローカライズコマンド 164 を参照してください。

server.unrestricted-filesystem-access

- `true`（デフォルトの値）に設定された場合、出力ファイルは、ユーザーおよび Python スクリプトにより指定された場所に直接書き込まれます（同じ名前の既存のファイルを上書きする可能性があります）。しかしながら、HTTP を使用してリモートのマシンからファイルにアクセスするためにローカルのファイルパスを使用することはできません。ですから、RaptorXML Server が利モードのマシンで作動している場合、このオプションの値を **`false`** に設定してください。クライアントサーバーが同じマシン上にあり、そのマシンのディレクトリ上に出力ファイルを書き込む場合のみ、値を **`true`** に設定してください。
- `false` に設定された場合、ファイルは、出力ディレクトリ 205 内のジョブのディレクトリに直接書き込まれます。これらのファイルの URI は、結果ドキュメント 224 に含まれます。値を `false` に設定すると、サーバー上の専用および既知のジョブディレクトリ内のディスクのみに書き込まれるため、セキュリティのレイヤーが与えられます。ジョブ出力ファイルは後に、信用される方法で他の場所にコピーされます。

server.output-root-dir
全ての提出されたジョブの出力が保存されているディレクトリ。

server.script-root-dir
Python スクリプト 306 が保存されるディレクトリ。`script` オプションは、HTTP インターフェイスを介して、信頼されるディレクトリからのスクリプトが使用されると作動します。Python スクリプトをこれ以外のディレクトリから指定するとエラーが生じます。*Python スクリプトを安全にする 306* を参照してください。

server.default-script-api-version
Python スクリプトを実行するための、デフォルトのPython API バージョン。デフォルトでは、最新バージョンのPython API が使用されます。現在サポートされるバージョンは、1 と2 です。

server.catalog-file
使用されるXML カタログ ファイルのURL。デフォルトでは、フォルダー `<ProgramFilesFolder>\Altova\RaptorXMLServer2021\etc` にある、カタログ ファイル `Rootcatalog.xml` が使用されます。`server.catalog-file` 設定は、デフォルトのカタログファイルを変更する時のみ使用してください。

server.log-file
サーバーログファイルの名前と場所。*サーバーの開始/中止* などのサーバー上のイベントが、システムのイベントログで継続的にログされ Windows イベント ビューアなどのシステムのイベントビューアーに表示されます。ビューアーの表示に加え、ログメッセージは `server.log-file` オプションにより指定されたファイルに書き込まれることができます。サーバーログファイルは、サーバー開始エラー、使用された構成ファイル、およびライセンスエラーなどの、サーバー上の全てのアクティビティに関する情報を含みます。

http.enable
ブール値によりHTTP: `true | false` の有効化、または、無効化を行います。HTTPS に関係なく、HTTP を有効化/無効化することができます。また、両方を 同時に有効化することもできます。

http.environment
raptorxml の内部環境: `production | development`。開発環境は、生産環境の使用時よりもデバッグを容易にし、開発者のニーズが対象にされています。

http.socket-host
RaptorXML Server がアクセスされるインターフェイス。リモートマシンからのRaptorXML Server へのアクセスを受け入れる場合は、要素のコメントを解除してコンテントを以下に設定します: `0.0.0.0`。例: `<http.socket-host>0.0.0.0</http.socket-host>`。これはサービスをアドレス指定することのできるすべてインターフェイスでホストします。この場合、ファイアウォールの設定が適切にされていることを確認してください。Altova 製品の受信ファイアウォールが以下のように登録されている必要があります: Altova LicenseServer: ポート `8088`; Altova RaptorXML Server: ポート `8087`; Altova FlowForce Server: ポート `8082`。

http.socket-port
サービスがアクセスされるポートです。HTTP リクエストが正確にサーバーにアドレス指定されるために、ポートは固定されており、既知である必要があります。

http.log-screen
RaptorXML Server がコマンド `RaptorXMLServer.exe debug` で開始された場合、(サーバーの開始 201 を参照) そして、`http.log-screen` が `true` に設定されている場合、サーバーアクティビティはコマンドラインコンソールに表示されます。それ以外の場合、サーバーアクティビティは表示されません。ログファイルの書き込みに加え、ログスクリーンも表示されています。

http.access-file
HTTP アクセスファイルの名前と場所。アクセスファイルはアクセスに関連したアクティビティの情報を含んでいます。接続に関する問題を解決するために役に立つ情報を含んでいます。

http.error-file

HTTP エラーファイルの名前と場所。エラーファイルは、サーバーへのおよびからのトラフィックに関連したエラーを含みます。このファイルは、接続に関する問題を解決するために役に立つ情報を含んでいます。

http.max_request_body_size
このオプションは、RaptorXML Server が受け入れることのできるリクエストボディの最大のサイズを指定します。デフォルトの値は、100MB です。リクエストボディのサイズがこのオプションで指定された値より大きい場合、サーバーは「HTTP Error 413: 要求するエンティティが大きすぎます」を表示します。オプションの値は、ゼロまたはゼロより大きくなくてはなりません。制限は、`http.max_request_body_size=0` により無効化することができます。

https.enable
HTTPS: `true | false` を有効化無効化するためのブールの値。HTTP に関係なく、HTTPS を有効化/無効化することができます。また、両方を 同時に有効化することもできます。HTTPS へのサポートはデフォルトでは、無効化されており、この設定の値を **true** に変更することで有効化することができます。

https.socket-host
HTTP 接続が受け入れられるホストアドレスである文字列の値を取ります。ローカルホストからの接続のみを受け入れるためには、**localhost** または**127.0.0.1**. に設定します。RaptorXML Server が遠隔のマシンすべてからの接続を受け入れる場合は、値を `0.0.0.0` に設定します。例: `<http.socket-host>0.0.0.0</http.socket-host>`。これは、サーバーマシンのアドレスを与えることのできるインターフェイス上のサービスをホストすることができます。この場合、ファイアーウォールの設定が適切に構成されていることを確認してください。Altova 製品のための受信ファイアーウォールの例外は以下のように登録します: Altova LicenseServer: ポート `8088`; Altova RaptorXML Server: ポート `8087`; Altova FlowForce Server: ポート `8082`。以下のような IPv6 アドレスを使用することができます:'::'。

https.socket-port
HTTPS が受け入れられるポートである整数の値です。 HTTP リクエストが正確にサーバーへアドレスされるために、ポートは固定され、既知である必要があります。

https.private-key, https.certificate
サーバーへの秘密キーと証明書ファイルであるパスである URI です。両方が必須であり、詳細に関しては、HTTPS 設定[208] とSSL 暗号化のセットアップ[209] を参照してください。Windows パスを使用することもできます。

https.certificate-chain
中間証明書ファイルをロケートするための URI のための任意の設定です。2 つの中間証明書(プライマリとセカンダリ)が存在する場合、9 月7 日にSSL 暗号化のセットアップ[209] で説明されているとおり、これらの証明書を一つのファイルに結合します。詳細に関しては、HTTPS 設定[208] とSSL 暗号化のセットアップ[209] を参照してください。

RaptorXML Server アドレス

サーバーの HTTP アドレスは、ソケットホストとソケットポートにより構成されます:

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

初期構成を使用したセットアップのアドレスは以下のとおりです:

```
http://localhost:8087/
```

アドレスを変更するには、サーバー構成ファイル `server_config.xml` 内の`http.socket-host` および `http.socket-port`設定を変更します。例えば、マシンが次の IP アドレスを持つ場合、`123.12.123.1`、そして次のサーバー構成が設定された場合:

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

RaptorXML Server は、以下を使用してアドレス指定されることができます:

```
http://123.12.123.1:8087/
```

メモ `server_config.xml` が変更された後、RaptorXML Server は、新しい値が適用されるために再起動される必要があります。

メモ RaptorXML Server に接続する際に問題がある場合は、`http.access-file` および `http.error-file` 内で名前の付けられているファイルの情報が問題の解決の助けになります。

メモ RaptorXML Server へ提出されたメッセージは、サーバーマシン上で有効なパス名を含んでいる必要があります。サーバーマシン上のドキュメントはローカルまたはリモートからアクセスすることができます（後者の場合は、例えば、HTTP URI を使用した場合）。

5.1.1.4 HTTPS 設定

RaptorXML Server は、スタートアップを HTTP サーバーとしてだけではなく、HTTPS サーバーとしてもサポートします。

HTTPS の有効化

HTTPS へのサポートは、デフォルトでは無効化されています。HTTPS を有効化するには、サーバー構成ファイル203 `server_config.xml` 内の **`https.enable`** 設定を **`true`** に変更します。構成ファイル203 の多種の HTTPS 設定をサーバーの必要条件に応じて変更します。

秘密キーと証明書

秘密キーと証明書ファイルを次の方法で取得することができます:

- 証明機関から SSL 暗号化のセットアップ209 .セクション内の説明に従います。
- （使用中の環境の必要に応じて）次の OpenSSL コマンド を使用して自身で署名する証明書を作成します:

```
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"
```

接続のテスト

URL を使用したデータの転送のために curl コマンドラインツールを使用して接続をテストすることができます。以下のコマンドを使用することができます:

```
curl.exe https://localhost:443/v1/version
```

証明書が信頼されない場合、**`-k`** オプションを以下のように使用します:

```
curl.exe -k https://localhost:443/v1/version
```

次の HTTP Python サンプルの実行するコマンドは、RaptorXML Server に搭載されています:

```
python3.exe examples\ServerAPI\python\RunRaptorXML.py --host localhost -p 443 -s
```

5.1.1.5 SSL 暗号化のセットアップ

RaptorXML Server データ転送を SSL プロトコールを用いて暗号化するには、以下を行います:

- SSL 秘密キーを生成して、SSL 公開キー証明書ファイルを作成します。
- SSL 通信のために RaptorXML Server をセットアップします。

これを行うためのステップは以下のとおりです。

このメソッドは、SSL 暗号化を管理するために、オープンソース OpenSSL ツールキットを使用します。 ですから、OpenSSL が使用できるコンピューターを使用する必要があります。 OpenSSL は、通常の多くの Linux および macOS に既にインストールされています。また、Windows コンピューターにインストールすることもできます。インストーラーバイナリへのリンクをダウンロードするには、OpenSSL Wiki を参照してください。

秘密キーを生成して、証明書機関から証明書を取得するには、以下をおこないます:

1. 秘密キーの生成

   SSL は、RaptorXML Server にインストールされた秘密キーを必要とします。RaptorXML Server データを暗号化するためにこの秘密キーを使用します。次の OpenSSL コマンドを使用して、秘密キーを作成します:

   ```
   openssl genrsa -out private.key 2048
   ```

   これは秘密キーを含む `private.key` と呼ばれるファイルを作成します。ファイルの保存先を保存してください。秘密キーは以下をおこなうために必要です: (i)証明書の署名要求 Certificate Signing Request (CSR) を生成するため(ii) RaptorXML Server にインストールするため。

2. 証明書の署名要求 (CSR)

   証明書の署名要求 Certificate Signing Request (CSR) は、公開キー証明書をリクエストするために VeriSign または Thawte などの証明書機関 (CA) に送信されます。CSR は、秘密キーをベースにしており、所属機関の情報を含んでいます。CSR を次の OpenSSL コマンドを使用して作成します (パラメーターの 1 つとしてステップ 1 で作成された秘密キーファイル `private.key` を提供します):

   ```
   openssl req -new -nodes -key private.key -out my.csr
   ```

   CSR の生成中、以下にリストされるような、所属機関の情報を提供する必要があります。この情報は証明書機関が所属機関の ID を検証するために使用されます。

   - *国名*
   - *住所* (所属機関が存在する場所)
   - *所属機関* (会社名)。特殊文字を使用しないでください。これらの文字は証明書を無効化する可能性があります。
   - *共通名* (サーバーの DNS 名)。サーバーに接続するために使用される DNS 名前クライアントアプリであるサーバーの公式名に一致する必要があります。
   - *チャレンジパスワード*。この項目は空白にしてください!

3. BSSL 証明書の購入

SSL 証明書を VeriSign または Thawte などの公式の証明書機関（CA）から購入します。これらの命令に関しては、VeriSign の手続きに従います。他の CA の手続きも同様に行います。

- VeriSign Web サイトに移動します。
- **証明書の購入**をクリックします。
- 異なる種類の SSL 証明書も使用することができます。RaptorXML Server に関しては、安全なサイト、または、安全なサイトプロでは、証明書で十分です。EV（拡張された検証）は、ユーザーが確認することのできる安全なサイトのアドレスバーが存在するため必要ありません。
- サインアッププロセスの手順を踏みで、必要な情報を記入します。
- (*ステップ 2 で作成された*) CSR に関してプロンプトされると、`my.csr` ファイルを注文フォームに内容をコピーして貼り付けます。
- 証明書をクレジットカードを使用して購入します。

> **証明書の取得間での時間**
> SSL 証明書機関（CA）から公開キー証明書を取得するには、**2-3 営業日** がかかります。RaptorXML Server をセットアップするには、この点を考慮してください。

4. CA から公開キーを取得する

   証明書機関は、2－3営業日以内に登録プロセスを完了します。この間に、電子メールまたは電話で、DNS ドメインのための SSL 証明書のリクエストの認証に関する連絡がある可能性があります。このプロセスを完了するために、関連機関と協力してください。

   承認と登録のプロセスが完了すると、SSL 証明書の **公開キー**を含む電子メールが送信されます。認証と登録が完了すると、証明書を含む電子メールが送信されます。公開キーは、テキストフォーム、または、`.cer ファイル`として添付され送信されます。

5. 公開キーをファイルに保存する

   RaptorXML Server と使用する場合、`.cer` ファイル内に公開キーが保存される必要があります。公開キーがテキストとして提供される場合、以下からラインをコピーして貼り付けます。

```
--BEGIN CERTIFICATE--
  ...
--END CERTIFICATE--
```

   `mycertificate.cer` を呼び出すテキストファイル

6. CA の中間証明書をファイルに保存する

   SSL 証明書を完了するには、以下の2つの追加証明書が必要になります: **プライマリ** と **セカンダリ 中間証明書**。証明書機関（CA）の Web サイトは、中間証明書の内容をリストしています。

   - Verisign の中間証明書: https://knowledge.verisign.com/support/ssl-certificates-support/index?page=content&id=AR657&actp=LIST&viewlocale=en_US
   - Verisign の安全なサイトの製品のための中間証明書: https://knowledge.verisign.com/support/ssl-certificates-support/index?page=content&id=AR1735

中間証明書（プライマリとセカンダリ）をそれぞれ個別にテキストファイルにコピー、貼り付け、使用中のコンピューターに保存します。

7. 証明書 を1 つの公開キー証明書ファイルにまとめる

3つの証明書ファイルが存在します:

- 公開キー(`mycertificate.cer`)
- セカンダリ中間証明書
- プライマリ中間証明書

公開キー証明書に中間証明書を統合することができます。統合する方法は下に説明されています。（または、**`https.certificate-chain`** 構成ファイル設定[205] を使用して、中間証明書の場所を指定します）。

それぞれは、以下のような括弧で囲まれたテキストのブロックを含んできます:

```
--BEGIN CERTIFICATE--
  ...
--END CERTIFICATE--
```

これら3つの証明書を1つのファイルに順番にコピーして、貼り付けます。シーケンスの順番は重要です: (i) 公開キー、(ii) セカンダリ中間証明書、(iii) プライマリ中間証明書。証明書と証明書の間に行は存在しません。

```
--BEGIN CERTIFICATE--
  PUBLIC KEY from mycertificate.cer (ステップ5を参照)
--END CERTIFICATE--
--BEGIN CERTIFICATE--
  セカンダリ中間証明書 (ステップ6を参照)
--END CERTIFICATE--
--BEGIN CERTIFICATE--
  プライマリ中間証明書 (ステップ6を参照)
--END CERTIFICATE--
```

`publickey.cer` という名前のファイルに証明書テキストと結合された結果を保存します。これが、SSL 証明書の *公開キー証明書ファイル* です。これには、公開キー証明書と、CA により証明書に署名するために使用された中間証明書のフォームの信頼のチェーンが含まれて居ます。

5.1.2 クライアントリクエスト

RaptorXML Server が サービスとして開始[201] されると、すべてのHTTP クライアント から、以下を行うことのできる機能をアクセスすることができます:

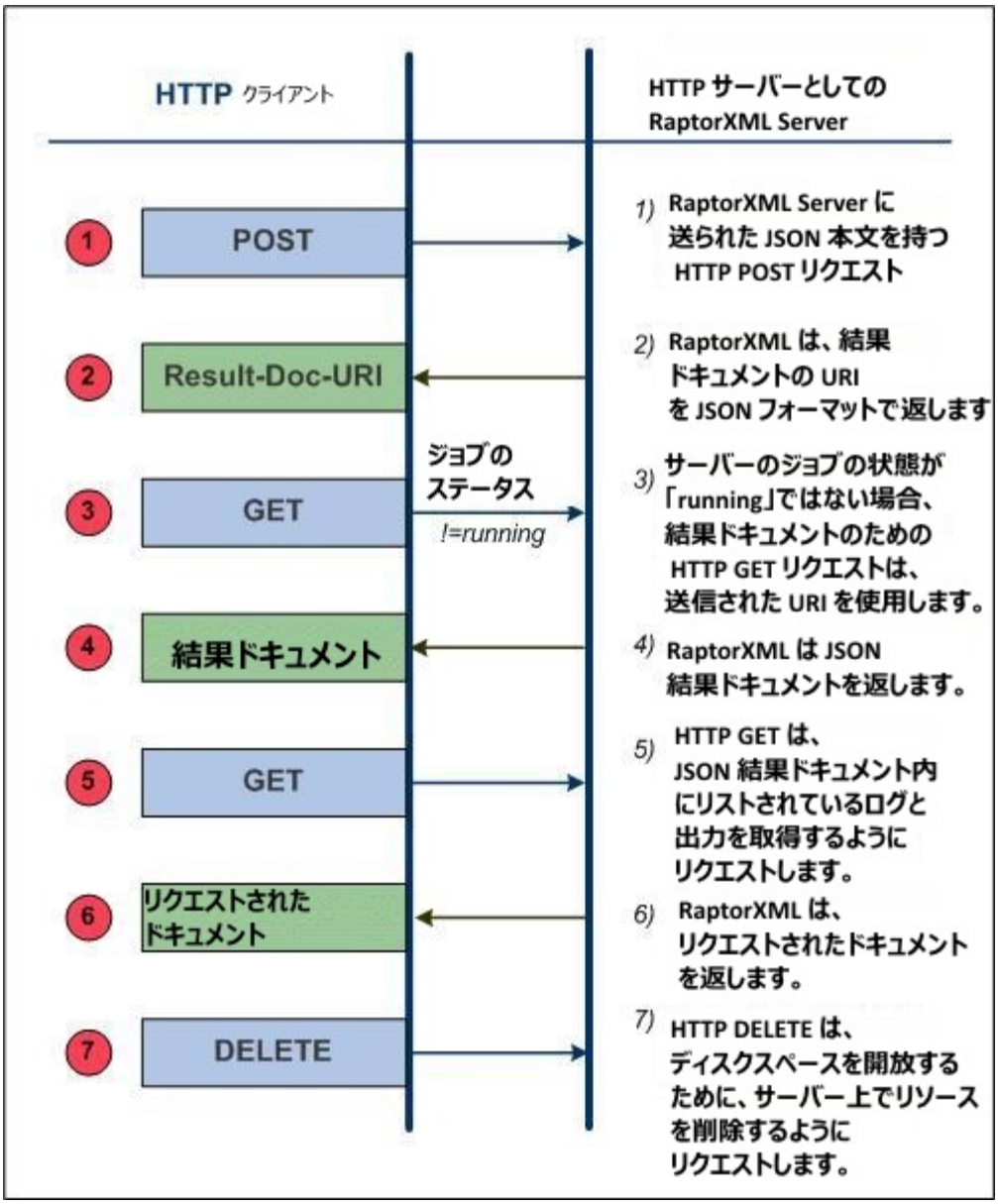
- HTTP メソッド `GET`、`PUT`、`POST`、および `DELETE` を使用します。
- `Content-Type` ヘッダーフィールドの設定

使用しやすいHTTP クライアント

インターネットからダウンロードすることのできる多数のW eb クライアントがあります。使用しやすく安定した、W eb クライアントは、Firefox のプラグインとして追加することのできるMozilla のRESTClient です。インストールしやすく、RaptorXMLが必要とする

HTTP メソッド をサポートし、十分な JSON 構文の色分けを提供します。HTTP クライアントの経験が無い場合、RESTClient を試してください。 RESTClient のインストールと使用はユーザー自身の責任において続行してください。

典型的なクライアントリクエストは、下の図に表示されるように一連のステップから構成されています。



各ステップの重要な点は下に説明されています。主要な用語は太字で表記されています。

1. リクエストの本文がJSON フォーマットである HTTP `POST` メソッド はリクエストの作成 213 に使用されます。リクエスト は RaptorXML Server の全ての機能に対して使用できます。例えば、リクエストは検証、XSLT 変換 等に使用することができます。リクエスト内で使用される、コマンド、引数、およびオプション はコマンドライン 52 内で使用されるものと同様です。リクエストは

以下にポストされます: `http://localhost:8087/v1/queue`。`localhost:8087` がRaptorXML Server（サーバーの初期設定アドレス[207]）のアドレスと仮定されます。リクエストは、**RaptorXML Server ジョブ** と称されます。

2. RaptorXML Server によりリクエストが受信され、処理が受け付けられると、ジョブが処理された後、サーバーアクションの結果を含む**結果ドキュメント** が作成されます。 **結果ドキュメントの URI** は、（上の図の Result-Doc-URI ）クライアントに帰されます。[224] 処理が完了していなくても、ジョブが処理のために受け付けられると（キューに並べられると）URI が即時に返されることに注意してください。

3. （URI 使用して結果ドキュメントを使用する場合、）クライアントはサーバーへの `GET` メソッド内で、結果ドキュメントのためにリクエストを送信します。[224] リクエストの受信時、ジョブの処理が開始されていない、または完了していない場合、サーバーは実行の状況を返します。`GET` リクエストは、ジョブの処理が完了し、結果ドキュメントが作成されるまで、繰り返されなければなりません。

4. RaptorXML Server は、結果ドキュメントを JSON フォーマットで返します。[225] 結果ドキュメントは、元のリクエストの RaptorXML Server 処理により、作成された**エラーの URI** または、**出力ドキュメント** を含む可能性があります。エラーログが返されます。例えば、検証がエラーを返す場合があります。XSLT 変換の結果などのプライマリ出力ドキュメントは、出力作成ジョブに成功した場合返されます。
5. クライアントは、サーバーへの HTTP `GET` メソッド を介して、ステップ 4 で受信した、出力ドキュメントの URI を送信します。[227] 各リクエストは個別の `GET` メソッドで送信されます。

6. RaptorXML Server は、ステップ 5 で作成された `GET` リクエストに応答して、リクエストされたドキュメントを返します。[227]

7. クライアント は、ジョブリクエストの結果として生成された、サーバー上の必要のないドキュメントを削除することができます。[229] これは、結果ドキュメントの URI 内の HTTP `DELETE` メソッドを送信することにより行えます。これは、一時的なファイルおよびエラー、出力ファイルなどを含みます。このステップは、ハードディスクのスペースを解放するために役に立ちます。

各ステップの詳細に関してはこのセクションのサブセクションで説明されています。

5.1.2.1 POST を使用してジョブを開始する

このセクション:

- リクエストの送信[213]
- POST リクエストのための JSON 構文[213]
- POST リクエストを使用してファイルを更新する[215]
- ZIP アーカイブのアップロード[216]

リクエストの送信

HTTP `POST` メソッドにより RaptorXML Server ジョブが開始されます。

HTTP メソッド	URI	コンテンツ型	ボディ
`POST`	`http://localhost:8087/v1/queue/`	`application/json`	JSON

以下の点に注意してください：

- 上の URI には、初期構成[204] の設定を使用するサーバーアドレスがあります。
- URI には、URI 内に存在しなくてはならない、`/v1/queue/` パスがあります。ジョブが置かれるメモリ内のあいまいなフォルダーとして考えられます。

- 正確なバージョン番号 /vN は、サーバーが返す番号です。(このドキュメント内に記載されている番号ではない可能性があります)。サーバーが返す番号は、現在のHTTP インターフェイスのバージョン番号。前のバージョン番号は、下位互換性により、サポートされているHTTP インターフェイスの古いバージョンを示します。
- ヘッダーは以下のフィールドを含む必要があります: Content-Type: application/json。しかし、POST リクエストの本文内でファイルをアップロードする場合は、メッセージヘッダーのコンテンツ型が multipart/form-data (例、Content-Type: multipart/form-data) に設定されている必要があります。 POST リクエストと共にファイルをアップロード 215 のセクションを参照してください。
- リクエストの本文は、JSON フォーマットである必要があります。
- 処理されるファイルはサーバー上にある必要があります。ですから、リクエストが作成される前に、ファイルはサーバー上にコピーされるか、POST リクエストと共にファイルがアップロード 215 されている必要があります。この場合、メッセージヘッダーは、コンテンツ型を multipart/form-data に設定する必要があります。詳細に関しては、POST リクエストと共にファイルをアップロード 215 のセクションを参照してください。

XML ファイルの整形式のチェックをするには、JSON フォーマット内のリクエストは以下に類似します:

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

有効なコマンドとその引数およびオプションは、コマンドライン セクション 52 で説明されているとおりです。

HTTP POST リクエストのための JSON 構文

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

- 黒いテキストはすべて含まれる必要があります。これは、すべてのかっこ、二重引用符、コロン、コンマおよび角かっこを含みます。空白文字は正規化されることができます。
- 青い斜体は、プレースホルダでコマンド名、オプション、オプションの値、および引数の値を意味します。コマンドについての説明は、コマンドライン セクション 52 を参照してください。
- command および args キーは必須です。options キーは任意です。options キーの一部は、デフォルトの値を持ちます。ですから、これらのオプションは、デフォルトの値が変更され指定される必要があります。
- すべての文字列は、二重引用符で囲まれる必要があります。ブール値および数値は引用符と必要としません。ですから: {"error-limit": "unlimited"} および {"error-limit": 1} が正しい使用方法です。
- ファイル パスより先、ファイルURI が奨励され、スラッシュを使用します。Windows ファイル パスは、使用されると、バックスラッシュを使用します。更に、Windows ファイル パスバックスラッシュは、JSON 内ではエスケープする必要があります。(バックスラッシュのエスケープと共に。ですから以下のようになります "c:\\dir\\filename")。 ファイルURI および ファイル パスは、文字列であり、かっこで囲まれる必要があることに注意してください。

以下がオプション付きのサンプルです。(input または xslt-version などの) オプションの一部には、オプションの値を取る場合がありますが、(param など) 他はキー値ペアなどを取るため、異なる構文を必要とします。

```
{
  "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"
          }
}
```

下のサンプルは、3 つめの型のオプションを表示しています。(下では、 xsd オプションで表示されるように) 値の配列の型です。この場合、使用される構文はJSON 配列です。

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

POST リクエストを使用してファイルを更新する

処理するファイルは、 POST リクエストの本文の中でアップロードすることができます。この場合、 POST リクエストは以下のように作成される必要があります。

リクエスト ヘッダー

リクエスト ヘッダー内で、 Content-Type を以下のように設定します: multipart/form-data そして、任意の文字列を境界として設定します。サンプルヘッダー:

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

この境界の目的は、リクエストの本文内で異なるフォームデータの部分に境界を設定するためです。(*以下を参照*).

リクエスト本文: メッセージ パート

リクエストの本文は、次のフォームデータの部分を持ち、リクエスト ヘッダー内で指定された境界文字列で区切られています(*上を参照*):

- *必須のフォームデータの部分*:リクエストされた処理アクションを指定する、 **msg** および、 msg フォームデータ パート内で指定されるコマンドの引数としてアップロードされるファイルを含む、 args。 *下のリストを参照してください*。
- *任意のフォームデータの部分*: msg または args フォームデータ パート内のファイルから参照されるファイルを含む additional_files。更に、コマンドのオプションから名前の付けられたフォームデータの部分は、アップロードされるファイルを含むことができます。

メモ 全てのアップロードされたファイルは、単一の仮想ディレクトリで作成されます。

コードに関する詳細はサンプル-1（コールアウト付き）: XML の検証 216 およびサンプル-2: カタログを使用したスキーマ検索 218 を参照してください。

> **CURL を使用してテスト**
>
> CURL (http://curl.haxx.se/) などのサードパーティデータ転送アプリケーションを使用して POST リクエストをテストすることができます。CURL はリクエストのパートバウンダリーを生成しリストする役に立つトレースオプションを提供します。これにより手動でのパートバウンダリーの生成の作業を省くことができます。 How you can use CURL の使用方法については Testing with CURL 220 のセクションで説明されています。.

ZIP アーカイブのアップロード

ZIP アーカイブもアップロードすることができます。ZIP 内のファイルは **additional-files** スキームを利用して参照することができます。例えば

```
additional-files:///mybigarchive.zip%7Czip/biginstance.xml
```

メモ **|zip/** パートでは、パイプ **|** シンボルが直接許可されていないため、URI RFC に対して準拠するために **%7Czip/** のような URI-エスケープ文字が必要です。glob パターン（***** および **?**）の使用は許可されています。ですから、以下に類似したものを ZIP アーカイブ内のすべての XML ファイルを検証するために使用することができます:

```
{"command": "xsi", "args": ["additional-files:///mybigarchive.zip%7Czip/*.xml"],
"options": {...}}
```

サンプルコードのリストはサンプル-3: ZIP アーカイブの使用 218 を参照してください。

5.1.2.1.1 サンプル-1（コールアウト付き）: XML の検証

下には POST リクエストの本文のリストが挙げられています。下で説明されている番号づけられたコールアウトがあります。リストリクエストに送信されたコマンドは、以下と同等の CLI を持ちます:

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

スキーマに従った、2 つの XML ファイルの検証のためのリクエスト.
リクエストの本文は、以下のようになります。---PartBoundary が境界文字列として、既にヘッダーで指定されていると仮定して、（上の*リクエストヘッダー* 215 *を参照してください*）。

```
-----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. メインフォームデータの部分の名前。境界はリクエストヘッダー 215 で宣言されています。パート境界セパレーターは、埋め込まれたドキュメント内で存在しない、一意の文字列である必要があります。2 本のダッシュの接頭辞があり、複数の部分を区切るために使用されます。 最初のフォームデータの部分（このサンプルでは） **msg**. 型は `application/json` であることに注意してください。
2. これは標準の HTTP POST リクエストの構文 214 です。`args` がファイルへの参照を含み、追加ファイルがアップロードされる場合、両方のセットのファイルはサーバーにパスされます。
3. `args` 配列の最初のメンバーは、`First.xml` と称されるファイル添付です。
4. ファイル `First.xml` のテキスト。これは、`additional_files` フォームデータの部分へアップロードされる `Demo.xsd` と呼ばれるスキーマへの参照を含みます。
5. `args` 配列の 2 番目のメンバーは、`Second.xml` と称される添付です。
6. ファイル `Second.xml` のテキスト。これもスキーマ `Demo.xsd` への参照を含みます。1コールアウト 7 を参照してください。
7. 最初の追加ファイルのパートは `Demo.xsd` 添付メタデータを含みます。
8. ファイル `Demo.xsd` のテキスト
9. `Demo.xsd` 追加ファイル部分の終わり、および `additional_files` フォームデータの部分。境界セパレーターの最後の部分は2 本のダッシュの接頭辞および接尾辞があります。

5.1.2.1.2 サンプル-2: カタログを使用したスキーマ検索

このサンプルでは、検証するXML ファイルにより参照されている、XML スキーマを検索するためにカタログファイルが使用されています。

```
-----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--
```

5.1.2.1.3 サンプル-3: ZIP アーカイブの使用

ZIP アーカイブもアップロードすることが可能です。ZIP 内のファイルは `additional-files` スキームを使用して参照することができます。例:

```
additional-files:///mybigarchive.zip%7Czip/biginstance.xml
```

メモ　`|zip/` パートでは、パイプ `|` シンボルが直接許可されていないため、URI RFC に対して準拠するために `%7Czip/` のような URI-エスケープ文字が必要です。glob パターン（`*` および `?`）の使用は許可されています。ですから、以下に類似したものを ZIP アーカイブ内のすべての XML ファイルを検証するために使用することができます:

```
{"command": "xsi", "args": ["additional-files:///mybigarchive.zip%7Czip/*.xml"],
"options": {…}}
```

サンプル: ZIP アーカイブ内の全ての XML ファイルを検証する

このサンプルでは、すべてのスキーマ参照は相対パスであり、すべてのスキーマは ZIP 内にあると想定されます。

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

{"command": "xsi", "args": ["additional-files:///Demo.zip%7Czip/*.xml"], "options": {}}

-----PartBoundary
Content-Disposition: attachment; filename="Demo.zip"; name="additional-files"
Content-Type: application/octet-stream

Binary content of Demo.zip archive

-----PartBoundary--
```

サンプル: 外部スキーマへの参照を含む ZIP アーカイブ内の XML ファイルの検証

このサンプルでは、第2 の ZIP アーカイブで与えられる、外部スキーマの参照を使用して、アーカイブ内の XML ファイルが検証されます。

```
-----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--
```

5.1.2.1.4 CURL を使用したテスト

第三者パーティアプリケーション CURL (http://curl.haxx.se/) は POST リクエストをテストするために使用することのできるコマンドライン ユーティリティです。CURL はリクエスト内で直接使用することのできる、または、参照として使用することのできるリクエストのパート境界を生成しリストするとても役に立つトレースオプションを提供します。

XML スキーマに対して XML ファイルが検証されるサンプルのテストシナリオは以下の通りです。以下が想定されます:

- 以下のコマンドは検証のために提出されたファイルが存在するフォルダーから実行されます (これによりこれらのファイルに対して簡単な相対的なパスを使用することができます)。Altova XMLSpy アプリケーションがインストールされている場合、このサンプル内で使用されているファイルは、デフォルトで C:\Users\<username>\Documents\Altova\XMLSpy2021\Examples に存在するアプリケーションの Examples フォルダー内にあります。
- RaptorXML Server はローカルで作動しているか、または、ポート 8087 で作動しています。

CURL コマンドラインオプションに関する詳細は、CURL ヘルプを参照してください。

検証コマンドを使用して Windows 上で CURL を呼び出す方法

```
[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
```

メモ powershell 内では、引用内で引用が存在する場合、異なる型の引用符 (一重引用符/二重引用符) が使用される必要があります。

```
[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"}
```

"result" の URL を使用して結果をフェッチする方法

```
[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 のトレースオプション

CURL にはサーバーから、および、サーバーへ送信された HTTP トラフィックをトレースしトレースオプション（`--trace-ascii`）があります。オプションは POST とのジョブを開始するために必要とされているパート境界をリストするため、とても役に立ちます。トレース内に情報を直接的、または、参照として使用してパート境界を作成することができます。下のリストは、上記のコマンドを実行して取得されたトレースを表示しています。

リスティングのトレース

```
== 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
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
```

メモ `'Content-Disposition: attachment'` に加え、`'Content-Disposition: form-data'` が有効なことが上記のリストで確認することができます。

整形式チェックコマンドを使用して Linux 上で CURL を呼び出す方法

```
/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
```

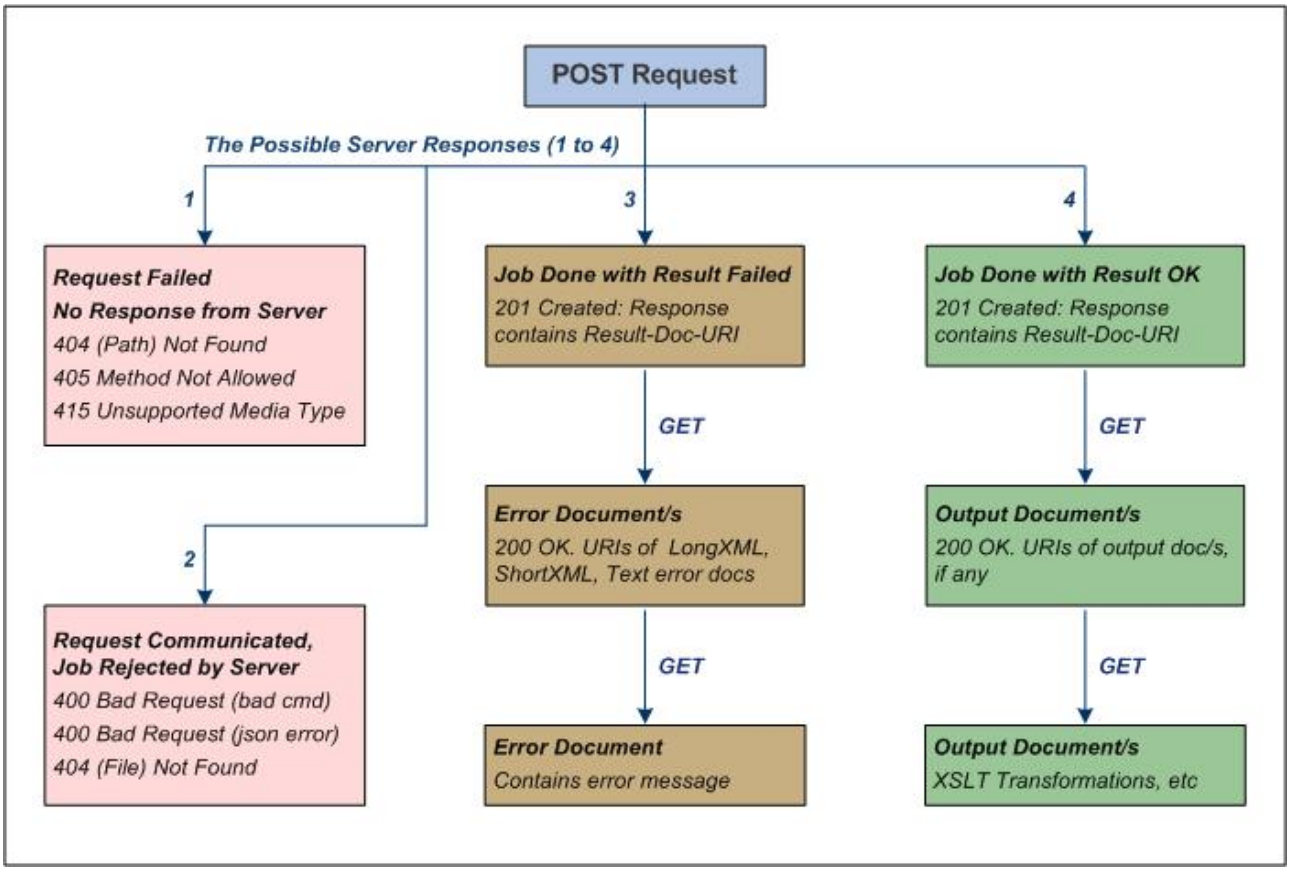
5.1.2.2 POST リクエストに対するサーバーの反応

このセクション:

- 考えられるサーバーの反応の概要 222
- 反応: リクエストは失敗し、サーバーからの反応がありません 223
- 反応: リクエストは通知されました、がサーバーによりジョブが拒否されました 223
- 反応: ジョブが実行されました（良いまたは悪い結果） 224

サーバーに対して `POST` リクエストの作成に成功すると、ジョブはサーバーキューに並べられます。 `201 作成されました` メッセージと結果ドキュメント URI が返されます。ジョブはできるだけ早く処理されます。その間、ジョブが完了しておらず、結果ドキュメントがリクエストされた 224 場合、`"status": "Running"` メッセージが返されます。クライアントは後で再度試行します。 `Dispatched` 状態は、ジョブがまだサーバーキューにあり、開始されていないことを示します。

ジョブの結果（例えば、検証リクエスト）がネガティブ（検証の失敗）またはポジティブ（検証の成功）であることができます。双方のケースで、`201 作成されました` メッセージが返され、結果ドキュメントが生成されます。`POST` リクエストがサーバーに通信されなかった場合（*リクエストの失敗*）、またはリクエストは通信済みでジョブがサーバーにより拒否された（*リクエストは通信されましたが、ジョブは拒否された*）ケースの可能性も存在します。多数の起こりうる結果が下の図に表示されています。



クライアントの `POST` リクエストの起りうる結果は以下の通りです:

リクエストは失敗し、サーバーからの反応がありません

サーバーへのリクエストに成功しない場合、多くの共通のエラーメッセージは以下のリストのとおりです:

メッセージ	説明
404 見つかりません	正確なパス: `http://localhost:8087/v1/queue/`
405 許可されていないメソッド	このリソースのための指定されたメソッドは無効です。`POST` メソッドを使用してください。
415 サポートしていないメディアの種類です	メッセージのヘッダーは以下である必要があります: `Content-Type:application/json`.

リクエストは通知されました、がサーバーによりジョブが拒否されました

サーバーへのリクエストの作成に成功した場合でも、サーバーは以下の理由でリクエストを拒否することがあります:

メッセージ	説明
400 無効な要求 (*無効な cmd*)	RaptorXML コマンド 52 は無効です。

400 無効なリクエスト(*json エラー*)	リクエスト本文にJSON 構文 214 エラーがあります。
404 ファイルが見つかりません	コマンドで名前が使用されている全てのファイルファイルURI (またはファイル パス) 構文 214 チェックしてください。

ジョブが実行されました(良いまたは悪い結果)

ジョブ(例えば、検証ジョブ) が実行されると、結果はポジティブ(*OK*) またはネガティブ(*失敗*) になります。例えば、検証ジョブの結果がポジティブ(*OK*) とは、検証されるドキュメントが有効な場合で、ネガティブ(*失敗*) とはドキュメントが無効な場合です。

両方の場合とも、ジョブは実行されますが、異なる結果を出します。201 - 作成されましたメッセージは、ジョブがキューに並べられるとすぐに送信されます。また、両方の場合とも、結果ドキュメント URI は、リクエストを作成した、HTTP クライアントに返されます。(結果ドキュメント自身は、ジョブが開始されていない、または完了していない場合は、まだ作成されない可能性があります。) 結果ドキュメントが作成された後、 HTTP GET リクエストを介して取得することができます。結果ドキュメントに付け加え、他のドキュメントも生成される可能性があります:

- *'失敗した'結果と共に実行されたジョブ*:エラーログは3 つのフォーマットで作成されます: テキスト、ロング XML、およびショート XML。これら3 つのドキュメントのURI は、(JSON フォーマットの) 結果ドキュメント 内に送信されます。URI は、HTTP GET リクエスト 内でエラードキュメントを取得するために 227 使用することができます。
- *'OK' の結果と共に実行されたジョブ*: ジョブの処理に成功し、XSLT 変換により作成された出力などの出力ドキュメント—が作成された場合。出力ファイルが作成され、JSON-フォーマットの結果ドキュメン内にURI が送信された場合。URI は、HTTP GET リクエスト 内で出力ドキュメントを取得するために使用されることができます。すべてのジョブに出力ファイルがあるわけではない点に注意してください。検証ジョブはその例です。ジョブは'OK' の状態で完了することができますが、警告または他のメッセージがエラーファイルに書き込まれることがあります。この場合、(出力ドキュメントに付け加え、)エラーファイルのURI も結果ドキュメントに送信されます。

これらのドキュメントの詳細とアクセス方法については、結果ドキュメントの取得 224 および エラー/出力ドキュメントの取得 227 を参照してください。

5.1.2.3 結果ドキュメントの取得

このセクション:

- 結果ドキュメント URI 224
- 結果ドキュメントの取得 225
  - エラードキュメントのURI を含む結果ドキュメント 225
  - 出力ドキュメントのURI を含む結果ドキュメント 226
  - URI を含まない結果ドキュメント 227
- 結果ドキュメント内にリストされるエラーと出力ドキュメントへのアクセス 227

結果ドキュメント URI

(例えば、検証が ポジティブ、(ドキュメントが有効) またはネガティブ(ドキュメントが無効)など、ジョブの結果に関わらず、結果ドキュメントは、ジョブが作成される都度作成されます。両方の場合とも、201 作成されましたメッセージが返されます。このメッセージはJSON フォーマットで、相対 結果ドキュメントのURIを含みます。JSON フラグメントは以下に類似します:

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

result オブジェクトは相対結果ドキュメントのURIを含みます。URI はサーバーアドレス[207]に対して相対的です。例えば、サーバーアドレスが以下の場合 http://localhost:8087/ (初期構成アドレス[204])。上のリストで指定されている展開された結果ドキュメントの URI は以下のとおりです:

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

メモ　正確なバージョン番号 /vN は、サーバーが返す番号です。(このドキュメント内に記載されている番号ではない可能性があります)。サーバーが返す番号は、現在のHTTP インターフェイスのバージョン番号。前のバージョン番号は、下位互換性により、サポートされているHTTP インターフェイスの古いバージョンを示します。

結果ドキュメントの取得

HTTP GET リクエスト内のドキュメントの展開された URI に提出された結果ドキュメントを取得します(*上を参照*[224])。 下に説明されるジェネリックの型の1 つにて、結果ドキュメントが返されます。

メモ　サーバーキューにジョブが並べられると、サーバーは結果ドキュメントの URI を返します。クライアントが、ジョブが開始される前に結果をリクエストした場合、(リクエストがキューの中にある場合) "status": "Dispatched" メッセージが返されます。ジョブが既に開始されており、まだ完了していない場合は、(例えば大きなジョブの場合) "status": "Running" メッセージが返されます。これらの場合、クライアントは結果ドキュメントの新しいリクエストを行うまで時間を置く必要があります。

メモ　下のサンプルドキュメントは、すべて制限されたクライアントアクセス[200]と仮定されます。ですから、エラードキュメント、メッセージドキュメント、および出力ドキュメントは、サーバー上の関連するジョブディレクトリに保存されます。結果ドキュメント内のこれらのドキュメントの URI はですからすべて相対 URI です。すべては、ファイル URI ではありません(無制限のクライアントアクセス[200]の場合生成されるの種類ではありません)。これらの URI の詳細に関しては、エラー/メッセージ/出力ドキュメントの取得[227]のセクションを参照してください。.

エラードキュメントの URI を含む結果ドキュメント

リクエストされたジョブが*失敗した*状態で完了した場合、ジョブはネガティブな結果を返したことになります。 例えば、検証ジョブが無効なドキュメントの結果を返した場合。ジョブ実行中に発生したエラーは、3 つのファイルフォーマット内で作成されたエラーログに保管されます: (i) テキスト、(ii) ロング-XML (詳細つきのエラーログ)、および(iii) ショート-XML (短い詳細付きのエラーログ)。 下の JSON リストを参照してください。

```
{
   "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"
       }
```

```
      }
    ]
}
```

以下の点に注意してください：

- ジョブにはサブジョブがあります。
- サブレベルのエラーは、トップレベルのジョブに反映されます。トップレベルのジョブの状態は、すべてのサブジョブの状態が OK であり始めて OK になります。
- 各ジョブまたはサブジョブにはそれぞれのエラーログがあります。
- エラーログ は、警告ログを含みます。ジョブが OK の状態で完了したにもかかわらず、結果ドキュメントは、エラーファイルの URI を含む場合があります。
- エラーファイルの URI はサーバーアドレスに対して相対的です（上を参照 224）。

出力ドキュメントの URI を含む結果ドキュメント

リクエストされたジョブが OK の状態で完了すると、ジョブはポジティブな結果を返します。例えば、検証ジョブがドキュメント有効な結果を返した場合。ジョブが出力ドキュメントを作成した場合。—例えば、XSLT 変換の結果— 出力ドキュメントの URI が返されます。下の JSON リストを参照してください。

```
{
    "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":
        {
        }
      }
    ]
}
```

以下の点に注意してください：

- 出力ファイルがジョブの `output` フォルダー内に作成されます。相対 URI を使用してファイルにアクセスします。
- 出力ファイルの URI はサーバーアドレスに相対します。（上を参照 224）。
- `output-mapping` アイテムは、ジョブリクエスト内のクライアントにより指定されているファイルロケーション上のジョブディレクトリ内の出力ドキュメントにマップします。ジョブリクエスト内のクライアントにより指定された出力ドキュメントのみにマッピングがあることに注意してください。（エラーファイルなどの）サーバーに生成されたジョブに関連したファイルにはマッピングがありません。

- または、URL "/v1/results/*JOBID*/output/zip" を使用して、ZIP アーカイブとして、特定のジョブのために生成されたすべての結果ドキュメントを取得することができます。この機能は、制約されていない filesystem モードでは使用することができません。ZIP アーカイブは、output-mapping オブジェクトを使用し実際の名前にマップされ戻される壊れたファイル名を含みます。

URI を含まない結果ドキュメント
リクエストされたジョブが OK の状態で完了すると、ジョブはポジティブな結果を返します。例えば、検証ジョブがドキュメント有効な結果を返した場合。検証または整形式のチェックなどジョブの一部が出力ドキュメントを作成しない場合。この型のジョブが OK の状態で完了すると、結果ドキュメントは、出力ドキュメントの URI または エラーログの URI を持ちません。下の JSON リストを参照してください。

```
{
    "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":
        {
        }
      }
    ]
}
```

以下の点に注意してください：

- 上のリストのサブジョブの出力とエラーコンポーネントの双方は空白です。
- ジョブは、OK の状態で完了することができますが、エラーファイル内でエラーとログされる警告または他のメッセージを含んでいる場合があります。この様な場合、ジョブが OK の状態で完了したにもかかわらず、結果ドキュメントは、エラーファイルの URI を含む場合があります。

結果ドキュメント内にリストされるエラーと出力ドキュメントへのアクセス

エラーと出力ドキュメントは、HTTP GET リクエストを使用してアクセスすることができます。この点については次のセクションエラー/出力ドキュメントの取得 227で説明されています。

5.1.2.4 エラー/メッセージ/出力ドキュメントの取得

結果ドキュメント 224 は、ファイル URI または エラードキュメント 225 の相対 URI、（ログなどの）メッセージドキュメント、および/または 出力ドキュメント 226 を含むことができます。（結果ドキュメントが URI を含まない 場合 227 もあります。）異なる種類の URI は、下に説明されています。 228

HTTP を介して、ドキュメントにアクセスするには、以下を行います:

1. 絶対 URI に対する結果ドキュメント内のファイルの相対 URI の展開 228
2. ファイルにアクセスするためにHTTP GET リクエスト内で展開された URI を使用する 229

エラー/メッセージ/出力ドキュメントのURI（結果ドキュメント内）

エラー、メッセージ、出力ドキュメントのURI を含む結果ドキュメント。エラーとメッセージドキュメントはジョブに関連したドキュメントで、サーバーにより生成されます。サーバー上のジョブディレクトリに常に保存されます。（XSLT 変換の出力など）出力ドキュメントは、次のいずれかの場所に保存することができます：

- ファイルによりアクセスすることのできるファイルロケーション。出力ファイルが任意の場所に保存されるためにサーバーはクライアント[クライアントのアクセスを無制限にする](205)（デフォルトの設定）ことを許可するように構成される必要があります。
- サーバー上のジョブディレクトリ。クライアントアクセスを制限するために[サーバーが構成されています。](203)

クライアントが出力ファイルの作成を指定すると、出力ファイルが保存される場所がサーバー構成ファイルの`server.unrestricted-filesystem-access`[205]オプションにより決定されます。

- アクセスが無制限の場合、ファイルはクライアントに指定された場所に保存されます。ドキュメントのために帰されたURI は、ファイルURIです。
- アクセスが制限される場合、ファイルはジョブディレクトリに保存され、そのURI は、相対 URI です。更に、クライアントにより指定されたファイルURL に対するこの相対 URL のマッピングがあります。（*出力ドキュメントのURI を含む結果ドキュメントのリスト*[226]*を参照してください。*）

要約すると、次の型のURI が発生します:

エラー/メッセージ ドキュメントのファイルURI
これらのドキュメントはサーバー上のジョブディレクトリ内に保存されます。ファイルURI は次のフォームを持ちます:

```
file:///<output-root-dir>/JOBID/message.doc
```

出力ドキュメントのファイルURI
これらのドキュメントはどこにでも保存することができます。ファイルURI は次のフォームを持ちます:

```
file:///<path-to-file>/output.doc
```

エラー/メッセージ/出力ドキュメントのHTTP URI
これらのドキュメントはサーバー上のジョブディレクトリ内に保存されます。URI は、サーバーアドレスに対して相対的で、フルHTTP URI に展開される必要があります。相対は次のフォームを持ちます:

`/vN/results/JOBID/error/error.txt`	エラードキュメントのため
`/vN/results/JOBID/output/verbose.log`	メッセージドキュメントのため
`/vN/results/JOBID/output/1`	出力ドキュメントのため

ドキュメントの出力の場合、出力マッピングが与えらえます（*サンプルリストを参照*[226]）。これらのマッピングは、結果ドキュメント内の各出力ドキュメント URI をクライアントリクエスト内の対応するドキュメントにマップします。

相対 URI の展開

絶対 HTTP URI に対する[結果ドキュメント](224) 内の相対 URI を、サーバーアドレスを持つURI に相対にプレフィックスを与えることにより、展開します。例えば、サーバーアドレスが以下の場合:

`http://localhost:8087/` (初期構成アドレス[204])

[結果ドキュメント](224) 内のエラーファイルの相対 URI は以下の通りです:

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

展開される絶対アドレスは以下のようになります

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

関連情報に関しては、サーバーの構成 203 および結果ドキュメントの取得 224 のセクションを参照してください。

ファイルにアクセスするためにHTTP GET リクエストを使用する

HTTP GET リクエスト内の展開されたURI を使用して、必要なファイルを取得します。 RaptorXML Server は、リクエストされたドキュメントを返します。

5.1.2.5 処理後のサーバーリソースの解放

RaptorXML Server は、ハードディスク上の処理済みのジョブに関連した結果ドキュメントファイル、一時的ファイル、およびエラーと出力ドキュメントファイルを保持します。これらのファイルは以下の2 つの方法の1 つにより削除されることができます:

- HTTP DELETE メソッドと共に結果ドキュメントのURI 224 を提供します。これは、エラーと出力ドキュメントを含む、提出された結果ドキュメントURI により示されたジョブに関連したすべてのファイルの削除します。
- 管理者によりサーバー上の個々のファイルを手動的に削除します。

HTTP DELETE メソッドと共に使用されるURI の構造は下に表示されています。フルURI は、サーバーアドレスと相対結果ドキュメントのURI により構成されていることに注意してください。

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

ディスク上のジョブの出力ディレクトリを検索するには、URI を以下の方法で作成します:

```
[<server.output-root-dir> サーバー構成ファイル参照 204] + [jobid 224]
```

メモ　多数のエラーと出力ドキュメントファイルが作成されることができるため、ハードディスクの使用をモニターし、使用中の環境と必要状況に合わせ削除を予定することが奨励されます。

5.2 COM/.NET API

RaptorXML はインストールされているコンピューター上でライセンス供与されています。.NET インターフェイスは、COM インターフェイスの周りのラッパーとしてビルドされます。RaptorXML Server のCOM and .NET インターフェイスは、単一のAPI を使用します: RaptorXML Server (オブジェクトレファレンス 241)のCOM/.NET API

RaptorXML を以下と使用することができます:

- COM インターフェイスを介した、JavaScript などのスクリプト言語
- .NET インターフェイスを介した、C# などのプログラム言語

5.2.1 COM インターフェイス

RaptorXML Server がインストールされると、RaptorXML Server は自動的にCOM サーバーオブジェクトとして登録されます。アプリケーション内部および呼 COM び出しへのプログラムサポートするスクリプト言語から呼び出されることができます。希望する場合は RaptorXML Server のインストールパッケージの場所を変更することができます。RaptorXML Server の登録を一度解除して、必要とされる場所にインストールすることが最善策です。この方法で、必要とされる登録解除および登録がインストールプロセスで実行されます。

登録の成功をチェックする

登録に成功すると、レジストリはクラス **RaptorXML.Server** クラスを含みます。このクラスは通常 **HKEY_LOCAL_MACHINE\SOFTWARE\Classes** の下で検索することができます。

コードサンプル

- VBScript サンプル 230 は、COM インターフェイスを介する RaptorXML API の使用が次のトピックで使用できることを示しています。
- このリストに対応するサンプルファイルは、RaptorXML アプリケーションフォルダーの `examples/API` フォルダー内で見つけることができます。

5.2.2 COM サンプル: VBScript

下のVBScript サンプルは以下の通り整理されています:

- RaptorXML COM オブジェクトのセットアップと初期化 230
- XML ファイルの検証 231
- XSLT 変換を実行し、結果を文字列として返す 231
- XQuery ドキュメントの処理し、結果をファイルに保存する 231
- コードとそのエントリポイントの実行シーケンスのセットアップ 232

```
' 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
```

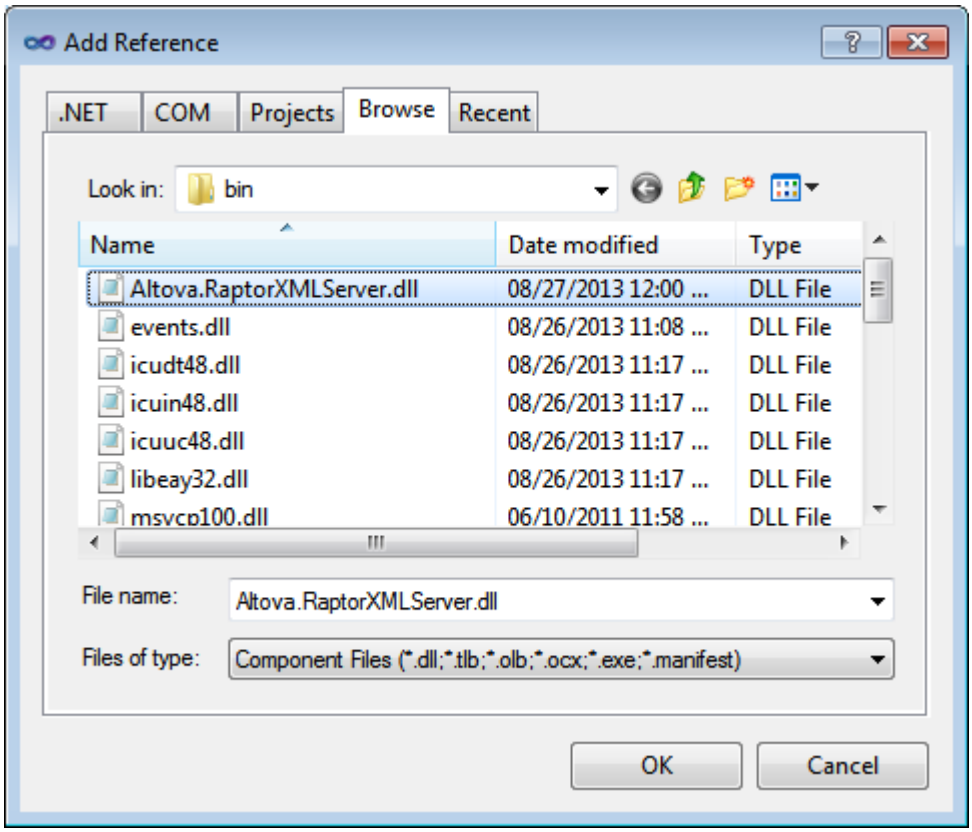
5.2.3 .NET インターフェイス

.NET インターフェイスは、RaptorXML COM インターフェイスの周りのラッパーとして構築されています。 Altova により署名されたプライマリ相互運用アセンブリとして与えらえます。名前空間 `Altova.RaptorXMLServer` を使用します。

RaptorXML DLL をレファレンスとして、Visual Studio .NET プロジェクトに追加する。
.NET プロジェクト内で RaptorXML を使用するには、プロジェクト内の RaptorXML DLL (`Altova.RaptorXMLServer.dll`) にレファレンスを追加します。RaptorXML Server インストールは、`Altova.RaptorXMLServer.dll` という名前の署名された DLL ファイルを含みます。RaptorXML インストーラーを使用して RaptorXML がインストールされる時、この DLL ファイルは、自動的に、グローバルアセンブリキャッシュ(GAC)に追加されます。 GAC は通常以下のフォルダーにあります: C:¥WINDOWS¥assembly。

RaptorXML DLL をレファレンスとして .NET プロジェクトに追加するには、以下を行います:

1. .NET プロジェクトを開き、「**プロジェクト | レファレンスの追加**」をクリックします。レファレンスの追加ダイアログがポップアップします (*下のスクリーンショット*)。



2. ブラウザータブから、次のフォルダーに移動します: `<RaptorXML application folder>/bin`、RaptorXML DLL `Altova.RaptorXMLServer.dll` を選択し、[OK] をクリックします。
3. 「コマンドビュー | オブジェクトブラウザー」を選択して、RaptorXML API のオブジェクトを確認します。

`Altova.RaptorXMLServer.dll` が.NET インターフェイスに対して使用することができるようになると、RaptorXML はCOM サーバーオブジェクトとして登録され、.NET プロジェクト内で使用可能な RaptorXML 機能を確認することができます。

メモ RaptorXML は、自動的にCOM サーバーオブジェクトとしてインストール中に登録されます。手動の登録は必要ありません。

メモ アクセスエラーを受信した場合、許可が正確に設定されていることを確認してください。コンポーネントサービスへ移動して、同じアカウントに許可を与え、RaptorXML を含む、アプリケーションプールを実行します。

コードサンプル

C# サンプル[233] とVisual Basic .NET サンプル[236] は、インターフェイスを介して RaptorXML API を次のトピック内でリストするの方法を表示します。.NET インターフェイスは次のトピック内でリストされます。これらのリスティングに対応するファイルは RaptorXML アプリケーションフォルダーの `examples/serverAPI` 内で検索することができます。

5.2.4 .NET Example: C#

C# のサンプルは以下を行います:

- RaptorXML .NET オブジェクトのセットアップと初期化[234]
- XML ファイルの検証[234]

- XSLT 変換を実行し、結果を文字列として返す 234
- XQuery ドキュメントの処理し、結果をファイルに保存する 235
- コードとそのエントリポイントの実行シーケンスのセットアップ 235


```
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 );
          }




       // 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() );
          }
      }
   }
}
```

5.2.5 .NET サンプル: Visual Basic .NET

Visual Basic のサンプルは以下を行います:

- RaptorXML .NET オブジェクトのセットアップと初期化 236
- XML ファイルの検証 236
- XSLT 変換を実行し、結果を文字列として返す 236
- XQuery ドキュメントの処理し、結果をファイルに保存する 237
- コードとそのエントリポイントの実行シーケンスのセットアップ 237

```
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
        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
```

5.3 Java API

RaptorXML API にJava コードからアクセスすることができます。Java コードから RaptorXML Server にアクセスするには、下にリストされるライブラリがクラスパス内に存在する必要があります。これらのライブラリは、インストールフォルダーの bin フォルダー内にインストールされています。

- RaptorXMLServer.jar: ライブラリは、HTTP リクエストを使用して RaptorXML サーバーと通信します。
- RaptorXMLServer_JavaDoc.zip: Java API のためのヘルプドキュメントを含む Javadoc ファイル。

メモ Java API を使用するには、Jar ファイルは、Java クラスパス上にある必要があります。インストールされた場所から参照するよりも、プロジェクトセットアップの条件に合う場合は、ファイルを希望する場所にコピーすることができます。

5.3.1 インターフェイスの概要

Java API は、com.altova.raptorxml パッケージ内にパッケージされています。RaptorXML クラスは、RaptorXMLFactory [241] オブジェクトを返す、getFactory() と称されるエントリポイントメソッドを提供します。RaptorXMLFactory インスタンスは、以下の呼び出しで作成されることができます: RaptorXML.getFactory()。

RaptorXMLFactory [241] インターフェイスは、検証と、(XSLT 変換などの) 更なる処理のためにエンジンオブジェクトを取得するメソッドを提供します。

RaptorXMLFactory

RaptorXMLFactory [241] のパブリックインターフェイスは、は次のリストの通り説明されています:

```
public interface RaptorXMLFactory
{
   public XMLValidator [258] getXMLValidator();
   public XMLDSig getXQuery();
   public XQuery [270] getXQuery();
   public XSLT [281] getXSLT();
   public void setServerName(String name) throws RaptorXMLException [250];
   public void setServerPath(String path) throws RaptorXMLException [250];
   public void setServerPort(int port) throws RaptorXMLException [250];
   public void setGlobalCatalog(String catalog);
   public void setUserCatalog(String catalog);
   public void setGlobalResourcesFile(String file);
   public void setGlobalResourceConfig(String config);
   public void setErrorFormat(RaptorXMLException [250] format);
   public void setErrorLimit(int limit);
   public void setReportOptionalWarnings(boolean report);
}
```

詳細に関しては、RaptorXMLFactory [241] および各 Java インターフェイスを参照してください。また サンプル Java プロジェクト [239] も参照することができます。

5.3.2 サンプル Java プロジェクト

下にリストされる Java コードは、どの様に基本的な機能にアクセスするかを表示しています。この点については、以下の部分で構成されています。

- サンプルフォルダーの検索、および RaptorXML COM オブジェクト インスタンスの作成。 239
- XML ファイルの検証 239
- XSLT 変換を実行し、結果を文字列として返す 239
- XQuery ドキュメントの処理後、結果を文字列として返す 240
- プロジェクトの実行 240

基本的な機能は、RaptorXML Server アプリケーションフォルダーの `examples/API` フォルダー内に含まれています。

```
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" );
        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();
       }

    }

}
```

5.4 サーバー API レファレンス

このセクションではRaptorXML API について説明されています: オブジェクトモデルとインターフェイスと列挙の詳細です。API 説明は、COM/.NET とJava インターフェイスの両方に適用されます。API の構造は、両方のインターフェイスと同じです。このため、各メソッド、プロパティ、および列挙はCOM/.NET とJava の署名のために個別に説明されています。

RaptorXML の機能の使用のための開始点はIServer[241] インターフェイス(COM/.NET) またはRaptorXMLFactory[241] クラス(Java) です。

5.4.1 インターフェイス/クラス

RaptorXML の機能を使用するための開始点はIServer[241] インターフェイス(COM/.NET) またはRaptorXMLFactory[241] クラス(Java) です。このオブジェクトはRaptorXML 機能を提供するオブジェクトを含んでいます: XML 検証、 XQuery ドキュメントとXML 署名処理、とXSLT 変換。

オブジェクトモデルの階層は下に示されるとおりで、インターフェイスは対応するセクションで詳細に説明されています。各インターフェイスのメソッドとプロパティは、そのインターフェイスのセクションで説明されています。

```
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)
```

5.4.1.1 IServer/RaptorXMLFactory

IServer/RaptorXMLFactory インターフェイスを使用して、RaptorXML エンジンにアクセスします。COM/.NET API 内のインターフェイスの名前は、Java API 内のインターフェイスの名前と異なることに注意してください!

- COM/.NET 内: **IServer**
- Java 内: **RaptorXMLFactory**

IServer/RaptorXMLFactory のメソッドとプロパティはこのセクションで説明されています。

Java API エントリポイント メソッド

Java API は、`com.altova.raptorxml` パッケージ内にパッケージされています。RaptorXML クラスは、RaptorXMLFactory[241] オブジェクトを返す、`getFactory()` と称されるエントリポイントメソッドを提供します。`RaptorXMLFactory` インスタンスは、以下の呼び出しで作成されることができます: `RaptorXML.getFactory()`。

5.4.1.1.1 メソッド

IServer (COM/.NET) と**RaptorXMLFactory** (Java) インターフェイスのメソッドは、RaptorXML エンジンまたはクラスのインスタンスをそれぞれ返します: XMLDSig、XML バリデーター、XSLT、とXQuery。

COM/.NET	Java
(XML 署名のための) GetXMLDsig 242	*(XML 署名のための)* getXMLDsig 242
GetXMLValidator 242	getXMLValidator 242
GetXQuery 242	getXQuery 242
GetXSLT 243	getXSLT 243

5.4.1.1.1.1 GetXMLDsig (for XML Signatures)

XML 署名 インターフェイス/クラス (**XMLDSig**) のインスタンスを返します。

COM and .NET

署名: **IXMLDSig GetXMLDSig()**

Java

署名: **public XMLDSig getXMLDSig()**

5.4.1.1.1.2 GetXMLValidator

XML 検証エンジンのインスタンスを返します。

COM and .NET

署名: **IXMLValidator** 258 **GetXMLValidator()**

Java

署名: **public XMLValidator** 258 **getXMLValidator()**

5.4.1.1.1.3 GetXQuery

XQuery Engine のインスタンスを返します。

COM and .NET

署名: **IXQuery**[270] **GetXQuery()**

Java

署名: **public XQuery**[270] **getXQuery()**

5.4.1.1.1.4　GetXSLT

XSLT エンジンのインスタンスを返します。

COM and .NET

署名: **IXSLT**[281] **GetXSLT()**

Java

署名: **public XSLT**[281] **getXSLT()**

5.4.1.1.2　プロパティ

IServer (COM/.NET) と **RaptorXMLFactory** (Java) インターフェイスのプロパティはこのセクションで説明されています。

COM/.NET	Java
APIMajorVersion[244]	getAPIMajorVersion[244]
APIMinorVersion[244]	getAPIMinorVersion[244]
APIServicePackVersion[245]	getAPIServicePackVersion[245]
ErrorFormat[245]	setErrorFormat[245]
ErrorLimit[245]	setErrorLimit[245]
GlobalCatalog[246]	setGlobalCatalog[246]
GlobalResourceConfig[246]	setGlobalResourceConfig[246]
GlobalResourcesFile[246]	setGlobalResourcesFile[246]
Is64Bit[247]	ss64Bit[247]
MajorVersion[247]	getMajorVersion[247]
MinorVersion[247]	getMinorVersion[247]

ProductName 248	getProductName 248
ProductNameAndVersion 248	getProductNameAndVersion 248
ReportOptionalWarnings 248	setReportOptionalWarnings 248
ServerName 249	setServerName 249
ServerPath 249	setServerPath 249
ServerPort 249	setServerPort 249
ServicePackVersion 250	getServicePackVersion 250
UserCatalog 250	setUserCatalog 250

5.4.1.1.2.1 APIMajorVersion

API のメジャーなバージョンを整数として返します。API が他のサーバーに接続されている場合、API のメジャーなバージョンは製品のメジャーなバージョン 247 と異なる場合があります

COM and .NET

署名: `int APIMajorVersion()`

Java

署名: `public int getAPIMajorVersion()`

5.4.1.1.2.2 APIMinorVersion

API のマイナーなバージョンを整数として返します。API が他のサーバーに接続されている場合、API のマイナーなバージョンは製品のマイナーバージョン 247 と異なる場合があります。

COM and .NET

署名: `int APIMinorVersion()`

Java

署名: `public int getAPIMinorVersion()`

5.4.1.1.2.3 APIServicePackVersion

API のサービス パック バージョンを整数で返します。API が他のサーバーに接続されている場合、API のサービス パックのバージョンは製品のサービス パックのバージョン 250 と異なる場合があります。

COM and .NET

署名: **int APIServicePackVersion()**

Java

署名: **public int getAPIServicePackVersion()**

5.4.1.1.2.4 ErrorFormat

RaptorXML エラーフォーマットを `ENUMErrorFormat` リテラルの 1 つに設定します。(Text、ShortXML、LongXML)。

COM and .NET

署名: **ErrorFormat(*ENUMErrorFormat* 293 format)**

Java

署名: **public void setErrorFormat(*ENUMErrorFormat* 293 format)**

5.4.1.1.2.5 ErrorLimit

RaptorXML 検証エラー制限を設定します。`limit` パラメーターは型 `int` (Java)、`uint` (COM/.NET) です。実行が中止されるまでの報告されるエラーの数を指定します。`limit` を無限に設定するには、 `-1` を使用します。(この設定によりすべてのエラーが報告されます)。デフォルトの値は 100 です。

COM and .NET

署名: **ErrorLimit(uint limit)**

Java

署名: **public int setErrorLimit(int limit)**

5.4.1.1.2.6 GlobalCatalog

メイン（エントリポイント）カタログファイルの場所を URL として設定します。提供される文字列は、使用するメインのカタログファイルの正確な場所を与える、絶対 URL である必要があります。

COM and .NET

署名: **GlobalCatalog(**string **catalog)**

Java

署名: **public** void **setGlobalCatalog(**string **catalog)**

5.4.1.1.2.7 GlobalResourceConfig

グローバルリソースのアクティブな構成を設定します。 `config` パラメーターは型 `String` です。アクティブなグローバルリソースにより使用される構成の名前を指定します。

COM and .NET

署名: **GlobalResourceConfig(**string **config)**

Java

署名: **public** void **setGlobalResourceConfig(**string **config)**

5.4.1.1.2.8 GlobalResourcesFile

グローバルリソースの XML ファイルの場所を URL として設定します。提供される文字列は、グローバルリソース XML ファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: **GlobalResourcesFile(**string **url)**

Java

署名: **public** void **setGlobalResourcesFile(**string **url)**

5.4.1.1.2.9 Is64Bit

アプリケーションが64 ビット実行可能ファイルかチェックします。アプリケーションが64 ビットの場合、`true` を返し、それ以外の場合 `false` を返します。*サンプル:* `Altova RaptorXML Server 2021r2sp1(x64)` に対しては、`true` を返します。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。

COM and .NET

署名: `boolean` **`Is64Bit()`**

Java

署名: **`public`** `boolean` **`is64Bit()`**

5.4.1.1.2.10 MajorVersion

製品のメジャーなバージョンを整数として返します *サンプル:* `Altova RaptorXML Server 2014r2sp1(x64)` に対しては、`16` が返されます(メジャーバージョン(`2014`) と最初の年の`1998` 差異である)。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。

COM and .NET

署名: `int` **`MajorVersion()`**

Java

署名: **`public`** `int` **`getMajorVersion()`**

5.4.1.1.2.11 MinorVersion

製品のマイナーバージョンを整数として返します *サンプル:* `Altova RaptorXML Server 2021r2sp1(x64)` に対しては、(マイナーバージョンの`r2` から)`2` が返されます。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。

COM and .NET

署名: `int` **`MinorVersion()`**

Java

署名: **`public`** `int` **`getMinorVersion()`**

5.4.1.1.2.12 ProductName

製品の名前を文字列として返します。 *サンプル:* `Altova RaptorXML Server 2021r2sp1(x64)` に対しては、`Altova RaptorXML Server` が返されます。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。

COM and .NET

署名: `string ProductName()`

Java

署名: `public string getProductName()`

5.4.1.1.2.13 ProductNameAndVersion

製品のサービス パック バージョンを整数として返します *サンプル:* `Altova RaptorXML Server 2021r2sp1(x64)` に対しては、`Altova RaptorXML Server 2021r2sp1(x64)` が返されます。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。

COM and .NET

署名: `string ProductNameAndVersion()`

Java

署名: `public string getProductNameAndVersion()`

5.4.1.1.2.14 ReportOptionalWarnings

警告のレポート を有効化または無効化します。`true` の値は、警告を有効化します。 `false` は無効化にします。

COM and .NET

署名: `ReportOptionalWarnings(boolean report)`

Java

署名: `public void setReportOptionalWarnings(boolean report)`

5.4.1.1.2.15 ServerName

HTTP サーバーの名前を設定します。エラーが発生すると、RaptorXMLException 250 が挙げられます。入力パラメーターは、HTTP サーバー名を与える文字列です。

COM and .NET

署名: **ServerName**(string **name**)

Java

署名: public void **setServerName**(string **name**)

5.4.1.1.2.16 ServerPath

HTTP サーバーへのパスを URL のフォームで指定します。

COM and .NET

署名: **ServerPath**(string **path**)

Java

署名: public void **setServerPath**(string **path**)

5.4.1.1.2.17 ServerPort

サーバーがアクセスされる HTTP サーバー上のポートを設定します。ポートは、HTTP リクエストが正確にサービスにアドレスされるように、固定されている必要があります エラーが発生すると、RaptorXMLException 250 が挙げられます。入力パラメーターは、サーバーがアクセスされる HTTP サーバー上のポートを設定します。

COM and .NET

署名: **ServerPort**(int **port**)

Java

署名: public void **setServerPort**(int **port**)

5.4.1.1.2.18 ServicePackVersion

製品のサービスパックバージョンを整数として返します。 *サンプル:* `RaptorXML Server 2021r2sp1(x64)` に対しては、(サービスパックバージョン `sp1` から) `1` が返されます 。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。

COM and .NET

署名: `int ServicePackVersion()`

Java

署名: `public int getServicePackVersion()`

5.4.1.1.2.19 UserCatalog

URL としてユーザー定義カタログファイルの場所を指定します。与えられた文字列は、使用するユーザーカタログファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: `UserCatalog(string userCatalog)`

Java

署名: `public void setUserCatalog(string userCatalog)`

5.4.1.2 RaptorXMLException

処理中に発生するエラーに関する情報を含む例外を生成します。`message` パラメーターはエラーに関する情報を与えます。

COM and .NET

署名: `RaptorXMLException(string message)`

Java

署名: `public void RaptorXMLException(string message)`

5.4.1.3 (XML 署名のための) XMLDSig

IXMLDSig/XMLDSig インターフェイス/クラスのメソッドは、XML ドキュメントの署名、署名済みのドキュメントの検証、(新規の署名と共に) 以前に署名されているドキュメントが変更されている場合更新を行う、および署名を削除するために使用することができます。

COM/.NET API 内のインターフェイスの名前は、Java API 内のクラスのインターフェイスの名前と異なることに注意してください。:

- COM/.NET 内: **IXMLDSig**
- Java 内: **XMLDSig**

5.4.1.3.1 メソッド

IXMLDSig インターフェイス (COM/.NET) と **XMLDSig** クラス (Java) のメソッドはこのセクションで説明されています。

5.4.1.3.1.1 ExecuteRemove

サイン済みの XML ファイルの XML 署名を削除し、ファイルロケーションの URL を提供する文字列である、`outputPath` により定義された出力のロケーションに出力される署名されていないドキュメントを保存します。成功した場合結果は、`true` を返し、失敗した場合は `false` を返します。

COM and .NET

署名: `boolean` **`ExecuteRemove(`**`string` **`outputPath)`**

Java

署名: **`public`** `boolean` **`executeRemove(`**`string` **`outputPath)`**

5.4.1.3.1.2 ExecuteSign

(`signatureType` と `canonicalizationMethod` パラメーター内に存在することを前提にして、使用可能な値に関しては *xmlsignature-sign CLI コマンド* 149 *を参照してください*) 指定される署名オプションに従い XML ドキュメントを署名します。出力ファイルの URL を提供する `outputPath` により出力ファイルは定義されています。成功した場合結果は、`true` を返し、失敗した場合は `false` を返します。

COM and .NET

署名: `boolean` **`ExecuteSign(`**`string` **`outputPath,`** `string` **`signatureType,`**`string` **`canonicalizationMethod)`**

Java

署名: `public boolean executeSign(string outputPath, string signatureType,string canonicalizationMethod)`

5.4.1.3.1.3 ExecuteUpdate

署名済みのXML ファイル内でXML 署名を更新します。ドキュメントが変更されると、更新済みのXML 署名は異なります。それ以外の場合、更新された署名は、前の署名と同じです。出力ファイルは、更新された署名を持つファイルのURL を提供する文字列である `outputPath` 内で指定されます。成功した場合結果は、`true` を返し、失敗した場合は `false` を返します。

(i) HMAC 秘密キー 256 プロパティ、または、(ii) certificate-name 253 とcertificate-store 254 プロパティを指定する必要があります。証明書オプションが指定されると、XML ドキュメントを以前署名される証明書オプションに一致する必要があります(Linux とmacOS では証明書ストアオプションは現在サポートされていないことご注意してください)。

COM and .NET

署名: `boolean ExecuteUpdate(string outputPath)`

Java

署名: `public boolean executeUpdate(string outputPath)`

5.4.1.3.1.4 ExecuteVerify

署名検証の結果を返します。:署名に成功すると **`true`** を返し、それ以外は **`false`** を返します。

COM and .NET

署名: `boolean ExecuteVerify()`

Java

署名: `public boolean executeVerify()`

5.4.1.3.1.5 InterfaceSupportsErrorInfo

インターフェイスのためにご使用さえるエラーメッセージを指定します。

COM and .NET

署名: `InterfaceSupportsError(guid riid)`

Java

署名: `public void setInterfaceSupportsError(guid riid)`

5.4.1.3.2 プロパティ

IXMLDSig インターフェイス(COM/.NET) と**XMLDSig** クラス(Java) のプロパティはこのセクションで説明されています。

5.4.1.3.2.1 AbsoluteReferenceUri

署名済みのドキュメント URI が絶対(`true`)、または、相対的(`false`)に読み取られるか指定します。デフォルトは`false` です。

COM and .NET

署名: `AbsoluteReferenceUri(boolean absoluteuri)`

Java

署名: `public void setAbsoluteReferenceUri(boolean absoluteuri)`

5.4.1.3.2.2 AppendKeyInfo

署名に`KeyInfo` 要素を含むかを指定します。デフォルトは`false` です。

COM and .NET

署名: `AppendKeyInfo(boolean include)`

Java

署名: `public void setAppendKeyInfo(boolean inlude)`

5.4.1.3.2.3 CertificateName

署名のために使用された証明書の名前。

Windows
選択された`--certificate-store` からの署名所の**Subject** 名前です。

証明書をリストするサンプル(PowerShell)

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

サンプル: **--certificate-name==certificate1**

Linux/MacOS

--certname はPEM エンコード済みのX.509v3 証明書のファイルの名前を秘密キーを使用して指定します。このようなファイルは拡張子 **.pem** を持っています。

サンプル: **--certificate-name==/path/to/certificate1.pem**

COM and .NET

署名: **CertificateName(string name)**

Java

署名: **public void setCertificateName(string name)**

5.4.1.3.2.4 CertificateStore

--certificate-name を使用して指定されている証明書が保存されている場所です。

Windows

cert://CurrentUser の下の証明書ストア。**% ls cert://CurrentUser/** を使用して使用することのできる証明書ストアを(PowerShell の下に)リストすることができます。証明書は次のようにリストされます:

```
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
```

サンプル: **--certificate-store==MyCertStore**

Linux/MacOS
`--certstore` オプションは現在サポートされていません。

COM and .NET

署名: **`CertificateStore(string filelocation)`**

Java

署名: **`public void setCertificateStore(string filelocation)`**

5.4.1.3.2.5 DigestMethod

入力 XML ファイルのダイジェスト値を計算するために使用されるアルゴリズム。使用可能な値: `sha1|sha256|sha384|sha512`.

COM and .NET

署名: **`DigestMethod(string algo)`**

Java

署名: **`public void setDigestMethod(string algo)`**

5.4.1.3.2.6 HMACOutputLength

HMAC アルゴリズムの出力を `length` ビットに切り捨てます。指定されると、値は以下の通りになります

- 8 の積
- 80 より大きな数
- 基のハッシュアルゴリズムの出力の長さの半分よりも長い数

COM and .NET

署名: **`HMACOutputLength(int length)`**

Java

署名: **`public void setHMACOutputLength(int length)`**

5.4.1.3.2.7 HMACSecretKey

HMAC は秘密キーを共有しています。少なくとも6文字以上の文字が必要です。

COM and .NET

署名: **HMACSecretKey(string key)**

Java

署名: **public void setHMACSecretKey(string key)**

5.4.1.3.2.8 InputXMLFileName

処理される XML ドキュメントの場所を URL として設定します。与えられる文字列は、XML ファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: **InputXMLFileName(string filepath)**

Java

署名: **public void setInputXMLFileName(string filepath)**

5.4.1.3.2.9 LastErrorMessage

RaptorXML エンジンからの最後のエラーメッセージを文字列として取得します。

COM and .NET

署名: **string LastErrorMessage()**

Java

署名: **public string getLastErrorMessage()**

5.4.1.3.2.10 SignatureMethod

署名の生成のために使用されるアルゴリズムを指定します。

証明書が使用される場合
証明書が指定されている場合 `SignatureMethod` は任意で、このパラメーターのための値は、証明書により生成されます。指定されている場合、証明書により使用されているアルゴリズムに一致する必要があります。サンプル: `rsa-sha256`。

--hmac-secret-key が使用される場合
`HMACSecretKey` が使用される場合、 `SignatureMethod` は必須です。サポートされる HMAC アルゴリズムの一つである必要があります:

- `hmac-sha256`
- `hmac-sha386`
- `hmac-sha512`
- `hmac-sha1` (仕様により奨励されていません)

サンプル: **hmac-sha256**

COM and .NET

署名: **SignatureMethod(string algo)**

Java

署名: **public void setSignatureMethod(string algo)**

5.4.1.3.2.11 Transforms

入力ドキュメントに適用された XML 署名変換を指定します。サポートされている値は以下のとおりです:

- Canonical XML 1.0 のための `REC-xml-c14n-20010315` (コメントを省略)
- Canonical XML 1.1 のための `xml-c14n11` (コメントを省略)
- Exclusive XML Canonicalization 1.0 のための `xml-exc-c14n#` (コメントを省略)
- Canonical XML 1.0 のための `REC-xml-c14n-20010315#WithComments` (コメント付き)
- Canonical XML 1.1 のための `xml-c14n11#WithComments` (コメント付き)
- Exclusive XML Canonicalization 1.0 のための `xml-exc-c14n#WithComments` (コメント付き)
- `base64`
- `strip-whitespaces` Altova 拡張子

COM and .NET

署名: **Transforms(string value)**

Java

署名: `public void setTransforms(string value)`

5.4.1.3.2.12 WriteDefaultAttributes

署名済みドキュメント内のDTD からのデフォルトの値を含むかを指定します。

COM and .NET

署名: `WriteDefaultAttributes(boolean write)`

Java

署名: `public void setWriteDefaultAttributes(boolean write)`

5.4.1.4 XMLValidator

IXMLValidator/XMLValidator インターフェイス/クラスは以下を行うメソッドを提供します (i) 多種のドキュメントを検証する (ii) ドキュメントの整形式をチェックするおよび (iii) Avro バイナリから Avro スキーマを抽出する。Python スクリプトを介して追加処理を提供することができます。

COM/.NET API 内のインターフェイスの名前は Java API 内のクラス名の名前とは異なることご注意してください!

- COM/.NET 内: **IXMLValidator**
- Java 内: **XMLValidator**

5.4.1.4.1 メソッド

IXMLValidator インターフェイス (COM/.NET) と **XMLValidator** クラス (Java) のメソッドはこのセクションで説明されています。

5.4.1.4.1.1 AddPythonScriptFile

検証のために提供されるファイルを追加処理する Python スクリプトファイルを指定します。提供される文字列は、Python ファイルの正確な場所を与える絶対 URL である必要があります。Python スクリプトは、RaptorXML Server とバンドルされている Python パッケージと共に処理されます。バンドルされている Python パッケージはバージョン 3.7 です。

COM and .NET

署名: `AddPythonScriptFile(string filepath)`

Java

署名: `public void addPythonScriptFile(string filepath)`

5.4.1.4.1.2 ClearPythonScriptFile

AddPythonScriptFile メソッド、または、**PythonScriptFile** プロパティを使用して追加された Python スクリプトファイルをクリアします。

COM and .NET

署名: `ClearPythonScriptFile()`

Java

署名: `public void clearPythonScriptFile()`

5.4.1.4.1.3 ExtractAvroSchema

Avro スキーマをバイナリファイルから抽出します。`outputPath` パラメーターは、出力の場所を指定する絶対 URL です。成功時にはブール値の `true` が、失敗時には `false` が返されます。エラーが発生すると、**RaptorXMLException** 250 が挙げられます。`LastErrorMessage` を使用して追加情報を使用します。

COM and .NET

署名: `ExtractAvroSchema(string outputPath)`

Java

署名: `public void addPythonScriptFile(string outputPath)`

5.4.1.4.1.4 InterfaceSupportsErrorInfo

インターフェイスのために使用されるエラーメッセージを指定します。

COM and .NET

署名: `InterfaceSupportsError(guid riid)`

Java

署名: `public void setInterfaceSupportsError(guid riid)`

5.4.1.4.1.5 IsValid

XML ドキュメント、スキーマドキュメント、または、DTD ドキュメントの検証の結果を返します。検証するドキュメントの型は、ENUMValidationType[296] リテラルを値として取ります。結果は成功時には true 、失敗時には false です。エラーが発生すると、**RaptorXMLException**[250] が挙げられます。LastErrorMessage を追加情報にアクセスするために使用します。

COM and .NET

署名: boolean **IsValid(**ENUMValidationType[296] **type)**

Java

署名: **public** boolean **isValid(**ENUMValidationType[296] **type)**

5.4.1.4.1.6 IsWellFormed

XML ドキュメントまたは DTD ドキュメントの整形式チェックの結果を返します。チェックするドキュメントの型は、ENUMWellformedCheckType[297] を取るリテラルです。type パラメーターにより指定されています。 結果は成功時には true 、失敗時には false です。エラーが発生すると、 **RaptorXMLException**[250] が挙げられます。LastErrorMessage を追加情報にアクセスするために使用します。

COM and .NET

署名: boolean **isWellFormed(**ENUMWellformedCheckType[296] **type)**

Java

署名: **public** boolean **isWellFormed(**ENUMWellformedCheckType[296] **type)**

5.4.1.4.2 プロパティ

IXMLValidator インターフェイス(COM/.NET) と **XMLValidator** クラス(Java) のプロパティはこのセクションで説明されています。

5.4.1.4.2.1 AssessmentMode

ENUMAssessmentMode[292] リテラルにより与えられる XML 検証 (Strict/Lax) のモードの評価モードを設定します。

COM and .NET

署名: **AssessmentMode(**ENUMAssessmentMode[292] **mode)**

Java

署名: `public void setAssessmentMode(ENUMAssessmentMode 292 mode)`

5.4.1.4.2.2 AvroSchemaFileName

使用する外部 Avro スキーマの場所を URL として設定します。与えられる文字列は、Avro スキーマファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: `AvroSchemaFileName(string url)`

Java

署名: `public void setAvroSchemaFileName(string url)`

5.4.1.4.2.3 AvroSchemaFromText

使用される Avro スキーマドキュメントのコンテンツです。与えられる文字列は、使用される Avro スキーマドキュメントのコンテンツです。

COM and .NET

署名: `AvroSchemaFromText(string avroschema)`

Java

署名: `public void setAvroSchemaFromText(string avroschema)`

5.4.1.4.2.4 DTDFileName

検証に使用される DTD ドキュメントの場所を URL として設定します。提供される文字列は、使用する DTD ドキュメントの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: `DTDFileName(string url)`

Java

署名: `public void setDTDFileName(string url)`

5.4.1.4.2.5 DTDFromText

検証のために使用するDTD ドキュメントのコンテンツを提供します。提供される文字列は、使用するDTD ドキュメントのコンテンツです。

COM and .NET

署名: `DTDFromText(string dtdtext)`

Java

署名: `public void setDTDFromText(string dtdtext)`

5.4.1.4.2.6 EnableNamespaces

名前空間を考慮した処理を有効化します。これは、XML インスタンスの正しくない名前空間によるエラーをチェックするために役に立ちます。`true` の値は、名前空間を考慮した処理を有効化します。`false` の値は無効化します。デフォルトは、`false` です。

COM and .NET

署名: `EnableNamespaces(boolean enableNS)`

Java

署名: `public void setEnableNamespaces(boolean enableNS)`

5.4.1.4.2.7 InputFileArray

入力データとして使用される XML ファイルの URL の配列を提供します。プロパティは、各 XML ファイルの絶対 URL を含むオブジェクトを文字列として提供します。

COM and .NET

署名: `InputFileArray(object fileArray)`

Java

署名: `public void setInputFileArray(object fileArray)`

5.4.1.4.2.8 InputFileName

検証される XML ファイルを指定します。提供された文字列 は、使用する XML ファイルのベースロケーションを与える絶対 URL である必要があります。

COM and .NET

署名: `InputFileName(string filepath)`

Java

署名: `public void setInputFileName(string filepath)`

5.4.1.4.2.9 InputFromText

処理する XML ドキュメントのコンテンツを提供します。提供される文字列は処理する XML ドキュメントのコンテンツです。

COM and .NET

署名: `InputFromText(string doc)`

Java

署名: `public void setInputFromText(string doc)`

5.4.1.4.2.10 InputTextArray

入力データとして使用されるテキストファイルの URL の配列を提供します。プロパティは、各テキストファイルの絶対 URL を含むオブジェクトを文字列として提供します。

COM and .NET

署名: `InputTextArray(object textfileArray)`

Java

署名: `public void setInputTextArray(object textfileArray)`

5.4.1.4.2.11 InputXMLFileName

処理される XML ドキュメントの場所を URL として設定します。与えられる文字列は、XML ファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: `InputXMLFileName(string url)`

Java

署名: `public void setInputXMLFileName(string url)`

5.4.1.4.2.12 InputXMLFromText

処理する XML ドキュメントのコンテンツを提供します。提供される文字列は処理する XML ドキュメントのコンテンツです。

COM and .NET

署名: `InputXMLFromText(string xml)`

Java

署名: `public void setInputXMLFromText(string xml)`

5.4.1.4.2.13 Json5

`true` に設定されている場合、JSON 5 へのサポートを有効化します。

COM and .NET

署名: `Json5(boolean json5)`

Java

署名: `public void setJson5(boolean json5)`

5.4.1.4.2.14　JSONSchemaFileName

JSON インスタンスドキュメント検証のために使用される JSON スキーマファイルの場所を URL として設定します。JSON スキーマファイルの実際のロケーションを与える絶対 URL である必要のある文字列を返します。

COM and .NET

署名: `JSONSchemaFileName(string url)`

Java

署名: `public void setJSONSchemaFileName(string url)`

5.4.1.4.2.15　JSONSchemaFromText

JSON インスタンスドキュメントの検証のために使用される JSON スキーマドキュメントのテキストコンテンツである文字列を返します。

COM and .NET

署名: `JSONSchemaFromText(string jsonschema)`

Java

署名: `public void setJSONSchemaFromText(string jsonschema)`

5.4.1.4.2.16　LastErrorMessage

RaptorXML エンジンからの最後のエラーメッセージを文字列として取得します。

COM and .NET

署名: `string LastErrorMessage()`

Java

署名: `public string getLastErrorMessage()`

5.4.1.4.2.17 ParallelAssessment

パラレルスキーマの有効性の評価 186 を有効化/無効化します。

COM and .NET

署名: **ParallelAssessment**(boolean **enable**)

Java

署名: public void **setParallelAssessment**(boolean **enable**)

5.4.1.4.2.18 PythonScriptFile

検証のために提供されるファイルを追加処理するPython スクリプトファイルを指定します。提供される文字列は、Python ファイルの正確な場所を与える絶対 URL である必要があります。Python スクリプトは、RaptorXML Server とバンドルされているPython パッケージと共に処理されます。バンドルされているPython パッケージはバージョン 3.7 です。

COM and .NET

署名: **PythonScriptFile**(string **filepath**)

Java

署名: public void **setPythonScriptFile**(string **filepath**)

5.4.1.4.2.19 SchemaFileArray

外部 XML スキーマとして使用されるXML スキーマファイルのコレクションを提供します。ファイルはURL により識別されます。それぞれが XML スキーマファイルの絶対 URL である、文字列のコレクション。

COM and .NET

署名: **SchemaFileArray**(object **urlArray**)

Java

署名: public void **setSchemaFileArray**(object **urlArray**)

5.4.1.4.2.20 SchemaFileName

検証される XML スキーマドキュメントの場所を URL として設定します。提供される文字列は、使用する XML スキーマファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: **SchemaFileName(string filepath)**

Java

署名: **public void setSchemaFileName(string filepath)**

5.4.1.4.2.21 SchemaFromText

使用される XML スキーマファイルのコンテンツを提供します。提供される文字列は、使用する XML スキーマドキュメントのコンテンツです。

COM and .NET

署名: **SchemaFileName(string xsdText)**

Java

署名: **public void setSchemaFileName(string xsdText)**

5.4.1.4.2.22 SchemaImports

`xs:import` 要素の属性の値に従い、スキーマインポートがどのように扱われるか指定します。扱いは選択された `ENUMSchemaImports` リテラルにより指定されます。

COM and .NET

署名: **SchemaImports(ENUMSchemaImports 294 importOption)**

Java

署名: **public void setSchemaImports(ENUMSchemaImports 294 importOption)**

5.4.1.4.2.23 SchemalocatiionHints

スキーマの検索に使用されるメカニズムを指定します。メカニズムは選択された ENUMSchemaImports リテラルにより指定されます。

COM and .NET

署名: **SchemalocationHints(ENUMLoadSchemalocation[293] hint)**

Java

署名: **public void setSchemalocationHints(ENUMLoadSchemalocation[293] hint)**

5.4.1.4.2.24 SchemaMapping

スキーマの検索内で使用されるマッピングを設定します。マッピングは選択された ENUMSchemaMapping リテラルにより指定されます。

COM and .NET

署名: **SchemaMapping(ENUMSchemaMapping[295] mappingOption)**

Java

署名: **public void setSchemaMapping(ENUMSchemaMapping[295] mappingOption)**

5.4.1.4.2.25 SchemaTextArray

複数のスキーマファイルのコンテンツを提供します。それぞれが入力 XML スキーマドキュメントのコンテンツである、文字列のコレクション。

COM and .NET

署名: **SchemaTextArray(object schemaDocs)**

Java

署名: **public void setSchemaTextArray(object schemaDocs)**

5.4.1.4.2.26 Streaming

ストリーミング検証を有効化します。ストリーミングモードでは、メモリに保管されるデータが最小化され、処理がより速くなります。`true` の値は、ストリーミングを有効化します。`false` の値は無効化します。デフォルトは `true` です。

COM and .NET

署名: **`Streaming(boolean enable)`**

Java

署名: **`public void setStreaming(boolean enable)`**

5.4.1.4.2.27 XincludeSupport

`XInclude` 要素の使用を有効化または無効化します。true の値は、XInclude へのサポートを有効化します。false の値は無効化します。デフォルト値はfalse です。

COM and .NET

署名: **`XincludeSupport(boolean xinclude)`**

Java

署名: **`public void setXincludeSupport(boolean xinclude)`**

5.4.1.4.2.28 XMLValidationMode

有効性または整形式をチェックするかを決定する、`ENUMXMLValidationMode` 298 の列挙リテラルである XML 検証モードを設定します。

COM and .NET

署名: **`XMLValidationMode(ENUMXMLValidationMode` 298 **`valMode)`**

Java

署名: **`public void setXMLValidationMode(ENUMXMLValidationMode` 298 **`valMode)`**

5.4.1.4.2.29 XSDVersion

検証されるXMLドキュメントに対して、XMLスキーマバージョンを指定します。ENUMXSDVersion[301] は、の列挙リテラルです。

COM and .NET

署名: **XSDVersion(ENUMXSDVersion[301] version)**

Java

署名: **public void setXSDVersion(ENUMXSDVersion[301] version)**

5.4.1.5 XQuery

IXQuery/XQuery インターフェイス/クラスのための(i) XQuery ドキュメントの実行とXQuery 更新、(ii) XQuery 関連ドキュメントの検証を行うメソッドを提供します。外部変数を使用して実行のためにデータを提供することもできます。

COM/.NET API 内のインターフェイスの名前は、Java API 内のクラスのインターフェイスの名前と異なることに注意してください。:

- COM/.NET 内: **IXQuery**
- Java 内: **XQuery**

5.4.1.5.1 メソッド

IXQuery インターフェイス(COM/.NET) と**XQuery** クラス(Java) のメソッドはこのセクションで説明されています。

5.4.1.5.1.1 AddExternalVariable

新規の外部変数の名前と値を追加します。各外部変数とその値は、メソッドへの個別の呼び出し内で指定されます。変数はXQuery ドキュメント内で(任意の型宣言と共に)宣言されている必要があります。変数が文字列の場合、値を一重引用符で囲みます。name パラメーターはQName である変数の値を文字列としてホールドします。value パラメーターは変数の値を文字列としてホールドします。

COM and .NET

署名: **AddExternalVariable(string name, string value)**

Java

署名: **public void addExternalVariable(string name, string value)**

5.4.1.5.1.2 ClearExternalVariableList

AddExternalVariable 270 メソッドにより作成済みの外部変数のリストをクリアします。

COM and .NET

署名: **ClearExternalVariableList()**

Java

署名: `public void` **clearExternalVariableList()**

5.4.1.5.1.3 Execute

EngineVersion 274 プロパティ内で名前の付けられた XQuery バージョンに従い XQuery 変換を実行し、`outputFile` パラメータ内で名前が付けられている通り出力ファイルの結果を保存します。パラメータは出力ファイルのロケーション（パスとファイル名）を提供する文字列です。成功した場合結果は、`true` を返し、失敗した場合は `false` を返します。エラーが発生すると、RaptorXMLException 250 が挙げられます。`LastErrorMessage` プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: `boolean` **Execute(**`string` **outputFile)**

Java

署名: `public boolean` **execute(**`string` **outputFile)**

5.4.1.5.1.4 ExecuteAndGetResultAsString

EngineVersion 274 プロパティ内で名前の付けられた XQuery Update 仕様に従い XQuery Update を実行し、結果を文字列として返します。このメソッドはチャートやセカンダリの結果などの追加結果ファイルを生成しません。また、.docx OOXML ファイルなどのバイナリの結果をホールドしません。追加の出力ファイルが必要な場合、Execute 282 メソッドを使用します。

COM and .NET

署名: `string` **ExecuteAndGetResultAsString()**

Java

署名: `public string` **executeAndGetResultAsString()**

5.4.1.5.1.5 ExecuteUpdate

XQueryUpdateVersion 281 プロパティ内で名前の付けられたXQuery Update 仕様に従いXQuery Update を実行し、outputFile パラメータ内で名前が付けられている通り出力ファイルの結果を保存します。パラメータは出力ファイルのロケーション（パスとファイル名）を提供する文字列です。成功した場合結果は、true を返し、失敗した場合は false を返します。エラーが発生すると、RaptorXMLException 250 が挙げられます。LastErrorMessage プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: boolean **ExecuteUpdate(**string **outputFile)**

Java

署名: **public** boolean **executeUpdate(**string **outputFile)**

5.4.1.5.1.6 ExecuteUpdateAndGetResultAsString

XQueryUpdateVersion 281 プロパティ内で名前の付けられたXQuery Update 仕様に従いXQuery Update を実行し、結果を文字列として返します。このメソッドはチャートやセカンダリの結果などの追加結果ファイルを生成しません。また、.docx OOXML ファイルなどのバイナリの結果をホールドしません。

COM and .NET

署名: string **ExecuteUpdateAndGetResultAsString()**

Java

署名: **public** string e**xecuteUpdateAndGetResultAsString()**

5.4.1.5.1.7 IsValid

EngineVersion 274 プロパティ内で名前の付けられているXQuery 仕様に従いXQuery ドキュメントの検証の結果を返します。成功した場合結果は、true を返し、失敗した場合は false を返します。エラーが発生すると、RaptorXMLException 250 が挙げられます。LastErrorMessage プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: boolean **IsValid()**

Java

署名: **public** boolean **isValid()**

5.4.1.5.1.8　IsValidUpdate

XQueryUpdateVersion [281] プロパティ内で名前の与えられているXQuery Update 仕様に従いXQuery Update ドキュメントの検証の結果を返します。成功した場合結果は、`true` を返し、失敗した場合は`false` を返します。エラーが発生すると、RaptorXMLException [250] が挙げられます。`LastErrorMessage` プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: `boolean` **`IsValidUpdate()`**

Java

署名: **`public`** `boolean` **`isValidUpdate()`**

5.4.1.5.2　プロパティ

`IXQuery` インターフェイス（COM/.NET）と**`XQuery`** クラス（Java）のプロパティはこのセクションで説明されています。

5.4.1.5.2.1　AdditionalOutputs

最後に実行されたジョブの追加の出力を返します。

COM and .NET

署名: `string` **`AdditionalOutputs()`**

Java

署名: **`public`** `string` **`getAdditionalOutputs()`**

5.4.1.5.2.2　ChartExtensionsEnabled

Altova チャート 拡張関数を有効化または無効化します。`true` の値は、チャート 拡張子を有効化します。`false` は無効化します。デフォルト値は `true` です。

COM and .NET

署名: **ChartExtensionsEnabled(boolean enable)**

Java

署名: **public void setChartExtensionsEnabled(boolean enable)**

5.4.1.5.2.3 DotNetExtensionsEnabled

Visual Studio .NET 拡張関数を有効化または無効化します。 `true` の値は、.NET 拡張子を有効化します。 `false` の値は無効化にします。デフォルト値は `true` です。

COM and .NET

署名: **DotNetExtensionsEnabled(boolean enable)**

Java

署名: **public void setDotNetExtensionsEnabled(boolean enable)**

5.4.1.5.2.4 EngineVersion

使用する XQuery バージョンを指定します。プロパティの値は ENUMXQueryVersion 300 リテラルです。

COM and .NET

署名: **EngineVersion(ENUMXQueryVersion 300 version)**

Java

署名: **public void setEngineVersion(ENUMXQueryVersion 300 version)**

5.4.1.5.2.5 IndentCharacters

出力内でインデントとして使用される文字列を設定します。

COM and .NET

署名: **IndentCharacters(string indentChars)**

Java

署名: `public void setIndentCharacters(string indentChars)`

5.4.1.5.2.6　InputXMLFileName

処理される XML ドキュメントの場所を URL として設定します。与えられる文字列は、XML ファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: `InputXMLFileName(string url)`

Java

署名: `public void setInputXMLFileName(string url)`

5.4.1.5.2.7　InputXMLFromText

処理する XML ドキュメントのコンテンツを提供します。提供される文字列は処理する XML ドキュメントのコンテンツです。

COM and .NET

署名: `InputXMLFromText(string xml)`

Java

署名: `public void setInputXMLFromText(string xml)`

5.4.1.5.2.8　JavaBarcodeExtensionLocation

バーコード拡張ファイルの場所を指定します。詳細に関しては、Altova のバーコード拡張関数を参照してください。与えられる文字列は、使用するベースロケーションを与える絶対 URL である必要があります。

COM and .NET

署名: `JavaBarcodeExtensionLocation(string url)`

Java

署名: `public void setJavaBarcodeExtensionLocation(string url)`

5.4.1.5.2.9 JavaExtensionsEnabled

Java 拡張子を有効化または無効化します。`true` の値は、Java 拡張子を有効化します `false` の値は無効化にします。デフォルト値はtrue です。

COM and .NET

署名: `JavaExtensionsEnabled(boolean enable)`

Java

署名: `public void setJavaExtensionsEnabled(boolean enable)`

5.4.1.5.2.10 KeepFormatting

元のドキュメントのフォーマットを（可能な限り）保持するかを指定します。`true` の値はフォーマットを継続し、`false` はフォーマットを継続しません。デフォルトの値は`true` です。

COM and .NET

署名: `KeepFormatting(boolean keep)`

Java

署名: `public void setKeepFormatting(boolean keep)`

5.4.1.5.2.11 LastErrorMessage

RaptorXML エンジンからの最後のエラーメッセージを文字列として取得します。

COM and .NET

署名: `string LastErrorMessage()`

Java

署名: `public string getLastErrorMessage()`

5.4.1.5.2.12　LoadXMLWithPSVI

Post Schema Validation Infoset（PSVI）（検証済み XML ノードのスキーマ検証後の infoset）のロードおよび使用のを有効化または無効化します。PSVI がロードされると、スキーマから取得された情報を、ドキュメント内のデータを修飾するために使用することができます。`true` の値は、PSVI ロードを有効化します。`false` の値は無効化します。デフォルト値は `true` です。

COM and .NET

署名: `LoadXMLWithPSVI(boolean enable)`

Java

署名: `public void setLoadXMLWithPSVI(boolean enable)`

5.4.1.5.2.13　MainOutput

最後に実行されたジョブの主要な出力を返します。

COM and .NET

署名: `string MainOutput()`

Java

署名: `public string getMainOutput()`

5.4.1.5.2.14　OutputEncoding

結果ドキュメントのエンコードを設定します。`UTF-8`、`UTF-16`、`US-ASCII`、`ISO-8859-1` などの公式の IANA エンコード名を文字列として使用します。

COM and .NET

署名: `OutputEncoding(string encoding)`

Java

署名: `public void setOutputEncoding(string encoding)`

5.4.1.5.2.15 OutputIndent

出力ドキュメントのインデントを有効化または無効化します。`true` の値は、インデントを有効化します。`false` の値はサポートを無効化にします。

COM and .NET

署名: `OutputIndent(boolean outputIndent)`

Java

署名: `public void setOutputIndent(boolean outputIndent)`

5.4.1.5.2.16 OutputMethod

出力ドキュメントのシリアル化を指定します。有効な値: `xml|xhtml|html|text`。デフォルトの値は `xml` です。

COM and .NET

署名: `OutputMethod(string format)`

Java

署名: `public void setOutputMethod(string format)`

5.4.1.5.2.17 OutputOmitXMLDeclaraton

結果ドキュメント内のXML 宣言の包含を有効化/無効化します。`true` の値は、宣言を無視します。`false` は、宣言を含みます。デフォルトの値は `false` です。

COM and .NET

署名: `OutputOmitXMLDeclaration(boolean omitDeclaration)`

Java

署名: **public void setOutputOmitXMLDeclaration(boolean omitDeclaration)**

5.4.1.5.2.18　UpdatedXMLWriteMode

XML ファイルへの更新の方法が扱われるか指定します。プロパティの値は ENUMXQueryUpdatedXML 299 リテラルです。

COM and .NET

署名: **UpdateXMLWriteMode(ENUMXQueryUpdatedXML 299 updateMode)**

Java

署名: **public void setUpdateXMLWriteMode(ENUMXQueryUpdatedXML 299 updateMode)**

5.4.1.5.2.19　XincludeSupport

XInclude 要素の使用を有効化または無効化します。true の値は、XInclude へのサポートを有効化します。false の値は無効化します。デフォルト値は false です。

COM and .NET

署名: **XincludeSupport(boolean xinclude)**

Java

署名: **public void setXincludeSupport(boolean xinclude)**

5.4.1.5.2.20　XMLValidationErrorsAsWarnings

XML 検証エラーを警告として扱うことを有効化します。ブール値の true または false をとります。

COM and .NET

署名: **XMLValidationErrorsAsWarnings(boolean enable)**

Java

署名: `public void setXMLValidationErrorsAsWarnings(boolean enable)`

5.4.1.5.2.21 XMLValidationMode

有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode 298 の列挙リテラルである XML 検証モードを設定します。

COM and .NET

署名: `XMLValidationMode(ENUMXMLValidationMode 298 valMode)`

Java

署名: `public void setXMLValidationMode(ENUMXMLValidationMode 298 valMode)`

5.4.1.5.2.22 XQueryFileName

使用する XQuery ファイルを指定します。提供される文字列は使用する XQuery ファイルの場所を提供する絶対 URL である必要があります。

COM and .NET

署名: `XQueryFileName(string fileurl)`

Java

署名: `public void setXQueryFileName(string fileurl)`

5.4.1.5.2.23 XQueryFromText

テキスト文字列として、使用される XQuery ドキュメントのコンテンツを提供します。

COM and .NET

署名: `XQueryFromText(string xqtext)`

Java

署名: `public void setXQueryFromText(string xqtext)`

5.4.1.5.2.24　XQueryUpdateVersion

使用するXQuery Update バージョンを指定します。プロパティの値は ENUMXQueryUpdateVersion 300 リテラルです。

COM and .NET

署名: **XQueryUpdateVersion(ENUMXQueryUpdateVersion 300 version)**

Java

署名: `public void` **setXQueryUpdateVersion(ENUMXQueryUpdateVersion 300 version)**

5.4.1.5.2.25　XSDVersion

検証されるXMLドキュメントに対して、XML スキーマ バージョンを指定します。ENUMXSDVersion 301 は、の列挙リテラルです。

COM and .NET

署名: **XSDVersion(ENUMXSDVersion 301 version)**

Java

署名: `public void` **setXSDVersion(ENUMXSDVersion 301 version)**

5.4.1.6　XSLT

IXSLT/XSLT インターフェイス/クラスは、XSLT 変換とXSLT に関連したドキュメントの検証を行うメソッドを適用します。外部パラメーターを介して変更のためにデータを提供することもできます。

COM/.NET API 内のインターフェイスの名前は、Java API 内のクラスのインターフェイスの名前と異なることにご注意してください。:

- COM/.NET 内: **IXSLT**
- Java 内: **XSLT**

5.4.1.6.1　メソッド

IXSLT インターフェイス(COM/.NET) と**XSLT** クラス(Java) のメソッドはこのセクションで説明されています。

5.4.1.6.1.1 AddExternalParameter

新規の外部パラメーターの名前と値を追加します。各外部パラメーターとその値は、メソッドへの個別の呼び出し内で指定されます。パラメーターはXSLTドキュメント内で宣言されている必要があります。パラメーター値はXPath 式であるため、パラメーターの値を一重引用符で囲む必要があります。 `name` パラメーターはQName である変数の値を文字列としてホールドします。`value` パラメーターは変数の値を文字列としてホールドします。

COM and .NET

署名: **`AddExternalParameter(string name, string value)`**

Java

署名: **`public void addExternalParameter(string name, string value)`**

5.4.1.6.1.2 ClearExternalParameterList

AddExternalParameter 282 メソッドにより作成された外部パラメーターリストをクリアします。

COM and .NET

署名: **`ClearExternalParameterList()`**

Java

署名: **`public void clearExternalParameterList()`**

5.4.1.6.1.3 Execute

EngineVersion 285 プロパティ内で名前の付けられたXSLT 仕様に従いXSLT 変換を実行し、`outputFile` パラメータ内で名前が付けられている通り出力ファイルの結果を保存します。エラーが発生すると、RaptorXMLException 250 が挙げられます。`LastErrorMessage` プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: `boolean` **`Execute(`**`string` **`outputFile)`**

Java

署名: **`public`** `boolean` **`execute(`**`string` **`outputFile)`**

5.4.1.6.1.4 ExecuteAndGetResultAsString

EngineVersion[285] プロパティ内で名前の付けられたXSLT 仕様に従いXSLT 変換を実行し、結果を文字列として返します。このメソッドはチャートやセカンダリの結果などの追加結果ファイルを生成しません。また、.docx OOXML ファイルなどのバイナリの結果をホールドしません。追加の出力ファイルが必要な場合、Execute[282] メソッドを使用します。エラーが発生すると、RaptorXMLException[250] が挙げられます。LastErrorMessage プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: `string ExecuteAndGetResultAsString()`

Java

署名: `public string executeAndGetResultAsString()`

5.4.1.6.1.5 ExecuteAndGetResultAsStringWithBaseOutputURI

EngineVersion[285] プロパティ内で名前の付けられたXSLT 仕様に従いXSLT変換を実行し、ベースURI により定義されているロケーションで文字列として結果を返します。baseURI パラメーターはURI を提供する文字列です。このメソッドはチャートやセカンダリの結果などの追加結果ファイルを生成しません。また、.docx OOXML ファイルなどのバイナリの結果をホールドしません。追加の出力ファイルが必要な場合、Execute[282] メソッドを使用します。エラーが発生すると、RaptorXMLException[250] が挙げられます。LastErrorMessage プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: `string ExecuteAndGetResultAsStringWithBaseOutputURI(string baseURI)`

Java

署名: `public string ExecuteAndGetResultAsStringWithBaseOutputURI(string baseURI)`

5.4.1.6.1.6 IsValid

EngineVersion[285] プロパティ内で名前の付けられているXSLT 仕様に従いXSLT ドキュメントの検証の結果を返します。成功した場合結果は、true を返し、失敗した場合は false を返します。エラーが発生すると、RaptorXMLException[250] が挙げられます。LastErrorMessage プロパティを使用して追加情報にアクセスします。

COM and .NET

署名: `boolean IsValid()`

Java

署名: `public boolean isValid()`

5.4.1.6.2 プロパティ

IXSLT インターフェイス（COM/.NET）と**XSLT** クラス（Java）のプロパティはこのセクションで説明されています。

5.4.1.6.2.1 AdditionalOutputs

最後に実行されたジョブの追加の出力を返します。

COM and .NET

署名: `string AdditionalOutputs()`

Java

署名: `public string getAdditionalOutputs()`

5.4.1.6.2.2 ChartExtensionsEnabled

Altova チャート 拡張関数を有効化または無効化します。`true` の値は、チャート 拡張子を有効化します。`false` は無効化にします。デフォルト値は `true` です。

COM and .NET

署名: `ChartExtensionsEnabled(boolean enable)`

Java

署名: `public void setChartExtensionsEnabled(boolean enable)`

5.4.1.6.2.3 DotNetExtensionsEnabled

Visual Studio .NET 拡張関数を有効化または無効化します。 `true` の値は、.NET 拡張子を有効化します。 `false` の値は無効化にします。デフォルト値は `true` です。

COM and .NET

署名: **DotNetExtensionsEnabled**(boolean **enable**)

Java

署名: public void **setDotNetExtensionsEnabled**(boolean **enable**)

5.4.1.6.2.4 EngineVersion

使用するXSLT バージョンを指定します。プロパティの値は ENUMXSLTVersion 302 リテラルです。

COM and .NET

署名: **EngineVersion**(ENUMXSLTVersion 302 **version**)

Java

署名: public void **setEngineVersion**(ENUMXSLTVersion 302 **version**)

5.4.1.6.2.5 IndentCharacters

出力内でインデントとして使用される文字列を設定します。

COM and .NET

署名: **IndentCharacters**(string **indentChars**)

Java

署名: public void **setIndentCharacters**(string **indentChars**)

5.4.1.6.2.6 InitialTemplateMode

XSLT 処理のための初期のモードを設定します。処理する文字列に与えられた値に等しいモードの値を持つテンプレート。

COM and .NET

署名: **InitialTemplateMode**(string **mode**)

Java

署名: `public void setInitialTemplateMode(string mode)`

5.4.1.6.2.7 InputXMLFileName

処理される XML ドキュメントの場所を URL として設定します。与えられる文字列は、XML ファイルの正確な場所を与える絶対 URL である必要があります。

COM and .NET

署名: `InputXMLFileName(string url)`

Java

署名: `public void setInputXMLFileName(string url)`

5.4.1.6.2.8 InputXMLFromText

処理する XML ドキュメントのコンテンツを提供します。提供される文字列は処理する XML ドキュメントのコンテンツです。

COM and .NET

署名: `InputXMLFromText(string xml)`

Java

署名: `public void setInputXMLFromText(string xml)`

5.4.1.6.2.9 JavaBarcodeExtensionLocation

バーコード拡張ファイルの場所を指定します。詳細に関しては、Altova のバーコード拡張関数を参照してください。与えられる文字列は、使用するベースロケーションを与える絶対 URL である必要があります。

COM and .NET

署名: `JavaBarcodeExtensionLocation(string url)`

Java

署名: `public void setJavaBarcodeExtensionLocation(string url)`

5.4.1.6.2.10　JavaExtensionsEnabled

Java 拡張子を有効化または無効化します。`true` の値は、Java 拡張子を有効化します `false` の値は無効化にします。デフォルト値はtrue です。

COM and .NET

署名: `JavaExtensionsEnabled(boolean enable)`

Java

署名: `public void setJavaExtensionsEnabled(boolean enable)`

5.4.1.6.2.11　LastErrorMessage

RaptorXML エンジンからの最後のエラーメッセージを文字列として取得します。

COM and .NET

署名: `string LastErrorMessage()`

Java

署名: `public string getLastErrorMessage()`

5.4.1.6.2.12　LoadXMLWithPSVI

Post Schema Validation Infoset（PSVI）（検証済み XML ノードのスキーマ検証後のinfoset）のロードおよび使用のを有効化または無効化します。PSVI がロードされると、スキーマから取得された情報を、ドキュメント内のデータを修飾するために使用することができます。`true` の値は、PSVI ロードを有効化します。`false` の値は無効化します。デフォルト値は`true` です。

COM and .NET

署名: `LoadXMLWithPSVI(boolean enable)`

Java

署名: `public void setLoadXMLWithPSVI(boolean enable)`

5.4.1.6.2.13 MainOutput

最後に実行されたジョブの主要な出力を返します。

COM and .NET

署名: `string MainOutput()`

Java

署名: `public string getMainOutput()`

5.4.1.6.2.14 NamedTemplateEntryPoint

変換のためのエントリポイントとして使用される名前を持つテンプレートの名前を文字列として指定します。

COM and .NET

署名: `NamedTemplateEntryPoint(string template)`

Java

署名: `public void setNamedTemplateEntryPoint(string template)`

5.4.1.6.2.15 SchemaImports

`xs:import` 要素の属性の値に従い、スキーマインポートがどのように扱われるか指定します。扱いは選択された `ENUMSchemaImports` リテラルにより指定されます。

COM and .NET

署名: `SchemaImports(ENUMSchemaImports` [294] `importOption)`

Java

署名: **public** void **setSchemaImports(**ENUMSchemaImports[294] **importOption)**

5.4.1.6.2.16 SchemalocationHints

スキーマの検索に使用されるメカニズムを指定します。メカニズムは選択された ENUMSchemaImports リテラルにより指定されます。

COM and .NET

署名: **SchemalocationHints(**ENUMLoadSchemalocation[293] **hint)**

Java

署名: **public** void **setSchemalocationHints(**ENUMLoadSchemalocation[293] **hint)**

5.4.1.6.2.17 SchemaMapping

スキーマの検索内で使用されるマッピングを設定します。マッピングは選択された ENUMSchemaMapping リテラルにより指定されます。

COM and .NET

署名: **SchemaMapping(**ENUMSchemaMapping[295] **mappingOption)**

Java

署名: **public** void **setSchemaMapping(**ENUMSchemaMapping[295] **mappingOption)**

5.4.1.6.2.18 StreamingSerialization

ストリームシリアル化を有効化します。ストリーミングモードでは、メモリ内に保管されているデータは最小化され、更に処理されます。true の値は、ストリームシリアル化を有効化します。false の値はサポートを無効化にします。

COM and .NET

署名: **StreamingSerialization(**boolean **enable)**

Java

署名: **public** void **setStreamingSerialization(**boolean **enable)**

5.4.1.6.2.19 XincludeSupport

`XInclude` 要素の使用を有効化または無効化します。true の値は、XInclude へのサポートを有効化します。false の値は無効化します。デフォルト値はfalse です。

COM and .NET

署名: **XincludeSupport**(boolean **xinclude**)

Java

署名: **public** void **setXincludeSupport**(boolean **xinclude**)

5.4.1.6.2.20 XMLValidationErrorsAsWarnings

XML 検証エラーを警告として扱うことを有効化します。ブール値の `true` または `false` をとります。

COM and .NET

署名: **XMLValidationErrorsAsWarnings**(boolean **enable**)

Java

署名: **public** void **setXMLValidationErrorsAsWarnings**(boolean **enable**)

5.4.1.6.2.21 XMLValidationMode

有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode 298 の列挙リテラルである XML 検証モードを設定します。

COM and .NET

署名: **XMLValidationMode**(ENUMXMLValidationMode 298 **valMode**)

Java

署名: **public** void **setXMLValidationMode**(ENUMXMLValidationMode 298 **valMode**)

5.4.1.6.2.22 XSDVersion

検証されるXMLドキュメントに対して、XMLスキーマバージョンを指定します。ENUMXSDVersion 301 は、の列挙リテラルです。

COM and .NET

署名: **XSDVersion(ENUMXSDVersion 301 version)**

Java

署名: `public void setXSDVersion(ENUMXSDVersion 301 version)`

5.4.1.6.2.23 XSLFileName

使用するXSLTファイルを指定します。提供される文字列は使用するXSLTファイルの場所を提供する絶対URLである必要があります。

COM and .NET

署名: **XSLFileName(string fileurl)**

Java

署名: `public void setXSLFileName(string fileurl)`

5.4.1.6.2.24 XSLFromText

使用されるXSLTドキュメントのコンテンツをテキスト文字列として提供します。

COM and .NET

署名: **XSLFromText(string xsltext)**

Java

署名: `public void setXSLFromText(string xsltext)`

5.4.2 列挙

COM/.NET and Java サーバーAPI の列挙がこのセクションで説明されています。各説明文には、列挙を使用するメソッド、または、プロパティへのリンクが含まれています。

- ENUMAssessmentMode 292
- ENUMErrorFormat 293
- ENUMLoadSchemalocation 293
- ENUMSchemaImports 294
- ENUMSchemaMapping 295
- ENUMValidationType 296
- ENUMWellformedCheckType 297
- ENUMXMLValidationMode 298
- ENUMXQueryUpdatedXML 299
- ENUMXQueryUpdateVersion 300
- ENUMXQueryVersion 300
- ENUMXSDVersion 301
- ENUMXSLTVersion 302

5.4.2.1 ENUMAssessmentMode

XML バリデータの評価モードを strict または lax に定義します:

- **eAssessmentModeStrict**: スキーマの有効性評価モードを `Strict` に設定します。これはデフォルトの値です。
- **eAssessmentModeLax**: スキーマの有効性評価モードを `Lax` に設定します。

COM and .NET

eAssessmentModeStrict	= 0
eAssessmentModeLax	= 1

使用例

インターフェイス	*プロパティ*
IXMLValidator 258	AssessmentMode 260

Java

```
public enum ENUMAssessmentMode {
       eAssessmentModeLax
       eAssessmentModeStrict    }
```

使用例

クラス	*メソッド*

XMLValidator 258	setAssessmentMode 260

5.4.2.2 ENUMErrorFormat

エラー出力のフォーマットを指定します:

- **eFormatText**: エラー出力フォーマットを Text に設定します。デフォルトの値です。
- **eFormatShortXML**: エラー出力フォーマットを ShortXML に設定します。このフォーマットは、LongXML フォーマットの省略されたフォームです。
- **eFormatLongXML**: エラー出力フォーマットを LongXML に設定します。このフォーマットは、3 つすべての出力フォーマットの情報のほぼすべてを提供します。

COM and .NET

eFormatText	= 0
eFormatShortXML	= 1
eFormatLongXML	= 2

使用例

インターフェイス	*プロパティ*
IServer 241	ErrorFormat 245

Java

```
public enum ENUMErrorFormat {
        eFormatText
        eFormatShortXML
        eFormatLongXML }
```

使用例

クラス	*メソッド*
RaptorXMLFactory 241	setErrorFormat 245

5.4.2.3 ENUMLoadSchemalocation

スキーマロケーションがどのように決定されるかを示します。XML インスタンスドキュメントのスキーマロケーション属性をベースに選択されます。この属性は xsi:schemaLocation または xsi:noNamespaceSchemaLocation であることができます。

- **eSHLoadBySchemalocation** はXML インスタンスドキュメント内のスキーマロケーションのURL を使用します。列挙リテラルはデフォルトの値です。

- **eSHLoadByNamespace** は xsi:schemaLocation の名前空間の部分を使用し、xsi:noNamespaceSchemaLocation の場合は空の文字列を使用し、カタログマッピングを介してスキーマをロケートします。
- **eSHLoadCombiningBoth**: 名前空間またはURL のどちらかがカタログマッピングを有する場合、 そのカタログ マッピングが使用されます。双方がカタログマッピングを有する場合、ENUMSchemaMapping 295 の値がどちらのマッピングを使用するか決定します。名前空間とURL の双方がカタログマッピングを有しない場合、URL が使用されます。
- **eSHLoadIgnore**: xsi:schemaLocation とxsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

COM and .NET

eSHLoadBySchemalocation	= 0
eSHLoadByNamespace	= 1
eSHLoadCombiningBoth	= 2
eSHLoadIgnore	= 3

使用例

インターフェイス	*プロパティ*
IXMLValidator 258	SchemalocationHints 268
IXSLT 281	SchemalocationHints 289

Java

```
public enum ENUMLoadSchemalocation {
       eSHLoadBySchemalocation
       eSHLoadByNamespace
       eSHLoadCombiningBoth
       eSHLoadIgnore    }
```

使用例

クラス	*メソッド*
XMLValidator 258	setSchemalocationHints 268
XSLT 281	setSchemalocationHints 289

5.4.2.4 ENUMSchemaImports

xs:import 要素の振る舞いを定義する列挙リテラルを含みます。xs:import 要素はnamespace とschemaLocation 属性を有し、双方の属性は任意です。

- **eSILoadBySchemalocation** はschemaLocation 属性の値をスキーマを検索する際に使用します。namespace 属性が存在する場合、名前空間はインポートされます（ライセンスが与えられます）。

- **eSILoadPreferringSchemalocation**: schemaLocation が存在し、使用されている場合、からログまたは考慮されます。schemaLocation が存在しない場合、namespace 属性の値がカタログマッピングを介して使用されます。列挙リテラルはデフォルトの値です。
- **eSILoadByNamespace** は namespace 属性の値を使用し、カタログマッピングを介して、スキーマをロケートします。
- **eSILoadCombiningBoth**: namespace URL または schemaLocation URL にカタログマッピングが存在する場合、カタログマッピングが使用されます。からログマッピングが存在する場合、ENUMSchemaMapping 295 は、どのマッピングが使用されるかを決定します。namespace または schemaLocation URL にカタログマッピングが存在しない場合、schemaLocation URL が使用されます。
- **eSILicenseNamespaceOnly**: 名前空間がインポートされます。スキーマドキュメントはインポートされません。

COM and .NET

eSILoadBySchemalocation	= 0
eSILoadPreferringSchemalocation	= 1
eSILoadByNamespace	= 2
eSICombiningBoth	= 3
eSILicenseNamespaceOnly	= 4

使用例

インターフェイス	*プロパティ*
IXMLValidator 258	SchemaImports 267
IXSLT 281	SchemaImports 288

Java

```
public enum ENUMSchemaImports {
       eSILoadBySchemalocation
       eSILoadPreferringSchemalocation
       eSILoadByNamespace
       eSILoadCombiningBoth
       eSILicenseNamespaceOnly    }
```

使用例

クラス	*メソッド*
XMLValidator 258	setSchemaImports 267
XSLT 281	setSchemaImports 288

5.4.2.5 ENUMSchemaMapping

次のどちらのカタログマッピングが優先されるか定義する列挙リテラルを含みます: 名前空間またはスキーマの場所 URL。列挙は ENUMLoadSchemalocation 293 と ENUMSchemaImports 294 のあいまいさを除去するために役に立ちます。

- **eSMPreferNamespace**: 名前空間を選択します。
- **eSMPreferSchemalocation**: スキーマロケーションを選択します。これはデフォルトの値です。

COM and .NET

eSMPreferSchemalocation	= 0
eSMPreferNamespace	= 1

使用例

インターフェイス	*プロパティ*
IXMLValidator 258	SchemaMapping 268
IXSLT 281	SchemaMapping 289

Java

```
public enum ENUMSchemaMapping {
       eSMPreferSchemalocation
       eSMPreferNamespace    }
```

使用例

クラス	*メソッド*
IXMLValidator 258	setSchemaMapping 268
IXSLT 281	setSchemaMapping 289

5.4.2.6 ENUMValidationType

XML ドキュメントの場合 DTD または XSD に対して実行する検証を指定します。

- **eValidateAny**: ドキュメントの型（例えば、XML または XSD）が検知され、そのドキュメントの型のために検証が自動的に設定されます。
- **eValidateXMLWithDTD**: DTD に対しての XML ドキュメントの検証を指定します。
- **eValidateXMLWithXSD**: XSD（XML スキーマ）に対しての XML ドキュメントの検証を指定します。
- **eValidateDTD**: DTD ドキュメントの検証を指定します。
- **eValidateXSD**: XSD（W 3C XMLSchema）ドキュメントの検証を指定します。
- **eValidateJSON**: JSON インスタンスドキュメントの検証を指定します。
- **eValidateJSONSchema**: JSON スキーマドキュメントの検証を JSON Schema v4 に従い指定します。
- **eValidateAvro**: Avro バイナリファイルの検証を指定します。バイナリファイル内の Avro データはバイナリファイル内に含まれる Avro スキーマに対して検証されます。
- **eValidateAvroSchema**: Avro スキーマの検証を Avro スキーマ仕様に従い指定します。
- **eValidateAvroJSON**: JSON シリアル化内の Avro データドキュメントの検証を Avro スキーマに従い指定します。

COM and .NET

eValidateAny	= 0
eValidateXMLWithDTD	= 1
eValidateXMLWithXSD	= 2
eValidateDTD	= 3
eValidateXSD	= 4
eValidateJSON	= 5
eValidateJSONSchema	= 6
eValidateAvro	= 7
eValidateAvroSchema	= 8
eValidateAvroJSON	= 9

使用例

インターフェイス	*メソッド*
IXMLValidator 258	isValid 260

Java

```
public enum ENUMValidationType {
       eValidateAny
       eValidateXMLWithDTD
       eValidateXMLWithXSD
       eValidateDTD
       eValidateXSD
       eValidateJSON
       eValidateJSONSchema
       eValidateAvro
       eValidateAvroSchema
       eValidateAvroJSON }
```

使用例

クラス	*メソッド*
XMLValidator 258	isValid 260

5.4.2.7 ENUMWellformedCheckType

実行する整形式の型を指定します（XML、DTD、またはJSON）。

- **eWellformedAny**: ドキュメントの型が検出され、チェックの型が自動的に設定されます。
- **eWellformedXML**: XML ドキュメントの整形式をチェックします。
- **eWellformedDTD**: DTD ドキュメントの整形式をチェックします。
- **eWellformedJSON**: JSON ドキュメントの整形式をチェックします。

COM and .NET

eWellFormedAny	= 0
eWellFormedXML	= 1
eWellFormedDTD	= 2
eWellFormedJSON	= 3

使用例

インターフェイス	*メソッド*
IXMLValidator 258	isWellFormed 260

Java

```
public enum ENUMWellformedCheckType {
       eWellformedAny
       eWellformedXML
       eWellformedDTD
       eWellformedJSON    }
```

使用例

クラス	*メソッド*
XMLValidator 258	isWellFormed 260

5.4.2.8 ENUMXMLValidationMode

XML 検証の型を指定します（検証、または、整形式のチェック）.

- **eProcessingModeWF**: XML 処理モードを `wellformed` に設定します。これはデフォルトの値です。
- **eProcessingModeValid**: XML 処理モードを `validation` に設定します。
- **eProcessingModeID**: 内部使用のみで、使用することはできません。

COM and .NET

eXMLValidationModeWF	= 0
eXMLValidationModeID	= 1
eXMLValidationModeValid	= 2

使用例

インターフェイス	*プロパティ*

IXMLValidator 258	XMLValidationMode 269
IXQuery 270	XMLValidationMode 280
IXSLT 281	XMLValidationMode 290

Java

```
public enum ENUMXMLValidationMode {
       eProcessingModeValid
       eProcessingModeWF
       eProcessingModeID  }
```

使用例

クラス	*メソッド*
XMLValidator 258	setXMLValidationMode 269
XQuery 270	setXMLValidationMode 280
XSLT 281	setXMLValidationMode 290

5.4.2.9 ENUMXQueryUpdatedXML

XQuery の更新の処理の方法を指定します。

- **eUpdatedDiscard**: アップデートは破棄され、ファイルに書き込まれません。
- **eUpdatedWriteback**: アップデートは入力ファイルに書き込まれ、(set)InputXMLFileName 275 と共に指定されます。
- **eUpdatedAsMainResult**: ExecuteUpdate 252 の outputFile パラメータにより指定された場所に書き込まれます。

COM and .NET

eUpdatedDiscard	= 1
eUpdatedWriteback	= 2
eUpdatedAsMainResult	= 3

使用例

インターフェイス	*プロパティ*
IXQuery 270	UpdatedXMLWriteMode 279

Java

```
public enum ENUMXQueryUpdatedXML {
       eUpdatedDiscard
       eUpdatedWriteback
       eeUpdatedAsMainResult }
```

使用例

クラス	*メソッド*
XQuery 270	setUpdatedXMLWriteMode 279

5.4.2.10 ENUMXQueryUpdateVersion

処理のために使用される XQuery Update バージョンを設定します。

- **eXQUpdateVersion10**: XQuery Update バージョンを XQuery Update 1.0 に設定します。

メモ COM/.NET リテラルとは異なる方法で Java 列挙リテラルの名前が付けられますです。*以下を参照してください。*

COM and .NET

eXQUpdateVersion10	= 1

使用例

インターフェイス	*プロパティ*
IXQuery 270	XQueryUpdateVersion 281

Java

```
public enum ENUMXQueryUpdateVersion {
       eVersion10        }
```

使用例

クラス	*メソッド*
XQuery 270	setXQueryUpdateVersion 281

.

5.4.2.11 ENUMXQueryVersion

処理のために使用される XQuery バージョンを設定します（実行または検証）。

- **eXQVersion10**: XQuery バージョンを XQuery 1.0 に設定します。
- **eXQVersion30**: XQuery バージョンを XQuery 3.0 に設定します。デフォルトの値です。
- **eXQVersion31**: XQuery バージョンを XQuery 3.1 に設定します。

メモ Java 列挙リテラルは、COM/.NET リテラルとは異なる方法で名前が付けられます。以下を参照してください。

COM and .NET

eXQVersion10	= 1
eXQVersion30	= 3
eXQVersion31	= 31

使用例

インターフェイス	*プロパティ*
IXQuery 270	EngineVersion 274

Java

```
public enum ENUMXQueryVersion {
       eVersion10
       eVersion30
       eVersion31 }
```

使用例

クラス	*メソッド*
XQuery 270	setEngineVersion 274

5.4.2.12 ENUMXSDVersion

検証に使用される XML スキーマバージョンを指定します。

- **eXSDVersionAuto**: 検証のための XML スキーマバージョンを Auto-detect に設定します。XSD ドキュメントを解析した後、XSD のバージョンは自動的に検出されます。XSD ドキュメントの `vc:minVersion` 属性が 1.1 の値を持つ場合、ドキュメントは XSD 1.1 とみなされます。 属性が他の値を持つ場合、または存在しない場合、ドキュメントは XSD 1.0 とみなされます。
- **eXSDVersion10**: 検証のための XML スキーマバージョンを XML スキーマ 1.0 に設定します。
- **eXSDVersion11**: 検証のための XML スキーマバージョンを XML スキーマ 1.1 に設定します。

COM and .NET

eXSDVersionAuto	= 0
eXSDVersion10	= 1
eXSDVersion11	= 2

使用例

インターフェイス	*プロパティ*
IXMLValidator 258	XSDVersion 270
IXQuery 270	XSDVersion 281
IXSLT 281	XSDVersion 291

Java

```
public enum ENUMXSDVersion {
       eXSDVersionAuto
       eXSDVersion10
       eXSDVersion11    }
```

使用例

クラス	メソッド
XMLValidator 258	setXSDVersion 270
XQuery 270	setXSDVersion 281
XSLT 281	setXSDVersion 291

5.4.2.13 ENUMXSLTVersion

処理(検証、または、XSLT 変換)に使用されるXSLT バージョンを設定します。

- **eVersion10**: XSLT バージョンをXSLT 1.0 に設定します。
- **eVersion20**: XSLT バージョンをXSLT 2.0 に設定します。
- **eVersion30**: XSLT バージョンをXSLT 3.0 に設定します。

COM and .NET

eVersion10	= 1
eVersion20	= 2
eVersion30	= 3

使用例

インターフェイス	プロパティ
IXSLT 281	EngineVersion 285

Java

```
public enum ENUMXSLTVersion {
       eVersion10
       eVersion20
       eVersion30 }
```

使用例

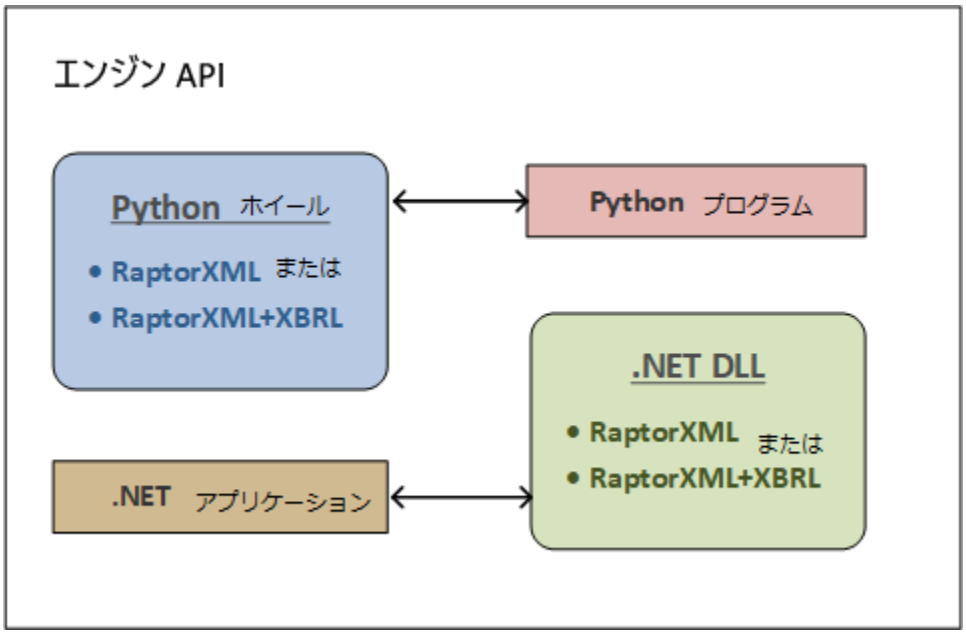
クラス	メソッド
XSLT 281	setEngineVersion 285

6 エンジン API: Python と .NET

RaptorXML Server には2つのエンジン API が存在します:

- Python エンジン API: `raptorxml<versiondetails>.whl` である Python ホイールファイル(`.whl`)
- .NET エンジン API: `raptorxmlapi.dll` である .NET DLL ファイル(`.dll`)

2つのエンジン API はそれぞれ RaptorXML Server を含み、マシンにインストールされている RaptorXML Server とは区別されています(下の図を参照してください)。各パッケージは、Python モジュールとしてインポートされる、または、カスタム .NET アプリケーションに統合される前にユーザーのマシンにインストールされている必要があります。全ての処理はユーザーのマシン上で処理されるため、Python と .NET エンジン API は有効な XML と XBRL インスタンス、XSD スキーマと XBRL タクソノミのデータモデルのアクセス情報を提供します。API は数行のコードで、XBRL インスタンスのコンテンツを反復、または、タクソノミからの特定の情報を抽出するためのメソッドの豊富なセットを公開します。



エンジン API に関する以下の点に注意してください:

- RaptorXML Server のインストール後、両方のエンジン API は RaptorXML Server インストールフォルダーの `bin` フォルダー内でロケートすることができます。
- エンジン API は、API 内の多種のオブジェクトを使用して更に高度な処理を提供します。
- エンジン API を使用するには、RaptorXML Server のライセンス供与済みのバージョンが、Python プログラム、または、.NET アプリケーションが実行されるマシン上にインストールされている必要があります(下の使用法を参照してください)。

使用法

Python プログラム、または、.NET アプリケーションを次のように作成することができます:

Python プログラム

Python プログラムは Python API オブジェクト 306 を使用して RaptorXML 機能にアクセスすることができます(ここ 306 を参照してください)。Python プログラムが実行される個所で、Python ホイールが使用されると Python 環境内にインストールされている RaptorXML ライブラリが使用されます。Python ホイールが Python バージョン 3.7 と互換性を有することを確認してください。

.NET アプリケーション
.NET アプリケーションは.NET API オブジェクト 314 を使用して、RaptorXML 機能にアクセスすることができます（ここ 314 を参照してください）。.NET アプリケーションが実行される個所で、.NET API DLL 内に含まれる RaptorXML を使用します。

ライセンス

API を使用するために、ライセンス供与済みの RaptorXML Server のバージョンが Python プログラム、または、.NET アプリケーションが実行されるマシンにインストールされる必要があります（下の使用方法を参照してください）。詳細に関してはライセンス 305 のセクションを参照してください。

(C) 2015-2021 Altova GmbH

6.1 ライセンス

API パッケージをクライアントマシン上で実行するには、RaptorXML Server クライアントとしてマシンにライセンスが供与されている必要があります。ライセンスの供与は以下の2つのステップから構成されています:

1. RaptorXML Server クライアントとしてマシンを Altova LicenseServer に登録します。
2. LicenseServer からマシンに RaptorXML Server ライセンスを割り当てます。

与えられたマシンから API パッケージを使用を計画する場合、以下の2つのシチュエーションが発生する可能性があります:

- クライアント マシンが既に、ライセンスされている RaptorXML Server のインストールを作動している場合、API パッケージは、追加のステップを行う必要がなく、作動することができます。これは、マシンが RaptorXML Server を作動するためにライセンスを与えられているからです。この結果、このマシン上の API パッケージの使用は、そのマシン上の RaptorXML Server に割り当てられているライセンスによりカバーされています。
- RaptorXML Server がクライアントマシンにインストールされていない場合、そのマシンに RaptorXML Server をインストールする必要があります。この場合、マシンを RaptorXML Server クライアント として登録し、RaptorXML Server ライセンスを与えます。方法は以下で説明されるとおりです。

(RaptorXML Server がインストールされていない)マシンを RaptorXML Server クライアントとして登録するには、アプリケーションの **bin** フォルダー内にあるコマンドライン アプリケーション **registerlicense.exe** を使用します:

Windows	`Program Files\Altova\RaptorXMLServer2021\bin`
Linux	`/opt/Altova/RaptorXMLServer2021/bin`
Mac	`/usr/local/Altova/RaptorXMLServer2021/bin`

コマンドラインでコマンドを実行します:

```
registerlicense <LicenseServer>
```

<LicenseServer> が LicenseServer マシンの IP アドレス、または、ホスト名である箇所で、

このコマンドは、マシンを RaptorXML Server クライアントとして Altova LicenseServer に登録します。RaptorXML Server ライセンスをマシンに割り当てる方法、およびライセンスの供与に関する情報については、次を参照してください: Altova LicenseServer ドキュメンテーション。

Linux にデプロイ

Python ホイール パッケージを使用して、**registerlicense** アプリケーションにデプロイする場合、下にリストされる共有されたライブラリは、兄弟 lib ディレクトリ内に存在する必要があります。共有されているライブラリは Raptor インストールフォルダーからコピーされることができます:

```
/opt/Altova/RaptorXMLServerRaptorXMLServer2021/lib
```

- `libcrypto.so.1.0.0`
- `libssl.so.1.0.0`
- `libstdc++.so.6`
- `libtbb.so.2`

6.2 Python API

RaptorXML Python API によりXMLドキュメントとXML スキーマドキュメント内でのデータへのアクセス、および、Python スクリプト内での操作を有効化することができます。Python API の一般的な使用ケースは以下の通りです:

- カスタムの検証ルールとエラーメッセージの実装
- XMLドキュメントからコンテンツをデータベースにエクスポート
- XMLドキュメントからコンテンツをカスタムデータ書式にエクスポート
- Python シェル、または、Jupyter ノートブック（http://jupyter.org/）内でXMLドキュメントのデータモデルをインタラクティブにナビゲートしクエリ

RaptorXML Server のPython インターフェイスは、Python API を介して、XML およびXSD のためのXMLドキュメント およびXML スキーマドキュメント 内のデータへのアクセスおよび取得を可能にします。ソースドキュメント内のどのデータが処理され、どのように処理されるかは、RaptorXML Server へパスされたPython スクリプト内で指定されています。

Python API

（XML およびXSDのための）Python API は、XML およびXSD ドキュメント内に含まれるメタ情報、構造情報、データへのアクセスを提供します。この結果、API を使用してドキュメント情報にアクセスし処理する Python スクリプトが作成されることができます。例えば、XMLドキュメント からデータベースまたはCSV ファイルへデータを書き込むPython スクリプトがRaptorXML Server へパスされることができます。

Raptor のPython API のためのサンプルスクリプトは以下で使用することが可能です: https://github.com/altova

Python API はAPI レファレンスで説明されています:

- Python API v1 レファレンス
- Python API v2 レファレンス

メモ Raptor のPython API v1 を使用することはできません。Python API v2 を使用してください。

Python のためのRaptorXML Server パッケージ

RaptorXML Server のインストール内で、ホイールフォーマットのPython パッケージを検索することができます。Python の`pip` コマンドを使用して、このパッケージをインストールのモジュールとしてインストールします。RaptorXML モジュールがインストールされると、モジュールの関数をコード内で使用することができます。この方法で、RaptorXML の機能は、作成するPython プログラム内で、グラフィックライブラリなどの他のサード パーティライブラリと簡単に使用することができます。

RaptorXML Server のPython パッケージの使用方法に関する詳細は、Python パッケージとしてのRaptorXML Server 309を参照してください。

Python スクリプト

ユーザーにより作成されたPython スクリプトは以下のコマンドの`--script` パラメータと共に送信されます。

- `valxml-withxsd (xsi)` 56
- `valxsd (xsd)` 66

Python スクリプトを呼び出すこれらのコマンドは、コマンドライン インターフェイス（CLI）52 およびHTTP インターフェイスを介して200使用することができます。RaptorXML Server のでPython API のPython スクリプトの使用方法は、https://github.com/altova で説明されています。

Python スクリプトを安全にする

HTTP を介して RaptorXML Server に対して、Python スクリプトがコマンド内で指定されている場合、スクリプトは信頼できるディレクトリ[205]にある場合のみ作動します。スクリプトは信頼できるディレクトリから実行されます。Python スクリプトをこれ以外のディレクトリから指定するとエラーが生じます。信頼できるディレクトリはサーバー構成ファイル[203]の server.script-root-dir[204] 設定で指定されており、信頼できるディレクトリは Python スクリプトを使用する場合指定されていなければなりません。使用されるすべての Python スクリプトがこのディレクトリに保存されていることを確認してください。

HTTP ジョブリクエストのためにサーバーにより生成される出力は、(output-root-directory[205] のサブディレクトリである) ジョブ出力ディレクトリ[205] に書き込まれますが、この制限はあらゆる箇所に書き込むことのできる Python スクリプトに対しては適用されません。サーバー管理者は、信頼できるディレクトリ[205]内のスクリプトを脆弱性に関する問題の可能性の点からレビューする必要があります。

6.2.1 Python API バージョン

RaptorXML Server は、Python API バージョン 複数のをサポートします。Python API の以前のバージョンは、RaptorXML Server の現在のバージョンでもサポートされています。Python API バージョンは、**--script-api-version=MAJOR_VERSION** コマンドラインフラグによりサポートされています。 MAJOR_VERSION のデフォルトの引数は、常に最新のバージョンです。新しい RaptorXML Server Python API MAJOR_VERSION は、互換性に関する変更または大幅な機能の向上が追加されると紹介されます。API のユーザーは、新しいバージョンがリリースされても、既存のスクリプトをアップデートする必要はありません。

以下が奨励されます:

- --script-api-version=MAJOR_VERSION フラグを使用して、RaptorXML Server コマンドライン (または Web-API) からユーティリティスクリプトを呼び出します。新しい API MAJOR_VERSION がリリースされても、RaptorXML Server アップデートの後、スクリプトが作動することを保証することができます。
- 今後の RaptorXML Server のリリースで以前のバージョンはサポートされますが、新規プロジェクトのために API の最新バージョンを使用してください。

下にリストされる Python API バージョンを使用することができます。異なる API のドキュメントは、下にリストされるオンライン上のロケーションで使用することができます。

サンプルファイル

Raptor の Python API のためのサンプルスクリプトは以下で使用することが可能です: https://github.com/altova

Python API バージョン 1

RaptorXML Server v2014 で紹介されました。

コマンドラインフラグ: **--script-api-version=1**	
ドキュメンテーション: Python API バージョン 1 レファレンス	

これは、オリジナルの RaptorXML Server Python API です。以下の RaptorXML Server の内部モデルへのアクセスへのサポートをカバーします:

- XML 1.0 and XML 1.1 (API モジュール **xml**)
- XMLSchema 1.0 and XMLSchema 1.1 (API モジュール **xsd**)

- XBRL 2.1 (API モジュール**xbrl**)

API は、Python スクリプトファイル内で実装されるコールバック関数を介して使用することができます。on_xsi_valid

- on_xsd_valid
- on_dts_valid
- on_xbrl_valid

スクリプトは、コマンドライン上の--script オプションで指定されています。コールバック関数は検証に成功した場合のみ呼び出されます。コールバック関数とAPI の詳細については、RaptorXML Server Python API バージョン 1 レファレンス内で説明されています。

メモ Raptor の**Python API v1 は非奨励されています。**Python API v2 を使用してください。

Python API バージョン 2

RaptorXML Server v2015r3 で紹介されました。最新のAPI バージョンは 2.8.3 です。

コマンドラインプラグ	リリース
--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*
ドキュメンテーション: Python API バージョン 2 レファレンス	

API バージョンは、 300 個の新しいクラスを紹介し、頻繁に使用される情報 (例えば、PSVI データ) が更に簡単にアクセスすることができ、関連したAPI が論理的にグループ化 (例えば、xbrl.taxonomy、xbrl.formula、xbrl.table) できるよう RaptorXML Server Python API バージョン 1 からのモジュールを再構成します。このバージョンでは、コールバック機能は検証が成功した場合だけではなく、検証が失敗した場合にも呼び出されます。この振る舞いを反映するため、コールバック機能の名前は以下のように変更されます:

- on_xsi_finished
- on_xsd_finished
- on_dts_finished
- on_xbrl_finished

モジュール化を有効化するために、RaptorXML Server は複数の `--script` オプションをサポートします。Python スクリプトファイル内で実装されるコールバックは、コマンドライン上で指定された順番で実行されます。

6.2.2 Python パッケージとしての RaptorXML Server

RaptorXML Server 2017 から、Python API がネイティブの Python ホイールパッケージ **Python 3.7** として使用できるようになりました。Python ホイールパッケージは、(例えば、python.org からの) 優先する Python 3.7 に拡張モジュールとしてインストールできます。(例えば、from jupyter.org、anaconda.org と SciPy.org) Python 3 配布の一部は、大きなデータ、数学、科学、工学、グラフィックなどのための広範囲におよぶ拡張モジュールを含んでいます。 RaptorXML Server のために特別にこれらのモジュールを構築することなく、これらのモジュールを、RaptorXML Server で使用することができるようになりました。これは、ホイールパッケージが RaptorXML Server 内に含まれる **`RaptorXMLXBRL-python.exe`** アプリケーションと同じ方法で作動するからです

メモ Python ホイールパッケージは、ネイティブの Python 3.7 拡張モジュールと Python バージョン 3.7 との互換性が必要です。

メモ Python ホイールパッケージは Python API v1 を含んでいません。

メモ RaptorXML Server のバージョンを更新する場合、Python 環境内の Python ホイールパッケージが更新されていることを確認してください。

RaptorXML Server パッケージの正しいインストールの必要条件は、下のセクションで説明されています:

- ホイールファイルの名前 309
- ホイールファイルのロケーション 309
- pip を使用してホイールをインストールする 309
- ルート カタログ ファイル 310
- JSON 構成ファイル 310

RaptorXML Server の Python API の使用方法に関しての詳細は、Python API レファレンスとサンプル 307 を参照してください。https://github.com/altova で Raptor の Python API を使用するスクリプトのサンプルも参照してください。

ホイールファイルの名前

ホイールファイルは、次のパターンに従い名前が付けられます:

```
raptorxmlserver-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl
```

サンプル:

```
raptorxmlserver-2.8.1-cp35-cp35m-win_amd64.whl
```

ホイールファイルのロケーション

ホイールファイルは、RaptorXML Server のインストールにパッケージされています。アプリケーションの **`bin`** フォルダー内にあります:

Windows	`Program Files\Altova\RaptorXMLServer2021\bin`
Linux	`/opt/Altova/RaptorXMLServer2021/bin`
Mac	`/usr/local/Altova/RaptorXMLServer2021/bin`

pip を使用してホイールをインストールする

RaptorXML Server パッケージを Python のモジュールとしてインストールするには、**`pip`** コマンドを使用します:

```
pip install <wheel-file>.whl
python -m pip install <wheel-file>.whl
```

python.org のPython 3.7 または以降がインストールされている場合、`pip` は、既にインストールされています。それ以外の場合、**`pip`** を最初にインストールする必要があります。詳細に関しては、https://docs.python.org/3/installing/ を参照してください。

ルート カタログ ファイル

Python のためのRaptorXML モジュールは、RaptorXML Server インストールフォルダー内に保管されている **`RootCatalog.xml`**、ルート カタログ ファイル、をロケートすることができなければなりません。これは、RaptorXML モジュールがカタログを使用して、スキーマや他の仕様などの様々なリソースを正確にロケートし、検証や変換などの関数を実行するために参照するためです。RaptorXML モジュールは、RaptorXML Server のインストールのあとにカタログの場所が変更されていない場合、自動的に **`RootCatalog.xml`** をロケートします。

RaptorXML Server 環境を移動、または、変更する場合、または、**`RootCatalog.xml`** をインストールされている元の場所から移動すると、カタログの場所を環境変数とRaptorXML モジュールのJSON 構成ファイル[309]を用いて指定することができます。これをおこなうための様々な方法が下にリストされています。 RaptorXML モジュールは、与えられた順番に次のリソースを探し、**`RootCatalog.xml`** の場所を決定します。

1	環境変数 `ALTOVA_RAPTORXML_PYTHON_CONFIG`	**`RootCatalog.xml`** へのパスである値と共に作成します。
2	HKLM レジストリ: `SOFTWARE\Altova\RaptorXMLServer\Installation_v2021_x64\Setup\CatalogPath`	レジストリキーは、RaptorXML Server インストーラーにより追加されます。値は**`RootCatalog.xml`** へのパスです。*Windows のみ*
3	ロケーション: `/opt/Altova/RaptorXMLServer2021/etc/RootCatalog.xml`	*Linux のみ*
4	ロケーション: `/usr/local/Altova/RaptorXMLServer2021/etc/RootCatalog.xml`	*Mac のみ*
5	環境変数 `ALTOVA_RAPTORXML_PYTHON_CONFIG`	JSON 構成ファイル[310]へのパスである値と共に作成します。
6	ロケーション: `.altova/`**`raptorxml-python.config`**	現在作業中のディレクトリ内のJSON 構成ファイル[310]
7	ロケーション: `~/.config/altova/`**`raptorxml-python.config`**	ユーザーのホームディレクトリ内のJSON 構成ファイル[310]
8	ロケーション: `/etc/altova/altova/`**`raptorxml-python.config`**	JSON 構成ファイル[310]。*Linux とMac のみ*

JSON 構成ファイル

RaptorXMLServer モジュールのためにJSON 構成ファイルを作成することができます。このファイルは、上のテーブル内のオプション 5 から 8 へルート カタログ ファイル[310]を検索するために、使用することができます。JSON 構成ファイルは、ルート カタログ ファイル[310] へのパスである値を持つ「`CatalogPath`」キーを持つマップを含んでいる必要があります。

JSON 構成ファイルのリスティング

```
{
    "CatalogPath": "/path/to/RootCatalog.xml"
}
```

6.2.3 サーバー側の Python スクリプトのデバッグ

サーバー固有のコールバックとは個別に、デバッグ機能の多くはRaptorXML Server モジュールを`pip` を使用してインストール後標準 Python インタープリターまたは環境内で使用することができます:

```
pip install -upgrade "/path/to/RaptorXML/application-folder/bin/raptorxml-version-cp37-cp37m-winversion.whl"
```

ホイールのインストール後 Python IDE を使用してスクリプトをデバッグできるようになります。メイン機能をインスタントオブジェクトを取る個別の機能に抽出することができます。これは(i) RaptorXML Server コールバックにより呼び出し、または(ii) Python インタープリターを使用して直接実行することができます。

```
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)
```

6.2.4 Visual Studio コード内のスクリプトの Python デバッグ

`ms-python.python` 拡張子を持つ更新された Visual Studio コード (VS コード) インストールが予想されます。公式の Visual Studio コードガイド内のPython デバッグ構成 を読んでください。

以下の点に注意してください:

- このガイドは`raptorxml-python` をPython インタープリターとして RaptorXML Server を実行するコマンドとして使用します。
- `raptorxml-python` 実行可能ファイルはRaptorXML Server アプリケーションフォルダーの`bin` フォルダー内で使用することができます。

概要

VS コードを使用して RaptorXML Server 内で Python スクリプトをでデバッグする2通りの方法を紹介します。

- メソッド 1 はサーバーと RaptorXML Python コールバック（`--script` オプション）に対して作動します。
- メソッド 2 はソースコードの変更を必要としません。RaptorXML の変更された呼び出しです。メソッド 2 はサーバーと RaptorXML Python コールバック（`--script` オプション）に対して作動しません。
- 両方のメソッドは標準の Python インタープリターとインポートされた RaptorXML Python モジュール（`'import altova_api.v2 as altova'`）と作動します。

メソッド 1: ソースコードを変更します。

以下のステップを行ってください：

1. 実行: `raptorxml-python -m pip install --upgrade debugpy`
2. Python ソースコードに以下のラインを追加します:
```
python
import debugpy
debugpy.listen(5678)
debugpy.wait_for_client()
debugpy.breakpoint()
```
3. VS コード `launch.json`（上記の値に対してはデフォルトを使用します）にこのローンチ構成をコピーし `Run` のために選択します。
```
json5
  {
    "name": "Python: Remote Attach",
    "type": "python",
    "request": "attach",
    "connect": {
    "host": "localhost",
    "port": 5678
    },
    "pathMappings": [
      {
        "localRoot": "${workspaceFolder}",
        "remoteRoot": "."
      }
    ]
  }
```
メニューコマンド **Run->Add Configuration...->Python->Remote Attach** を使用して受け入れられたデフォルトと共に実行することができます。
4. Python スクリプト（または RaptorXML を `--script` コールバック）を使用して通常通り実行します。
5. デバッグを開始します（通常は **F5**を使用します）。

メソッド 2: 変更されたコマンドラインの使用

以下のステップを行ってください：

1. ローンチ構成を追加し（メソッド 1 と同様）`Run` のために選択します。
2. Python スクリプト内でブレークポイントを設定します。
3. コマンドを実行します: `raptorxml-python -m debugpy --listen 0.0.0.0:5678 --wait-for-client your-script-.py`
4. デバッグを開始します（通常はショートカット **F5**を使用します）。

 (C) 2015-2021 Altova GmbH

メモ デバッグはコンテナーおよびリモートサーバーでも作動します。構成をローンチするにはconnect エントリのhost キーも変更する必要があります。コマンドラインとlaunch.json が整合性のある値を持つ限り他のポートを使用することもできます。

raptorxml-python.exe を VS コードのデフォルトのインタープリターとして設定する

raptorxml-python.exe をVS コードのデフォルトのPython インタープリターとして構成することもできます。これを以下をVS コード settigs.json ファイルに追加して行います。:

```
json
  "python.defaultInterpreterPath": "/path/to/raptorxml-python.exe"
   ...
```

この場合、デバッグのためにスクリプトを開始する[現在のファイル]のローンチ構成を使用することもできます。詳細に関してはオフィシャルの VS コードドキュメンテーションを参照してください。

6.2.5 よくある質問

Q: *Raptor サーバー内で実行中に要素を一個ずつ作成する新規の XML インスタンスを作成する Python スクリプトを作成できますか。これらはパラメーターに従い異なるエンコードとフォーマットを持つ出力でシリアル化される必要があります。RaptorXML Server 内で可能です。*

A: いいえ。任意のXML インスタンスを作成するAPI が存在しないためこれは現在不可能です。XBRL インスタンスの生成に関しては(コンテキスト/ユニットなどの複製を回避する)多くの技術を管理する高度な API をご使用いただけます。詳細に関しては https://www.altova.com/manual/RaptorXML/pyapiv2/html/xbrl.InstanceDocumentBuilder.html のセクションを参照してください。)

Q: *lxml を使用できますか。lxml ライブラリを "RaptorXMLXBRLServer2020/lib/python3.7/site-packages/" にインストールすることはできますか?*

A: Python モジュールの大半を直接以下のコマンドを管理者権限のあるターミナル内で実行してインストールすることができます:

```
"/path/to/RaptorXML/application-folder/bin/RaptorXMLXBRL-python.exe" -m pip install
lxml
```

Q: *XML インスタンスを含む大きな文字列を作成後にサイズを調整することは可能ですか。*

A: これは1つの可能性です。XML とXBRL インスタンスを以下のようにPython API を使用して文字列バッファーから解析し検証することができます:

```
from altova_api.v2 import xml
txt = '''<?xml version="1.0" encoding="utf-8"?>
<doc>
    <elem attr="foo">bar</elem>
</doc>'''
inst = xml.Instance.create_from_buffer(txt.encode('utf-8')).result
print(inst.serialize())
```

6.3 .NET Framework API

RaptorXML Server の **.NET Framework API** of により **RaptorXML エンジンを** C# と 他の .NET 言語により書かれたアプリケーションに統合することができます。

これは a .NET アセンブリとして実装され **RaptorXML エンジン** をアプリケーションに、または VSTO (Visual Studio Tools for Office)などの .NET-framework ベースの拡張メカニズム内に直接配置します。API はドキュメントを検証するためのアクセスであり、RaptorXML Server からの内部データモデルをクエリする為に使用されます。

レファレンスとソース

- *API ドキュメント:* 最新のRaptorXML Server .NET Framework API ドキュメントは以下でみつけることができます: https://www.altova.com/manual/RaptorXML/dotnetapiv2/html/index.html
- *サンプルコード:* サンプルコードは以下でホストされています https://github.com/altova/RaptorXML-Examples。

7 追加情報

このセクションは以下の追加情報を含んでいます:

- 終了コード 316
- スキーマのロケーションのヒント 317

7.1 終了コード

次の終了コードを使用することができます:

0	検証に成功しました。
1	エラーのために検証に失敗しました / Ctrl+C によりプロセスが中断されました/ブレーク/ターミナルが閉じられました / 実行中にライセンスの期限が切れました。
11	RaptorXML を開始することができませんでした。理由はログファイルを参照してください。
22	ルートカタログをロードできませんでした。 / リストファイルをロードすることができませんでした。
64	無効なコマンド/オプション
77	スタートアップでライセンスの取得に失敗しました。
128+**n**	RaptorXML がシグナル番号 **n** のために中断されました。**128** 以上の全ての終了コードは、受信された外部シグナルまたは内部でトリガーされたシグナルの結果の中断を意味します。例えば、終了コードが **134** の場合、シグナル番号は **134-128=6** (SI GABRT の番号)となります。

7.2 スキーマのロケーションのヒント

インスタンスドキュメントはスキーマのロケーションを示すヒントを使用することができます。ヒントには2 つの属性が使用されます:

- xsi:schemaLocation ターゲット名前空間を持つスキーマドキュメントのため。属性の値はアイテムのペアです。1 つは名前空間で、もう1 つはスキーマドキュメントを検索するURL です。名前空間の名前はスキーマドキュメントのターゲット名前空間に一致する必要があります。
```
<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 ターゲット名前空間のないスキーマドキュメントのため。属性の値はスキーマドキュメントのURL です。参照されるスキーマドキュメントは、ターゲット名前空間を持ってはなりません。
```
<document xmlns="http://www.altova.com/schemas/test03"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="Test.xsd">
```

--schemalocation-hints オプションは、これら2 つの属性がどのようにヒントと使用されるか、特にschemaLocation 属性の情報が扱われるかを、指定します（上のオプションの詳細を参照してください）。 RaptorXML Server は、名前空間を空白文字列となるxsi:noNamespaceSchemaLocation 値の一部として扱うことに注意してください。

スキーマロケーションのヒントはXML スキーマドキュメントのimport ステートメント内に与えられることができます。

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

import ステートメント内でも、カタログファイル内のスキーマにマップすることのできるヒントは名前空間を介して与えらえることができます。または、直接URL としてschemaLocation 属性に与えられることができます。（XBRL およびXSD/XML のための）--schema-imports 188 オプションは、スキーマロケーションがどのように選択されるかを指定します。

8 エンジン情報

このセクションにはRaptorXML Server 内に含まれているXSLT とXQuery エンジンについての情報が含まれています。この情報は、仕様ではなく実装が振る舞いを決定する状況のエンジンの振る舞いについての情報です。このセクションには、XPath/XQuery のための Altova 拡張関数に関する情報も含まれています。

(C) 2015-2021 Altova GmbH

8.1 XSLT および XQuery エンジンに関する情報

RaptorXML Server のXSLT およびXQuery エンジンは、W 3C 仕様に従っています、ですから、XMLSpy の以前のバージョン内の Altova エンジン およびRaptまたはXML の先行であるAltovaXML より厳密です。この結果、以前のエンジンで無視されていた小さなエラーが、RaptorXML Server によりエラーとして挙げられます。

例えば

- パス演算子の結果がノードと非-ノードを両方含む場合、型エラー(`err:XPTY0018`) です。
- パス式 `E1/E2` 内の`E1` がノードのシーケンスを評価しない場合、型エラー(`err:XPTY0019`) です。

この種類のエラーが発生した場合、XSLT/XQuery ドキュメントまたはインスタンスドキュメントを必要に応じて修正してください。

このセクションは、エンジンの実装固有の機能を仕様別に整理して説明します。

- XSLT 1.0 319
- XSLT 2.0 319
- XSLT 3.0 321
- XQuery 1.0 321
- XQuery 3.1 324

8.1.1 XSLT 1.0

RaptorXML Server のXSLT 1.0 エンジン は、W orld W ide W eb Consortium(ワールド・ワイド・ウェブ・コンソーシアム) (W 3C) の1999 年 11 月 16 日版のXSLT 1.0 勧告 および1999 年 11 月 16 日版のXPath 1.0 勧告 に準拠します。実装に関しての以下の情報に注意してください。

実装についての注意点

`xsl:output` の`method` 属性がHTMLに設定された場合、または、がHTML 出力 デフォルトで選択されている場合、XML または XSLT ファイル内の特殊文字はHTML ドキュメントにHTML 文字参照として出力内に挿入されます。例えば、文字 U+00A0 (ブレーク無しのスペースのための16 進数レファレンス) がHTML コード内に文字の参照(` ` or ` `) 、または、エンティティ参照 ` ` として挿入されます

8.1.2 XSLT 2.0

このセクション:

- エンジン 適合性 320
- 下位互換性 320
- 名前空間 320
- スキーマ認識 320
- 実装固有の振る舞い 321

適合性

RaptorXML Server のXSLT 2.0 エンジンは、W orld W ide W eb Consortium（ワールド・ワイド・ウェブ・コンソーシアム）（W 3C）の2007 年 1 月 23 日版のXSLT 2.0 勧告 および2010 年 12 月 14 日版のXPath 2.0 勧告に準拠します。

下位互換性

XSLT 2.0 エンジン は下位互換性を有します。XSLT 2.0 エンジンの 下位互換性が有効になるのは、 XSLT 1.0 スタイルシートを処理するためにXSLT 2.0 エンジン（CLI パラメーター--xslt=2 192）が使用される際です。XSLT 1.0 エンジン と下位互換性を持つ XSLT 2.0 エンジンにより作成される出力に違いがあるかもしれないことに注意してください。

名前空間

XSLT 2.0 スタイルシートは、 XSLT 2.0. プレフィックス内で使用することのできる型コンストラクタ および関数を使用するため、以下の名前空間を宣言する必要があります。 下のリストは通常使用されるリスト です。希望する場合は、代替プレフィックスを使用することもできます。

名前空間	プレフィックス	名前空間 URI
XML スキーマ型	xs:	http://www.w3.org/2001/XMLSchema
XPath 2.0 関数	fn:	http://www.w3.org/2005/xpath-functions

通常これらの名前空間は、以下のリスト で表示されるようにxsl：スタイルシートまたは、 xsl:transform 要素で宣言されます:

```
<xsl：スタイルシート 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：スタイルシート>
```

次の点に注意してください:

- XSLT 2.0 エンジンは、（上のテーブルでリストされている）XPath 2.0 およびXQuery 1.0 関数 名前空間 を**デフォルト の関数名前空間**として使用します。 XPath 2.0 およびXSLT 2.0 関数をプレフィックス無しでスタイルシート内で使用することができます。XPath 2.0 関数 名前空間 をスタイルシート内でプレフィックスと共に宣言すると、 割り当てられた宣言内でプレフィックスを追加して使用することができます。
- XML スキーマ 名前空間から型コンストラクタと型を使用する場合、名前空間 宣言内で使用された、プレフィックスを使用して型コンストラクタを呼び出さなければなりません（例えば、 xs:date）。
- XPath 2.0 関数の一部は、XML スキーマ データ型と同じ名前を保有します。 例えば、 XPath 関数 fn:string および fn:boolean のためには、同じロケーション名: xs:string および xs:boolean を持つ、XML スキーマ データ型 が存在します。ですから、 XPath 式 string('Hello') を使用する場合、 式は、 xs:string('Hello') ではなくて fn:string('Hello') として検証します。

スキーマ認識

XSLT 2.0 エンジンは、スキーマを認識します。ですから、ユーザー定義 スキーマ型 および xsl:validate 命令を使用することができます。

実装固有の振る舞い

以下は、XSLT 2.0 エンジンが、特定のXSLT 2.0 関数の振る舞いの実装-特定のアスペクトをどのように扱うかの説明です。

xsl:result-document
追加してサポートされるエンコードは以下の通りです（Altova-固有）: `x-base16tobinary` および `x-base64tobinary`.

function-available
インスコープ関数の使用をテストする関数（XSLT、XPath、および拡張関数）。

unparsed-text
`href` 属性は、以下を受け入れます（i）ベース-uri フォルダー内のファイルの相対パス、および（ii） 相対パスを持つまたは持たない `file://` プロトコール。追加してサポートされるエンコードは以下の通りです（Altova-固有）: `x-binarytobase16` および `x-binarytobase64`.

unparsed-text-available
`href` 属性は、以下を受け入れます（i）ベース-uri フォルダー内のファイルの絶対パス、および（ii）絶対パスを持つまたは持たない `file://` プロトコール。追加してサポートされるエンコードは以下の通りです（Altova-固有）: `x-binarytobase16` および `x-binarytobase64`.

メモ RaptorXML の先行製品である AltovaXML で実装されていた以下のエンコード値は使用しないでください: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

8.1.3 XSLT 3.0

RaptorXML Server のXSLT 3.0 エンジンは、W orld W ide W eb Consortium（ワールド・ワイド・ウェブ・コンソーシアム）（W 3C）の2017 年 6 月 8 日版 XSLT 3.0 勧告候補 および2017 年 3 月 21 日版 XPath 3.1 勧告候補 に準拠します。

XSLT 3.0 エンジンは、XSLT 2.0 エンジンと同様の実装固有の機能を搭載しています。更に、以下のXSLT 3.0 機能へのサポートを含みます。更に次の一連の新規のXSLT 3.0 機能をサポートします: XPath/XQuery 3.1 関数とオペレーターとXPath 3.1 仕様。

メモ 任意のストリーミングの機能は現在サポートされていません。`streamable` 属性の値に関係なく、ドキュメント全体がメモリにロードされ、使用することのできるメモリが十分な場合は処理されます。64-ビット アプリでは、これは問題になりません。thisメモリが問題の場合は、システムに解決策を追加する必要があります。

8.1.4 XQuery 1.0

このセクション:

- エンジン適合性 322
- スキーマ認識 322
- エンコード 322
- 名前空間 320
- XML ソースと検証 323
- 静的および動的な型のチェック 323
- ライブラリモジュール 323
- 外部関数 323

- 照合順序 323
- 数値データの精度 323
- XQuery 命令サポート 324

適合性

RaptorXML Server のXQuery 1.0 エンジンは、World Wide Web Consortium（ワールド・ワイド・ウェブ・コンソーシアム）（W3C）の2010 年 12 月 14 日版のXQuery 1.0 勧告に準拠します。XQuery 標準は、多数の機能の実装についての裁量を提供します。下記は、XQuery 1.0 エンジンがどのようにこれらの機能を実装するかについて説明するリストが下に挙げられています。

スキーマ認識

XQuery 1.0 エンジン はスキーマを認識します。

エンコード

UTF-8 およびUTF-16 文字のエンコードは、サポートされています。

名前空間

以下の名前空間 URI と関連するバインドは定義済みです。

名前空間	プレフィックス	名前空間 URI
XML スキーマ型	`xs:`	`http://www.w3.org/2001/XMLSchema`
スキーマインスタンス	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
内蔵の関数	`fn:`	`http://www.w3.org/2005/xpath-functions`
Local 関数	`local:`	`http://www.w3.org/2005/xquery-local-functions`

次の点に注意してください：

- XQuery 1.0 エンジンは、上にリストされたプレフィックスを名前空間に対応するバウンドとして認識します。
- Since the 上にリストされた内蔵の関数 名前空間は、XQuery内のデフォルトの関数です。内蔵の関数 が呼び出される際、名前空間、fn: プレフィックス を使用する必要はありません。（例えば、`string("Hello")` が`fn:string` 関数を呼び出す場合。）しかし、プレフィックス `fn:` は、クエリプロローグ内で名前空間を宣言することなく内蔵の関数を呼び出す時に使用することができます。（サンプル: `fn:string("Hello")`）.
- クエリプロローグ内で`default function` 名前空間 式を宣言することにより、デフォルトの関数 名前空間をすることにより変更することができます。
- XML スキーマ 名前空間空の型を使用する場合、プレフィックス `xs:` は、名前空間を明確に宣言することなく、また、これらのプレフィックスをクエリプロローグ内でバインドすることなく使用することができます。（サンプル: `xs:date` および `xs:yearMonthDuration`。）XML スキーマ 名前空間のために、他のプレフィックスを使用する場合は、クエリプロローグ内で明確に宣言されている必要があります。（サンプル: `declare 名前空間 alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.）
- `untypedAtomic`, `dayTimeDuration`, および`yearMonthDuration` データ型 が23 January 2007 のCR 共に、XPath データ型 名前空間からXML スキーマ 名前空間へ移動されていることに注意してください。ですから以下となります: `xs:yearMonthDuration`。

関数のための名前空間、型コンストラクタ、ノード テスト、が間違って割り当てられている場合、エラーが発生します。しかし、一部の関数はスキーマデータ型と同じ名前を持つことに注意してください。 例: `fn:string` および `fn:boolean`。(`xs:string` および `xs:boolean` は宣言されています) 名前空間 プレフィックス、関数または 型コンストラクタが使用されるか決定します。

XML ソースドキュメントと検証

XQuery 1.0 エンジンを使用して、実行される XQuery ドキュメント内の XML ドキュメントは、整形式である必要があります。しかし、XML スキーマに従い有効である必要はありません。XML ファイルが外部スキーマと関連付けられ、また有効な場合、ポスト スキーマ検証情報が XML データのために生成され、クエリ検証のために使用されます。

静的および動的な型のチェック

静的分析フェーズは、外部レファレンスの存在（例、モジュールのため）、呼び出された関数と変数が定義済みであるか、など構文などのクエリのアスペクトをチェックします。静的分析フェーズでエラーが検知されると、実行は停止されます。

クエリが実際に作動中にランタイム中に動的な型チェックは実行されます。型がオペレーションの必要条件と整合性を持たない場合、エラーが報告されます。例えば、式 `xs:string("1") + 1` は、エラーを返します。型 `xs:string` のオペランドを足し算のオペレーションが実行できないためです。

ライブラリモジュール

ライブラリモジュールは、再利用のため関数と変数を保管します。XQuery 1.0 エンジンは単一の外部 XQuery ファイルに保管されているモジュールをサポートします。このようなモジュール ファイルはプロローグにターゲット名前空間に関連するモジュール宣言を含む必要があります。 以下はモジュールサンプルです:

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

すべての関数および変数は、モジュールに関連した名前空間に属するモジュール内で宣言されています。モジュールはクエリプロローグ内の `import module` ステートメントを使用して XQuery ファイルにインポートする際に使用されます。`import module` ステートメントは、ライブラリモジュールファイル内で直接宣言された、関数と変数のみをインポートします。例:

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

外部関数

外部関数 は、サポートされていません。例えば、`external` キーワードを仕様する式など。以下参照:

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

照合順序

デフォルトの照合順序は、Unicode コードポイントをベースにした文字列を比較する Unicode-コードポイント照合順序です。その他にサポートされる照合順序は、ICU 照合順序 はここ 325 にリストされるとおりです。特定の照合順序を使用する場合、サポートされる照合順序のリスト 325 内に与えられているとおり URI を提供します。`fn:max` と `fn:min` 関数を含む文字列の比較は、指定された照合順序により行われます。 照合順序オプションが指定されていない場合、デフォルトの Unicode-コードポイント照合順序 が使用されます。

数値データの精度

- `xs:integer` データ型には任意の精度があり、表記できる桁数に制限はありません。
- `xs:decimal` データ型には小数点の後に 20 桁の制限があります。
- `xs:float` と `xs:double` データ型には 15 桁の精度の制限があります。

XQuery 命令サポート

`Pragma` 命令 は、サポートされていません。発生した場合、無視されフォールバックの式が検証されます。

8.1.5 XQuery 3.1

RaptorXML Server のXQuery 3.1 エンジンは、World Wide Web Consortium（ワールド・ワイド・ウェブ・コンソーシアム）（W3C）の2017 年 3 月 21 日版のXQuery 3.1 候補勧告に準拠し、またXPath およびXQuery 関数 3.1.へのサポートを含みます。XQuery 3.1 仕様は、3.0 仕様のスーパーセットです。XQuery 3.1 エンジンは、ですかXQuery 3.0 機能をサポートします。

実装固有の特性はXQuery 1.0 321 同様です。

8.2 XSLT と XPath/XQuery 関数

このセクションでは、XPath および/または XQuery 式で使用することのできる、Altova 拡張関数と他の拡張関数をリストします。Altova 拡張関数は Altova の XSLT および XQuery エンジンで使用することができ、W3C 標準で定義された関数ライブラリで使用することのできる機能に追加して機能を提供します。

一般的な情報

以下の一般的な情報に注意してください：

- W3C 仕様により定義されているコア関数ライブラリの関数は、関数の呼び出しにプレフィックスは必要ありません。これは、XSLT および XQuery エンジンが、XPath/XQuery 関数仕様で指定されている `http://www.w3.org/2005/xpath-functions` プレフィックス無しの関数をデフォルト関数の名前空間に属するものとして読み込むためです。
- 関数において、各アイテムが引数となるようなシーケンスが期待されており、2つ以上のアイテムがシーケンスにより呼び出された場合、エラーが返されます。
- 全ての比較は Unicode コードポイントコレクションを使用することで行われます。
- QName の結果は `[prefix:]localname` という形式でシリアル化されます。

xs:decimal の精度

精度とは、数値内にある桁数のことで、仕様では少なくとも18桁が求められます。`xs:decimal` 型に結果が収められる除算の場合、端数処理を行うことなく精度は小数点以下の19桁になります。

黙示的なタイムゾーン

2つの `date`、`time`、または `dateTime` 値を比較する場合、比較する値のタイムゾーンを明らかにする必要があります。値の中にタイムゾーンが明示的に与えられていない場合、黙示的なタイムゾーンが使用されます。黙示的なタイムゾーンはシステムクロックから取得され `implicit-timezone()` 関数によりその値をチェックすることができます。

照合順序

デフォルトの照合順序は、Unicode コードポイントをベースに文字列を比較する Unicode コードポイント照合順序です。エンジンは Unicode 照合アルゴリズムを使用しています。他のサポートされる照合順序は下にリストされる ICU 照合順序 です。使用するには、サポートされる照合順序のリストの URI を提供してください（*下のテーブル*）。`max` と `min` 関数を含む、文字列の比較は、指定された照合順序に沿って行われます。照合順序オプションが指定されていない場合、デフォルトの Unicode コードポイント照合順序が使用されます。

言語	URI
`da`: デンマーク語	`da_DK`
`de`: ドイツ語	`de_AT, de_BE, de_CH, de_DE, de_LI, de_LU`
`en`: 英語	`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`: スペイン語	`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`: フランス語	`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: イタリア語	it_CH, it_IT
ja: 日本語	ja_JP
nb: ノルウェー語 (ブークモール)	nb_NO
nl: オランダ語	nl_AW, nl_BE, nl_NL
nn: ノルウェー語 (ニーノシュク)	nn_NO
pt: ポルトガル語	pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST
ru: ロシア語	ru_MD, ru_RU, ru_UA
sv: スウェーデン語	sv_FI, sv_SE

名前空間軸
名前空間軸はXPath 2.0 にて廃止されましたが、名前空間軸の使用はサポートされています。XPath 2.0 メカニズムにより名前空間情報へアクセスするには、in-scope-prefixes()、namespace-uri()、namespace-uri-for-prefix(関数を使用してください。

8.2.1 Altova 拡張関数

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間、http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることに注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

W 3C のXPath/XQuery 関数仕様で定義された関数は、以下で使用することができます: (i) XSLT 子アンテキスト内のXPath 式と、(ii) XQuery 文書内のXQuery 式。このドキュメントでは、前者(XSLT 内のXPath) のコンテキストで使用することのできる関数を、**XP** シンボルと共に表示し、と称します。後者(XQuery) で使用することのできる関数はの**XQ** シンボルと共に表示され、XQuery 関数と共に作業することができます。W 3C のXSLT 仕様は、XPath/XQuery 関数の仕様ではなく、XSLT 文書内のXPath 式でも使用することのできる関数を定義します。これらの関数は、**XSLT** シンボルと共に表示され、XSLT 関数と称されます。関数を使用することのできるXPath/XQuery およびXSLT のバージョンは、関数の詳細に記載されています (*下のシンボルを参照してください*)。
XPath/XQuery およびXSLT 関数ライブラリからの関数は、プレフィックス無しでリストされています。Altova 拡張関数などの、他のライブラリからの関数はプレフィックスと共にリストされています。

XPath 関数 (XSLT 内のXPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内のXPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内のXQuery 式で使用):	**XQ1** **XQ3.1**

XSLT 関数 327

XSLT 関数はXSLT 2.0 の`current-group()` や`key()` 関数と同様に、XSLT コンテキストにて使用することができます。(例えば、XQuery コンテキストなどの) 非-XSLT コンテキストでは使用することができません。XBRL に対するXSLT 関数は、XBRL をサポートするエディションのAltova 製品でのみ使用することができます。

XPath/XQuery 関数

XPath/XQuery 関数は、 XSLT コンテキスト、 XQuery 関数のXPath 式で使用することができます:

- 日付/時刻 330
- 位置情報 344
- イメージに関連した 355
- 数値 363
- シーケンス 383
- 文字列 390
- その他 396

チャート関数 (Enterprise および Server Editions のみ) 398

チャート関数のためのAltova 拡張子は、**Enterprise** ならびに サーバー エディションのAltova 製品でしかサポートされていません。XML データからチャートを生成することができます。

バーコード関数 409

Altova バーコード拡張関数はバーコードを生成し、スタイルシートを介して生成された出力に配置することができます。

8.2.1.1 XSLT 関数

XSLT 拡張関数 はXSLT コンテキスト内のXPath 式にて使用することができます。(例えば、XQuery コンテキストなどの) 非-XSLT コンテキストでは使用することができません。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、`http://www.altova.com/xslt-extensions` に収められており、`altova:` プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

標準関数

▼ distinct-nodes [altova:]

altova:distinct-nodes(*node()****)** を **node()*** とする XSLT1 XSLT2 XSLT3
入力として1つ以上のノードを必要とし、同じセットから重複した値を持つノードを除いたノードを返します。XPath/XQuery 関数

fn:deep-equal を使用して比較を行うことができます。

サンプル

- **altova:altova:distinct-nodes**(country) は重複した値を持つものを除く、全ての子 country ノード返します。

evaluate [altova:]

altova:evaluate(XPathExpression *as xs:string*[, **ValueOf$p1, ... ValueOf$pN])** **XSLT1** **XSLT2** **XSLT3**

XPath 式を必要とし、必須引数として文字列をパスします。評価された式の出力を返します。例えば **altova:evaluate('//Name[1]')** は、ドキュメント内の最初の Name 要素のコンテンツを返します。式 //Name[1] は、一重引用符を使用することにより、文字列としてパスされます。

altova:evaluate 関数は、オプションとして追加の引数を持つことができます。これらの引数は、p1, p2, p3... pN の名前を持つスコープ内の変数の値です。使用に関して以下の点に注意してください：(i) 変数は、X が整数である箇所のフォーム pX の名前と共に定義される必要があります。(ii) altova:evaluate 関数の引数は、(*上の署名参照*) 2 番目の引数からは、数値順の変数のシーケンスに対応した引数のシーケンス変数の値を与えます: p1 to pN: 第 2 引数は変数 p1 の値で、第 3 引数は、変数 p2 の値です。(iii) 変数の値は型 item* である必要があります。

サンプル

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

outputs "hi 20 10"

上のリストに関して、以下の点に注意してください：

- altova:evaluate 式の第 2 引数は、変数 $p1 に割り当てられた値で、第三の引数は変数 $p2 に割り当てられた値です。
- 関数の第 4 番目の引数は、引用符による囲いで表示された文字列の値です。
- xs:variable 要素の select 属性は、XPath 式を提供します。この式は xs:string の型である必要があり、一重引用符で囲まれています。

変数の使用方法を更に説明するサンプル

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, //Name[1])" />
  ```
  最初の Name *要素の出力値*

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

altova:evaluate() 拡張関数は、XSLT スタイルシート内の XPath 式が動的に評価される必要のあるシチュエーションで役に立ちます。例えば、ユーザーが並べ替えの必要条件をリクエストする場合、このシチュエーションは属性 UserReq/@sortkey に保管されます。スタイルシートでは、以下の式が使用できます: <xsl:sort select="**altova:evaluate(../UserReq/@sortkey)**" order="ascending"/>。altova:evaluate() 関数は、コンテキストノードの親の UserReq 子要素の sortkey 属性を読み込みます。sortkey 属性の値が Price の場合、Price は、altova:evaluate() 関数により返され、select 属性 <xsl:sort select="**Price**" order="ascending"/> の値になります。この sort 命令が、Order という要素のコンテキスト内で発生する場合、Order 要素は Price の子の値に従い並べ替えられます。また、@sortkey の値が、Date の場合、Order 要素は、Date の子の値に従い並べ替えられます。ですから、Order の並べ替えの条件は、ランタイムでの sortkey 属性から選択されます。これは、以下の式な

どでは達成することはできません: `<xsl:sort select="../UserReq/@sortkey" order="ascending"/>`。上の場合、並べ替え条件は sortkey 属性自身であり、Price または Date（または、現在の sortkey のコンテンツ）ではありません。

メモ 静的なコンテキストは、変数以外以外で、呼び出し環境の名前空間、型、機能、を含みます。ベース URI とデフォルトの名前空間は継承されます。

追加サンプル

- 静的な変数: `<xsl:value-of select="$i3, $i2, $i1" />`
  3 つの変数の値を出力します。

- 動的な変数を持つ動的 XPath 式:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />
  ```
  "30 20 10" を出力します。

- 動的な変数を持たない XPath 式:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath)" />
  ```
  出力エラー: $p3 に対して定義されている変数はありません。

encode-for-rtf [altova:]

altova:encode-for-rtf(input *as xs:string*, **preserveallwhitespace** *as xs:boolean*, **preservenewlines** *as xs:boolean*) を **xs:string** とする XSLT2 XSLT3
RTF のためのコードに入力文字列を変換します。空白と新しい行は、それぞれの引数により指定される boolean の値に基づき保管されます。

[トップ 327]

XBRL 関数

Altova XBRL 関数は XBRL をサポートする Altova 製品のエディションのみで使用することができます。

xbrl-footnotes [altova:]

altova:xbrl-footnotes(*node()***)** を **node()*** とする XSLT2 XSLT3
ノードを入力引数として必要とし、入力ノードに参照される XBRL フットノート ノードを返します。

xbrl-labels [altova:]

altova:xbrl-labels(*xs:QName*, *xs:string***)** を **node()*** とする XSLT2 XSLT3
以下の 2 つの入力引数を必要とします: ノード名とノードを含むタクソノミファイルロケーション。関数は、入力ノードと関連した XBRL ラベルノードを返します。

[トップ 327]

8.2.1.2 XPath/XQuery 関数: 日付と時刻

Altova の日付/時刻拡張関数はXPath とXQuery 式で使用することができ、XML スキーマの異なる日付および時刻データ型で保存されているデータを処理するための追加機能を提供します。このセクションの関数は、Altova の**XPath 3.0** および**XQuery 3.0** エンジンで使用することができます。これらの関数は、XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、`http://www.altova.com/xslt-extensions` に収められており、`altova:` プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることに注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

機能によりグループ化

- xs:dateTime に期間を追加して、xs:dateTimeを返す 331
- xs:date に期間を追加して、xs:date を返す 333
- xs:time に期間を追加して、return xs:time を返す 334
- フォーマットと期間の取得 333
- 現在の日付/時刻を生成する関数からタイムゾーンを削除する 335
- 期間から日付、時刻、分数、および、秒数を返す 336
- 日付から整数を週の曜日として返す 338
- 日付から週数を整数として返す 338
- 各型の構文コンポーネントから日付、時刻、期間 の型を構築する 340
- 文字列入力から 日付、日付時刻 または 時刻 を構築する 341
- 年齢に関連した関数 343

アルファベット順にグループ化

altova:add-days-to-date 333
altova:add-days-to-dateTime 331
altova:add-hours-to-dateTime 331
altova:add-hours-to-time 334
altova:add-minutes-to-dateTime 331
altova:add-minutes-to-time 334
altova:add-months-to-date 333
altova:add-months-to-dateTime 331
altova:add-seconds-to-dateTime 331
altova:add-seconds-to-time 334
altova:add-years-to-date 333
altova:add-years-to-dateTime 331
altova:age 343
altova:age-details 343
altova:build-date 340
altova:build-duration 340
altova:build-time 340

altova:current-dateTime-no-TZ 335
altova:current-date-no-TZ 335
altova:current-time-no-TZ 335
altova:date-no-TZ 335
altova:dateTime-no-TZ 335
altova:days-in-month 336
altova:hours-from-dateTimeDuration-accumulated 336
altova:minutes-from-dateTimeDuration-accumulated 336
altova:seconds-from-dateTimeDuration-accumulated 336
altova:format-duration 333
altova:parse-date 341
altova:parse-dateTime 341
altova:parse-duration 333
altova:parse-time 341
altova:time-no-TZ 335
altova:weekday-from-date 338
altova:weekday-from-dateTime 338
altova:weeknumber-from-date 339
altova:weeknumber-from-dateTime 339

[トップ 330]

xs:dateTime に期間を追加する XP3.1 XQ3.1

これらの関数は **xs:dateTime** に期間を追加し、**xs:dateTime** を返します。xs:dateTime 型は CCYY-MM-DDThh:mm:ss.sss のフォーマットです。これは xs:date と xs:time フォーマットの連結で、T により区切られています。タイムゾーンサフィックス +01:00 (for example) は任意です。

add-years-to-dateTime [altova:]

altova:add-years-to-dateTime(DateTime *as xs:dateTime*, **Years** を *xs:integer*)
asxs:dateTime とする **XP3.1** **XQ3.1**
日付までの期間を年数で表示します。第 2 の引数は第 1 の引数として与えられた xs:date に追加される年数です。結果は xs:date 型です。

サンプル

- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) は 2024-01-15T14:00:00 を返します。
- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -4) は 2010-01-15T14:00:00 を返します。

add-months-to-dateTime [altova:]

altova:add-months-to-dateTime(DateTime *as xs:dateTime*, **Months** を *xs:integer*)
asxs:dateTime とする **XP3.1** **XQ3.1**
xs:dateTime に月数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される月数です。結果は xs:dateTime 型です。

サンプル

- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) 2014-11-15T14:00:00 を返します。
- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -2) 2013-11-

15T14:00:00 を返します。

add-days-to-dateTime [altova:]

altova:add-days-to-dateTime(DateTime *as xs:dateTime*, **Days** *as xs:integer*) を **xs:dateTime** とする **XP3.1** **XQ3.1**

xs:dateTime に日数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される日数です。結果は xs:dateTime 型です。

サンプル

- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) は 2014-01-25T14:00:00 を返します。
- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -8) は 2014-01-25T14:00:00 を返します。

add-hours-to-dateTime [altova:]

altova:add-hours-to-dateTime(DateTime *as xs:dateTime*, **Hours** *as xs:integer*) を **xs:dateTime** とする **XP3.1** **XQ3.1**

xs:dateTime に時間数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される時間数です。結果は xs:dateTime 型です。

サンプル

- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), 10) は 2014-01-15T23:00:00 を返します。
- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), -8) は 2014-01-15T05:00:00 を返します。

add-minutes-to-dateTime [altova:]

altova:add-minutes-to-dateTime(DateTime *as xs:dateTime*, **Minutes** *as xs:integer*) を **xs:dateTime** とする **XP3.1** **XQ3.1**

xs:dateTime に分数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される分数です。結果は xs:dateTime 型です。

サンプル

- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), 45) 2014-01-15T14:55:00 を返します。
- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), -5) 2014-01-15T14:05:00 を返します。

add-seconds-to-dateTime [altova:]

altova:add-seconds-to-dateTime(DateTime *as xs:dateTime*, **Seconds** を *xs:integer*) **asxs:dateTime** とする **XP3.1** **XQ3.1**

xs:dateTime に秒数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される秒数です。結果は xs:dateTime 型です。

サンプル

- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), 20) 2014-01-

15T14:00:30 を返します。

- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), -5) 2014-01-15T14:00:05 を返します。

[トップ 330]

xs:date に期間を追加する XP3.1 XQ3.1

これらの関数は**xs:date** に期間を追加し、**xs:date** を返します。xs:date 型はCCYY-MM-DD フォーマットです。

add-years-to-date [altova:]

altova:add-years-to-date(Date *as xs:date*, **Years** *as xs:integer*) を**xs:date** とする XP3.1 XQ3.1

日付までの期間を年数で表示します。第 2 の引数は第 1 の引数として与えられたxs:date に追加される年数です。結果は xs:date 型です。

サンプル

- **altova:add-years-to-date**(xs:date("2014-01-15"), 10) は2024-01-15 を返します。
- **altova:add-years-to-date**(xs:date("2014-01-15"), -4) は2010-01-15 を返します。

add-months-to-date [altova:]

altova:add-months-to-date(Date *as xs:date*, **Months** *as xs:integer*) を**xs:date** とする XP3.1 XQ3.1

日付までの期間を月数で表示します。第 2 の引数は第 1 の引数として与えられたxs:date に追加される月数です。結果は xs:date 型です。

サンプル

- **altova:add-months-to-date**(xs:date("2014-01-15"), 10) 2014-11-15 を返します。
- **altova:add-months-to-date**(xs:date("2014-01-15"), -2) 2013-11-15 を返します。

add-days-to-date [altova:]

altova:add-days-to-date(Date *as xs:date*, **Days** *as xs:integer*) を**xs:date** とする XP3.1 XQ3.1

日付までの期間を日数で表示します。第 2 の引数は第 1 の引数として与えられたxs:date に追加される日数です。結果は xs:date 型です。

サンプル

- **altova:add-days-to-date**(xs:date("2014-01-15"), 10) は 2014-01-25 を返します。
- **altova:add-days-to-date**(xs:date("2014-01-15"), -8) は2014-01-07 を返します。

[トップ 330]

フォーマットと期間の取得 XP3.1 XQ3.1

これらの関数は入力 **xs:duration** または**xs:string** を解析し、それぞれan **xs:string** または**xs:duration** を返します。

format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) **as xs:string** とする **XP3.1** **XQ3.1**
第 1 の引数として提出された期間を、第 2 の引数として提出された文字列によりをフォーマットします。出力は、文字列によりフォーマットされたテキスト文字列です。

サンプル

- **altova:format-duration**(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") は "Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7" を返します。
- **altova:format-duration**(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") は "Months:03 Days:02 Hours:02 Minutes:53" を返します。

parse-duration [altova:]

altova:parse-duration(InputString *as xs:string*, **Picture** *as xs:string*) を **xs:duration** とする **XP3.1** **XQ3.1**
パターン化された文字列を最初の引数として、文字を第 2 の引数とします。入力文字列は文字をベースに解析され、xs:duration が返されます。

サンプル

- **altova:parse-duration**("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") は "P2DT2H53M11.7S" を返します。
- **altova:parse-duration**("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") は "P3M2DT2H53M" を返します。

[トップ 330]

xs:time に期間を追加する **XP3.1** **XQ3.1**

これらの関数は **xs:time** に期間を追加し、 **xs:time** を返します。 **xs:time** 型は hh:mm:ss.sss 構文フォームです。文字 Z は協定世界時（UTC）を表します。他のタイムゾーンは、UTC との差異を +hh:mm または -hh:mm のフォーマットで表示しています。タイムゾーンの値が無い場合は、UTC ではない未知のタイムゾーンとして見なされます。

add-hours-to-time [altova:]

altova:add-hours-to-time(Time *as xs:time*, **Hours** *as xs:integer*) を **xs:time** とする **XP3.1** **XQ3.1**
日付までの期間を時間数で表示します。第 2 の引数は第 1 の引数として与えられた xs:time に追加される時間数です。結果は xs:time 型です。

サンプル

- **altova:add-hours-to-time**(xs:time("11:00:00"), 10) は 21:00:00 を返します。
- **altova:add-hours-to-time**(xs:time("11:00:00"), -7) は 04:00:00 を返します。

add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) を **xs:time** とする

XP3.1 **XQ3.1**
日付までの期間を分数で表示します。第 2 の引数は第 1 の引数として与えられた xs:date に追加される分数です。結果は xs:date 型です。

サンプル

- **altova:add-minutes-to-time**(xs:time("14:10:00"), 45) 14:55:00 を返します。
- **altova:add-minutes-to-time**(xs:time("14:10:00"), -5) 14:05:00 を返します。

add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) を **xs:time** とする
XP3.1 **XQ3.1**
時間までの期間を秒数で表示します。第 2 の引数は、第 1 の引数として与えられた xs:time に追加される秒数です。結果は xs:time 型です。第 2 のコンポーネントは 0 から 59.999 の範囲であることができます。

サンプル

- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20) は 14:00:20 を返します。
- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20.895) は 14:00:20.895 を返します。

[トップ 330]

日付/時刻データ型からタイムゾーンの部分を削除する **XP3.1** **XQ3.1**

これらの関数は、現在の **xs:dateTime**、**xs:date**、または **xs:time** 値からそれぞれタイムゾーンを削除します。xs:dateTime と xs:dateTimeStamp の差異は、後者のタイムゾーンが必要な場合です。(前者の場合は任意です。) xs:dateTimeStamp 値のフォーマットは CCYY-MM-DDThh:mm:ss.sss±hh:mm または CCYY-MM-DDThh:mm:ss.sssZ です。、日付と時刻が xs:dateTimeStamp としてシステムクロックから読み込まれる場合、 current-dateTime-no-TZ() 関数がタイムゾーンを削除するために使用されます。

current-date-no-TZ [altova:]

altova:current-date-no-TZ() を **xs:date** とする **XP3.1** **XQ3.1**
この関数は引数を必要としません。current-date() (システムクロックによる現在の時刻) のタイムゾーンの部分を削除し、xs:date の値を返します。

サンプル

現在の date が **2014-01-15+01:00** の場合:

- **altova:current-date-no-TZ**() は 2014-01-15 を返します。

current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() を **xs:dateTime** とする **XP3.1** **XQ3.1**
この関数は引数を必要としません。current-dateTime() (システムクロックによる現在の時刻) のタイムゾーンの部分を削除し、xs:dateTime の値を返します。

サンプル

現在の dateTime が **2014-01-15T14:00:00+01:00** の場合:

- **altova:current-dateTime-no-TZ**() は 2014-01-15T14:00:00 を返します。

current-time-no-TZ [altova:]

altova:current-time-no-TZ() を **xs:time** とする **XP3.1** **XQ3.1**
この関数は引数を必要としません。current-time()（システムクロックによる現在の時刻）のタイムゾーンの部分を削除し、xs:time の値を返します。

サンプル

現在の time が **14:00:00+01:00** の場合:

- **altova:current-time-no-TZ()** は 14:00:00 を返します。

date-no-TZ [altova:]

altova:date-no-TZ(InputDate *as xs:date***)** を **xs:date** とする **XP3.1** **XQ3.1**
この関数は xs:date 引数を必要とし、タイムゾーンの部分を削除し、xs:date の値を返します。日付が変更されていない点に注意してください。

サンプル

- **altova:date-no-TZ**(xs:date("2014-01-15+01:00")) は 2014-01-15 を返します。

dateTime-no-TZ [altova:]

altova:dateTime-no-TZ(InputDateTime *as xs:dateTime***)** を **xs:dateTime** とする **XP3.1** **XQ3.1**
この関数は xs:dateTime 引数を必要とし、タイムゾーンの部分を削除し、xs:dateTime の値を返します。日付が変更されていない点に注意してください。

サンプル

- **altova:dateTime-no-TZ**(xs:date("2014-01-15T14:00:00+01:00")) は 2014-01-15T14:00:00 を返します。

time-no-TZ [altova:]

altova:time-no-TZ(InputTime *as xs:time***)** を **xs:time** とする **XP3.1** **XQ3.1**
この関数は xs:time 引数を必要とし、タイムゾーンの部分を削除し、xs:time 値を返します。時刻は変更されていないことに注意してください。

サンプル

- **altova:time-no-TZ**(xs:time("14:00:00+01:00")) は 14:00:00 を返します。

[トップ 330]

期間から日数、時間数、分数、および、秒数を返す **XP3.1** **XQ3.1**

これらの関数は、期間から、月内の日数、時間数、分数、秒数をそれぞれ返します。

days-in-month [altova:]

altova:days-in-month(Year *as xs:integer*, **Month** *as xs:integer***) as xs:integer** **XP3.1** **XQ3.1**
は指定された月内の日数 を返します。Year と Month 引数を使用して月を指定することができます。

サンプル

- **altova:days-in-month**(2018, 10) は31 を返します。
- **altova:days-in-month**(2018, 2) は28 を返します。
- **altova:days-in-month**(2020, 2) は29 を返します。

hours-from-dayTimeDuration-accumulated

altova:hours-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*) **asxs:integer**
XP3.1 **XQ3.1**

DayAndTime 引数(which is of type xs:duration) により提供される期間内の時間の総数を返します。Day とTime コンポーネント内の時間は、整数である結果に追加されます。60 分として新規の一時間は考えられます。ネガティブな時間の値で期間の結果をナビゲートすることができます。

サンプル

- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5D") は5 日間の時間の総計である120 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H") は5 日間の時間の総計プラス2 時間である122 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H60M") は5 日間の時間の総計プラス2 時間プラス60 分である123 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H119M") は5 日間の時間の総計プラス2 時間とプラス119 分である123 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H120M") は5 日間の時間の総計プラス2 時間プラス120 分である124 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("-P5DT2H") は-122 を返します。

minutes-from-dayTimeDuration-accumulated

altova:minutes-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*)
asxs:integer **XP3.1** **XQ3.1**

DayAndTime 引数(which is of type xs:duration) により提出される期間内の分数の総数を返します Day とTime コンポーネントは、追加され、整数である結果に追加されます。ネガティブなマイナスの値で期間の結果をナビゲートします。

サンプル

- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT60M") は を返します。60
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H") は一時間内の分数である 60 を返します。
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H40M") は100 を返します。
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("P1D") は一日の秒数の総数である 1440 を返します。
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("-P1DT60M") は を返します。-1500

seconds-from-dayTimeDuration-accumulated

altova:seconds-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*)
asxs:integer **XP3.1** **XQ3.1**

(xs:duration の型である)DayAndTime 引数 により提出された期間内の秒数の総数を返します。Day とTime コンポーネ

ント内の秒数は、整数である結果に追加されます。ネガティブな秒の値で期間をナビゲートします。

サンプル

- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1M") は一分内の秒数である 60 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H") は一時間内の秒数の総数である 3600 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H2M") は 3720 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("P1D") は一日の秒数の総数である 86400 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("-P1DT1M") は -86460 を返します。

xs:dateTime または xs:dateReturn から週数を返す XP3.1 XQ3.1

これらの関数は、xs:dateTime または xs:date から曜日を(整数として)を返します。曜日は(米国式フォーマットを使用して) 1 から 7 と番号づけられています。 日曜=1 と番号付けられます。欧州のフォーマットでは月曜 (=1) として番号付けられます。日曜=1 である米国フォーマットは整数 0 がフォーマットを表示するために使用できる箇所で設定することができます。

weekday-from-dateTime [altova:]

altova:weekday-from-dateTime(DateTime *as xs:dateTime*) を **xs:integer** とする **XP3.1** **XQ3.1**

時刻付きの日付を単一の引数として、この日付の曜日を正数として返します。曜日は 日曜=1 から開始して番号を付けます。(月曜=1 とする) ヨーロッパのフォーマットが必要な場合、この関数の他の署名を使用します (*下の次の署名を参照*)。

サンプル

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00")) は月曜を表示する 2 を返します。

altova:weekday-from-dateTime(DateTime *as xs:dateTime*, **Format** *as xs:integer*) **asxs:integer** とする **XP3.1** **XQ3.1**

時間月の日付を最初の引数として、この日付の曜日を正数として返します。曜日は 月曜=1 から開始して番号を付けます。第 2 の引数(整数)が 0 の場合、日曜=1 から開始して、曜日は 1 から 7 と番号を付けます。第 2 の引数が 0 以外の整数の場合、月曜=1 です。第 2 の引数がない場合、関数はこの関数の他の署名を持つと読み込まれます (*前の次の署名を参照*)。

サンプル

- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 1) は月曜を表示する 1 を返します。
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 4) は月曜を表示する 1 を返します。
- **altova:weekday-from-dateTime**(xs:dateTime("2014-02-03T09:00:00"), 0) は月曜を表示する 2 を返します。

weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date*) を **xs:integer** とする **XP3.1** **XQ3.1**

日付を単一の引数として、この日付の曜日を正数として返します。曜日は 日曜=1 から開始して番号を付けます。(月曜=1 とする) ヨーロッパのフォーマットが必要な場合、この関数の他の署名を使用します (*下の次の署名を参照*)。

サンプル

- **altova:weekday-from-date**(xs:date("2014-02-03+01:00")) は月曜を表示する 2 を返します。

altova:weekday-from-date(Date *as xs:date*, **Format** *as xs:integer*) を **xs:integer** とする
XP3.1 **XQ3.1**
日付を最初の引数とし、この日付の曜日を正数として返します。曜日は 月曜=1 から開始して番号を付けます。第 2 (フォーマット) 引数が 0, の場合、日曜=1 から開始し、曜日は 1 から 7 で番号付けられます。第 2 の引数がない場合、関数はこの関数の他の署名を持つと読み込まれます (*前の署名を参照*)。

サンプル

- **altova:weekday-from-date**(xs:date("2014-02-03"), 1) は月曜を示す 1 を返します。
- **altova:weekday-from-date**(xs:date("2014-02-03"), 4) は月曜を示す 1 を返します。
- **altova:weekday-from-date**(xs:date("2014-02-03"), 0) は月曜を示す 2 を返します。

[トップ 330]

xs:dateTime または xs:date から週数を返す **XP2** **XQ1** **XP3.1** **XQ3.1**

これらの関数は週数(整数として)を xs:dateTime から xs:date. から返します。週の番号付けは、米国、欧州、イスラムのカレンダーフォーマットで使用することができます。週の始まりが異なるため、週数の番号付けは、カレンダーのフォーマットにより異なります。(米国フォーマットでは、日曜、欧州フォーマットでは月曜、イスラムフォーマットでは土曜が週の開始日です)。

weeknumber-from-date [altova:]

altova:weeknumber-from-date(Date *as xs:date*, **Calendar** *as xs:integer*) を **xs:integer** とする
XP2 **XQ1** **XP3.1** **XQ3.1**
正数として提出された Date 引数の週数を返します。第 2 の引数(カレンダー)は続くカレンダーシステムを指定します。
サポートされる カレンダー の値は次のとおりです:

- 0 = US 米国のカレンダー(週の始まりは日曜日)
- 1 = ISO 標準、 欧州のカレンダー(週の始まりは月曜日)
- 2 = イスラムのカレンダー(週の始まりは土曜日)

デフォルトは 0 です。

サンプル

- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 0) は 13 を返します。
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 1) は 13 を返します。
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 2) は 13 を返します。
- **altova:weeknumber-from-date**(xs:date("2014-03-23")) は 13 を返します。

上のサンプル(2014-03-23) の date の曜日は日曜日です。米国およびイスラムのカレンダーは欧州カレンダーのこの日付よりも一週間先です。

weeknumber-from-dateTime [altova:]

altova:weeknumber-from-dateTime(DateTime *as xs:dateTime*, **Calendar** *as xs:integer*) を **xs:integer** とする **XP2** **XQ1** **XP3.1** **XQ3.1**
正数として提出された DateTime 引数の週数を返します。第 2 の引数(カレンダー)は続くカレンダーシステムを指定します。

サポートされるカレンダー の値は次のとおりです:

- 0 = US 米国のカレンダー（週の始まりは日曜日）
- 1 = ISO 標準、 欧州のカレンダー（週の始まりは月曜日）
- 2 = イスラムのカレンダー（週の始まりは土曜日）

Default is 0.

サンプル

- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 0) は13 を返します。
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 1) は13 を返します。
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 2) は13 を返します。
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00")) は13 を返します。

上のサンプル（2014-03-23T00:00:00）のdateTime の曜日は日曜日です。米国およびイスラムのカレンダーは欧州カレンダーのこの日付より先一週間先です。

[トップ 330]

各型の構文コンポーネントから日付、時刻、期間の型を構築する XP3.1 XQ3.1

関数は、xs:date, xs:time または xs:duration の構文コンポーネントを入力引数とし、引数を結合させて対応するデータ型を構築します。

build-date [altova:]

altova:build-date(Year *as xs:integer*, **Month** *as xs:integer*, **Date** *as xs:integer*) を **xs:date** とする XP3.1 XQ3.1

第 1 、第 2 、第3 の引数は、それぞれ、年、月、日を表します。xs:date 型の値を構築するため結合されます。整数の値は、特定の日付の一部の適正な範囲内である必要があります。例えば第 2 の引数（月の部分）は12 以上であってはなりません。

サンプル

- **altova:build-date**(2014, 2, 03) は2014-02-03 を返します。

build-time [altova:]

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) を **xs:time** とする XP3.1 XQ3.1

第 1 、第 2 、第3 引数はそれぞれ時間数（0 から23）、分数（0 から59）、および秒数（0 から59）の値です。これらの値は、xs:time 型の値を作成するために結合されます。整数の値は、それぞれの部分の正しい範囲内である必要があります。例えば、秒（分）引数は、59 以上であってはなりません。値にタイムゾーンを追加するには、この関数の他の署名を使用してください（*次の署名を参照*）。

サンプル

- **altova:build-time**(23, 4, 57) は23:04:57 を返します。

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*,

TimeZone *as xs:string*) を**xs:time** とする **XP3.1** **XQ3.1**
第 1 、第 2 、第 3 引数はそれぞれ時間数（0 から23）、分数（0 から59）、および秒数（0 から59）の値です。第四の引数は、値の一部としてタイムゾーンを与えます。4 つの引数が結合され、xs:time 型の値を構築します。例えば、第 2 の引数（分数）は 59 以上であってはなりません。

サンプル

- **altova:build-time**(23, 4, 57, '+1') は23:04:57+01:00 を返します。

build-duration [altova:]

altova:build-duration(Years *as xs:integer*, **Months** *as xs:integer*) を **xs:yearMonthDuration** とする **XP3.1** **XQ3.1**
xs:yearMonthDuration 型の値を構築するためには2 つの引数が必要です。最初の引数は期間の 年数 値の部分を与え、第 2 の引数は、月数 値の部分を与えます。第 2 の引数（月数）が12と同じまたはより大きくなると、整数は12 により分割されます。商は、年数 の部分を表す 最初の引数に加算され、（除算の）残りは月の部分は 月数の部分を表すために使用されます。期間の xs:dayTimeDuration 型を作成するには、次の署名を参照してください。

サンプル

- **altova:build-duration**(2, 10) はP2Y10M を返します。
- **altova:build-duration**(14, 27) はP16Y3M を返します。
- **altova:build-duration**(2, 24) はP4Y を返します。

altova:build-duration(Days *as xs:integer*, **Hours** *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) を**xs:dayTimeDuration** とする **XP3.1** **XQ3.1**
xs:dayTimeDuration 型の値を構築するためには4 つの引数が必要です。最初の引数は期間の 日数 値の部分で、第 2 、第 3 、第 4 引数は、それぞれ期間の時間数、分数、秒数 値のです。これらの3 つの時間 引数は、次の高い単位の値に加算され、結果は期間の全体の値を計算するために使用されます。例えば、72 秒は、1M+12S（1 分 と12 秒）に変換され、この値が期間全体の値を計算するために使用されます。xs:yearMonthDuration 型の期間を構築するためには、次の署名を参照してください。

サンプル

- **altova:build-duration**(2, 10, 3, 56) はP2DT10H3M56S を返します。
- **altova:build-duration**(1, 0, 100, 0) はP1DT1H40M を返します。
- **altova:build-duration**(1, 0, 0, 3600) はP1DT1H を返します。

[トップ 330]

文字列入力から 日付、日付時刻 または 時刻 を構築する **XP2** **XQ1** **XP3.1** **XQ3.1**

関数は、文字列を引数として、xs:date、xs:dateTime または xs:time データ型を構築します。文字列は、データ型のコンポーネントを提出されたパターン引数をベースにして分析されます。

parse-date [altova:]

altova:parse-date(Date *as xs:string*, **DatePattern** *as xs:string*) を**xs:date** とする **XP2** **XQ1** **XP3.1** **XQ3.1**
Date 入力文字列を**xs:date** の値として返します。第二の引数である**DatePattern** は、入力文字列のパターン（コンポーネントのシーケンス）を指定します。**DatePattern** は、下にリストされるコンポーネント指定子および任意の文字であるコンポーネントセパレータと共に説明されています。下のサンプルを参照してください。

D 日付

M 月

Y 年

DatePattern のパターンは、**Date** のパターンに一致する必要があります。出力は **xs:date** 型であるため、出力は常に **YYYY-MM-DD** 構文フォーマットになります。

サンプル

- **altova:parse-date**(xs:string("06-03-2014"), "[D]-[M]-[Y]") は 2014-03-06 を返します。
- **altova:parse-date**(xs:string("06-03-2014"), "[M]-[D]-[Y]") は 2014-03-06 を返します。
- **altova:parse-date**("06/03/2014", "[M]/[D]/[Y]") は 2014-03-06 を返します。
- **altova:parse-date**("06 03 2014", "[M] [D] [Y]") は 2014-03-06 を返します。
- **altova:parse-date**("6 3 2014", "[M] [D] [Y]") は 2014-03-06 を返します。

parse-dateTime [altova:]

altova:parse-dateTime(DateTime *as xs:string*, **DateTimePattern** *as xs:string*) を **xs:dateTime** とする **XP2** **XQ1** **XP3.1** **XQ3.1**

DateTime 入力文字列を **xs:dateTime** の値として返します。第二の引数である **DateTimePattern** は、入力文字列のパターン（コンポーネントのシーケンス）を指定します。**DateTimePattern** は、下にリストされるコンポーネント指定子および任意の文字であるコンポーネント セパレータと共に説明されています。下のサンプルを参照してください。

D	日
M	月
Y	年
H	時間
m	分
s	秒

DateTimePattern のパターンは、**DateTime** のパターンに一致する必要があります。出力は **xs:dateTime** 型であるため、出力は常に **YYYY-MM-DDTHH:mm:ss** 構文フォーマットになります。

サンプル

- **altova:parse-dateTime**(xs:string("06-03-2014 13:56:24"), "[D]-[M]-[Y] [H]:[m]:[s]") は 2014-03-06T13:56:24 を返します。
- **altova:parse-dateTime**("time=13:56:24; date=06-03-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]") は 2014-03-06T13:56:24 を返します。

parse-time [altova:]

altova:parse-time(Time *as xs:string*, **TimePattern** *as xs:string*) を **xs:time** とする **XP2** **XQ1** **XP3.1** **XQ3.1**

Time 入力文字列を **xs:time** の値として返します。第二の引数である **TimePattern** は、入力文字列のパターン（コンポーネントのシーケンス）を指定します。**TimePattern** は、下にリストされるコンポーネント指定子および任意の文字であるコンポーネント セパレータと共に説明されています。下のサンプルを参照してください。

H	時間
m	分
s	秒

TimePattern のパターンは、**Time** のパターンに一致する必要があります。出力は **xs:time** 型であるため、出力は常に **YYYY-HH:mm:ss** 構文フォーマットになります。

サンプル

- **altova:parse-time**(xs:string("13:56:24"), "[H]:[m]:[s]") は13:56:24 を返します。
- **altova:parse-time**("13-56-24", "[H]-[m]") は13:56:00 を返します。
- **altova:parse-time**("time=13h56m24s", "time=[H]h[m]m[s]s") は13:56:24 を返します。
- **altova:parse-time**("time=24s56m13h", "time=[s]s[m]m[H]h") は13:56:24 を返します。

[トップ 330]

年齢に関連した関数 XP3.1 XQ3.1

関数は計算された年齢(i) 入力引数の日付と現在の日付の期間 (ii) 2 つの入力引数の日付の期間 を返します。**altova:age** 関数は、年齢を年数で返します、**altova:age-details** は年齢を年数、月数、日数からなる3 つの整数のシーケンスで返します。

age [altova:]

altova:age(StartDate *as xs:date***)** を**xs:integer** とする XP3.1 XQ3.1

引数として提出された開始日から(システムクロックから取得された)現在の日付 までの日数を数えて、あるオブジェクトの年齢の年数である正数を返します。入力引数が1 年より先大きい場合、または一年の場合、返される値は負の数です。

サンプル

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2013-01-15")) は1 を返します。
- **altova:age**(xs:date("2013-01-16")) は0 を返します。
- **altova:age**(xs:date("2015-01-15")) は-1 を返します。
- **altova:age**(xs:date("2015-01-14")) は0 を返します。

altova:age(StartDate *as xs:date*, **EndDate** *as xs:date***)** を**xs:integer** とする XP3.1 XQ3.1

最初の引数として提出された開始日から第 2 の引数の終了日 までの日数を数えて、あるオブジェクトの年齢の年数である正数を返します。最初の引数が1 年または第 2 の引数より先後の日付の場合、返される値は負の数です。

サンプル

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2000-01-15"), xs:date("2010-01-15")) は10 を返します。
- **altova:age**(xs:date("2000-01-15"), current-date()) は、現在の日付が2014-01-15 の場合 14 を返します。
- **altova:age**(xs:date("2014-01-15"), xs:date("2010-01-15")) は-4 を返します。

age-details [altova:]

altova:age-details(InputDate *as xs:date***)** を(**xs:integer**)* とする XP3.1 XQ3.1

引数とシステムクロックから取得された)現在の日付として提出された日付の間の年数、月数、日数の3 つの整数を返します。返された years+months+days の合計は、2 つの日付(入力の日付と現在の日付)の時間差です。 入力の日付は現在の日付より先早いまたはお遅い値をもつことができますが、入力の日付が早いか遅いかは返される値で示されていません。戻される値は常に正の数です。

サンプル

現在の日付が**2014-01-15** の場合:

- **altova:age-details**(xs:date("2014-01-16")) は (0 0 1) を返します。
- **altova:age-details**(xs:date("2014-01-14")) は (0 0 1) を返します。
- **altova:age-details**(xs:date("2013-01-16")) は (1 0 1) を返します。
- **altova:age-details**(current-date()) は (0 0 0) を返します。

altova:age-details(Date-1 *as xs:date*, **Date-2** *as xs:date*) を (**xs:integer**)* とする **XP3.1** **XQ3.1**
二つの引数間の年数、月数、日数の 3 つの整数を返します。 返された years+months+days の合計は、2 つの入力の日付の時間差です。最初の引数として提出される二つの日付はどちらが速くても、または遅くてもかまいません。返される値は入力の日付が現在の日付より早いかまたは遅いかを示しません。戻される値は常に正の数です。

サンプル

- **altova:age-details**(xs:date("2014-01-16"), xs:date("2014-01-15")) は (0 0 1) を返します。
- **altova:age-details**(xs:date("2014-01-15"), xs:date("2014-01-16")) は (0 0 1) を返します。

[トップ 330]

8.2.1.3 XPath/XQuery 関数: 位置情報

以下の位置情報 XPath/XQuery 拡張関数は、RaptorXML Server の現在のバージョンによりサポートされています。また、次で使用することができます: (i) XSLT コンテキスト内の XPath 式、または (ii) XQuery ドキュメント内の XQuery 式。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は **Altova 拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることに注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内の XPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内の XQuery 式で使用):	**XQ1** **XQ3.1**

format-geolocation [altova:]

altova:format-geolocation(Latitude *as xs:decimal*, **Longitude** *as xs:decimal*, **GeolocationOutputStringFormat** *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
緯度と経度を最初の 2 つの引数とし、位置情報を文字列として出力します。第 3 の引数 **GeolocationOutputStringFormat** は、位置情報出力文字列のフォーマットです。出力文字列のフォーマットを識別するために、正数の値 1 から 4 を使用します。(*下の「位置情報出力文字列フォーマット」を参照してください*)。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西) です。

メモ image-exif-data 355 関数と Exif メタデータの @Geolocation 355 属性を位置情報入力文字列を提供する際に使用することができます。

サンプル

- **altova:format-geolocation**(33.33, -22.22, 4) は xs:string "33.33 -22.22" を返します。
- **altova:format-geolocation**(33.33, -22.22, 2) は xs:string "33.33N 22.22W" を返します。
- **altova:format-geolocation**(-33.33, 22.22, 2) は xs:string "33.33S 22.22E" を返します。
- **altova:format-geolocation**(33.33, -22.22, 1) は xs:string "33°19'48.00"S 22°13'12.00"E" を返します。

位置情報出力文字列フォーマット:

与えられた緯度と経度は下にリストされる出力フォーマットによりフォーマットされます。希望するフォーマットは整数 ID (1 から 4) により識別されます。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西) です。

1
度、分、10 進の秒、方角のサフィックス付き (N/S, E/W) **D°M'S.SS"N/S D°M'S.SS"E/W** *サンプル:* **33°55'11.11"N 22°44'66.66"W**

2
10 進の度、方角のサフィックス付き (N/S, E/W) **D.DDN/S D.DDE/W** *サンプル:* **33.33N 22.22W**

3
度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。 **+/-D°M'S.SS" +/-D°M'S.SS"** *サンプル:* **33°55'11.11" -22°44'66.66"**

4
10 進の度、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。 **+/-D.DD +/-D.DD** *サンプル:* **33.33 -22.22**

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つの Exif タグの連結です: 単位の追加された (*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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*) を **xs:decimal+** とする **XP3.1 XQ3.1**

GeolocationInputString 引数を解析して、位置情報の緯度と経度（この通りの順番）を 2 つの xs:decimal アイテムのシーケンスとして返します。位置情報入力文字列が提供されることのできるフォーマットは以下のリストの通りです。

メモ image-exif-data 355 関数と Exif メタデータの @Geolocation 355 属性を位置情報入力文字列を提供する際に使用することができます。

サンプル

- **altova:parse-geolocation**("33.33 -22.22") は 2 つの xs:decimals (33.33, 22.22) のシーケンスを返します。
- **altova:parse-geolocation**("48°51'29.6""N 24°17'40.2""") は 2 つの xs:decimals (48.8582222222222, 24.2945) のシーケンスを返します。
- **altova:parse-geolocation**('48°51''29.6"N 24°17''40.2"') は 2 つの xs:decimals (48.8582222222222, 24.2945) のシーケンスを返します。
- **altova:parse-geolocation**(**image-exif-data**(//MyImages/Image20141130.01)**/@Geolocation**) は 2 つの xs:decimals のシーケンスを返します。

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度（この通りの順番）を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が 1 つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は +90 から -90 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルにしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い (") でハイライトされており、エスケープした単位インジケーターは青い ("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き (+/-); (**N/S E/W**) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つの Exif タグの連結です: 単位の追加された (*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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*) を **xs:decimal** とする **XP3.1** **XQ3.1**
2 つの位置情報の間の距離をキロメートルで計算します。位置情報入力文字列を提供することのできるフォーマットは下にリストされています。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西)です。

メモ image-exif-data 355 関数と Exif メタデータの @Geolocation 355 属性を位置情報入力文字列を提供する際に使用することができます。

サンプル

- **altova:geolocation-distance-km**("33.33 -22.22", "48°51'29.6""N 24°17'40.2""") は xs:decimal 4183.08132372392 を返します。

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度 (この通りの順番) を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が 1 つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は +90 から -90 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い (") でハイライトされており、エスケープした単位インジケーターは青い ("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)

D°M'S.SS"N/S D°M'S.SS"W/E
サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き (+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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*) を **xs:decimal** とする **XP3.1** **XQ3.1**
2 つの位置情報の間の距離をマイルで計算します。位置情報入力文字列を提供することのできるフォーマットは下にリストされています。緯度の値の範囲は+90 から-90 (北 から 南) です。経度の値の範囲は+180 から-180 (東 から 西)です。

メモ image-exif-data 355 関数とExif メタデータの @Geolocation 355 属性を位置情報入力文字列を提供する際に使用することができます。

サンプル

- **altova:geolocation-distance-mi**("33.33 -22.22", "48°51'29.6""N 24°17'40.2""")
  は xs:decimal 2599.40652340653 を返します。

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度（この通りの順番）を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が1つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は+90 から-90 (N からS)。経度の値の範囲は+180 から-180 (E からW)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い(") でハイライトされており、エスケープした単位インジケーターは青い("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き（N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き（+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き（N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き（+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き（N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き（+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つのExif タグの連結です: 単位の追加された（*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef。

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

geolocations-bounding-rectangle [altova:]

altova:geolocations-bounding-rectangle(GeolocationsOutputStringFormat を *xs:integer* として) を **xs:string** として **XP3.1** **XQ3.1**

最初の引数として文字列のシーケンスを取ります。シーケンスの各文字列が位置情報です。関数はそれぞれが、最初の引数に送信されたすべての位置情報を含むために最適にサイズ調整された長方形の左上と右下の位置情報の座標である2つの文字列のシーケンスを返します。位置情報入力文字列が表示される書式は以下にリストされています(*「位置情報入力文字列書式」を参照してください*)。緯度の値は+90 から-90 への(N からS への) の範囲です。経度の値は+180 から-180 への(E からW への) の範囲です。

関数の2番目の引数は出力シーケンス内の2つの位置情報文字列の書式を指定します。引数はそれぞれの値が異なる位置情報文字列書式を識別する1 から4 の整数の値を取ります(*下記の「位置情報出力文字列書式」を参照してください*)。

メモ image-exif-data 355 関数とExif メタデータの属性は入力文字列を提供するために使用することができます。

サンプル

- **altova:geolocations-bounding-rectangle**(("48.2143531 16.3707266", "51.50939 -0.11832"), 1) はシーケンス("51°30'33.804"N 0°7'5.952"W", "48°12'51.67116"N 16°22'14.61576"E")を返します。
- **altova:geolocations-bounding-rectangle**(("48.2143531 16.3707266", "51.50939 -0.11832", "42.5584577 -70.8893334"), 4) はシーケンス("51.50939 -70.8893334", "42.5584577 16.3707266")を返します

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度(この通りの順番) を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が1つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は+90 から-90 (N からS)。経度の値の範囲は+180 から-180 (E からW)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い(") でハイライトされており、エスケープした単位インジケーターは青い("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き(N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き(+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き(+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'

サンプル: +33°55.55' -22°44.44'

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: 33.33N 22.22W

- 10 進の度、プレフィックスサイン付き (+/-); (N/S E/W) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: 33.33 -22.22

フォーマットの組み合わせのサンプル:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

位置情報出力文字列フォーマット:

与えられた緯度と経度は下にリストされる出力フォーマットによりフォーマットされます。希望するフォーマットは整数 ID (1 から 4) により識別されます。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西) です。

1
度、分、10 進の秒、方角のサフィックス付き (N/S, E/W) D°M'S.SS"N/S D°M'S.SS"E/W *サンプル:* 33°55'11.11"N 22°44'66.66"W

2
10 進の度、方角のサフィックス付き (N/S, E/W) D.DDN/S D.DDE/W *サンプル:* 33.33N 22.22W

3
度、分、10 進の秒、プレフィックスサイン付き (+/-); (N/E) のためのプラスサインは任意です。 +/-D°M'S.SS" +/-D°M'S.SS" *サンプル:* 33°55'11.11" -22°44'66.66"

4
10 進の度、プレフィックスサイン付き (+/-); (N/E) のためのプラスサインは任意です。 +/-D.DD +/-D.DD *サンプル:* 33.33 -22.22

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 Geolocationを標準 Exif メタデータ タグから生成します。Geolocation は、4 つのExif タグの連結です: 単位の追加された (*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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*)+)) を**xs:boolean** とする **XP3.1** **XQ3.1**

PolygonPoint 引数により説明されている**Geolocation**（最初の引数）が多角形のエリア内に存在するかを決定します。もし、PolygonPoint 引数が最初と最後のポイントが同じ場合に作成される）閉じられたフィギュアを作成しない場合、フィギュアを閉じるために、最初のポイントが明示的に最後のポイントとして追加されます。全ての引数（Geolocation および PolygonPoint+）は、（*下にリストされるフォーマットの*）位置情報入力文字列により提供されます。Geolocation 引数が多角形エリア内にある場合、関数は true(); を返します。その他の場合は false() を返します。緯度の値の範囲は +90 から -90（北 から 南）です。経度の値の範囲は +180 から -180（東 から 西）です。

メモ image-exif-data 355 関数と Exif メタデータの @Geolocation 355 属性を位置情報入力文字列を提供する際に使用することができます。

サンプル

- **altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48 24", "58 -32"))** は true() を返します。
- **altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48 24"))** は true() を返します。
- **altova:geolocation-within-polygon("33 -22", ("58 -32", "-78 -55", "48° 51'29.6""N 24°17'40.2""")))** は true() を返します。

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度（この通りの順番）を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が１つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は +90 から -90（N から S）。経度の値の範囲は +180 から -180（E から W）。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い（"）でハイライトされており、エスケープした単位インジケーターは青い（""）でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き（N/S, E/W）
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き（+/-）; （**N/E**）のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き（N/S, E/W）
  D°M.MM'N/S D°M.MM'W/E

サンプル: 33°55.55'N 22°44.44'W

- 度、分、10 進の分、プレフィックスサイン付き(+/-); (N/E) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: +33°55.55' -22°44.44'

- 10 進の度、方角のサフィックス付き(N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: 33.33N 22.22W

- 10 進の度、プレフィックスサイン付き(+/-); (N/S E/W)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: 33.33 -22.22

フォーマットの組み合わせのサンプル:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 Geolocationを標準 Exif メタデータ タグから生成します。Geolocation は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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*) を**xs:boolean** とする **XP3.1** **XQ3.1**
長方形の対格の角を指定する第 2 及び第 3 引数(RectCorner-1、RectCorner-2)により説明されている Geolocation (最初の引数) が長方形のエリア内に存在するかを決定します。全ての引数(Geolocation、RectCorner-1 および RectCorner-2) は、(*下にリストされるフォーマットの*) 位置情報入力文字列により提供されます。Geolocation 引数が長方形エリア内にある場合、関数は true(); を返します。その他の場合は false() を返します。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西)です。

メモ image-exif-data 355 関数とExif メタデータの @Geolocation 355 属性を位置情報入力文字列を提供する際に使用することができます。

サンプル

- altova:geolocation-within-rectangle("33 -22", "58 -32", "-48 24") は true() を返します。
- altova:geolocation-within-rectangle("33 -22", "58 -32", "48 24") は false() を返します。
- altova:geolocation-within-rectangle("33 -22", "58 -32", "48°51'29.6""S 24°

17'40.2"""") は true() を返します。

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度（この通りの順番）を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が1つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は +90 から -90 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い (") でハイライトされており、エスケープした単位インジケーターは青い ("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き (+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つの Exif タグの連結です: 単位の追加された（*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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

[トップ 344]

8.2.1.4 XPath/XQuery 関数: イメージに関連

以下のイメージに関連したXPath/XQuery 拡張関数は、RaptorXML Server の現在のバージョンによりサポートされています。また、次で使用することができます: (i) XSLT コンテキスト内のXPath 式、または(ii) XQuery ドキュメント内のXQuery 式。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることに注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

suggested-image-file-extension [altova:]

altova:suggested-image-file-extension(Base64String *as string*) を **string?** とする XP3.1 XQ3.1

イメージファイルのBase64 エンコード を引数として、イメージのファイル拡張子を、イメージのBase64エンコード内の記録として返します。返された値は、エンコード内で使用することのできるイメージ型情報を基にしたヒントです。この情報が使用できない場合、空の文字列が返されます。この関数は、Base64 イメージをファイルとして保存し、適切なファイル拡張子を動的取得する際に役に立ちます。

サンプル

- **altova:suggested-image-file-extension**(/MyImages/MobilePhone/Image20141130.01) は 'jpg' を返します。
- **altova:suggested-image-file-extension**($XML1/Staff/Person/@photo) は '' を返します。

上のサンプルでは、関数の引数として与えられたノードはBase64 エンコードイメージを含むと仮定します。最初のサンプルは jpg をファイルの型および拡張子として取得します。二番目のサンプルでは、与えられたBase64 エンコードは使用できる拡張子の情報を提供しません。

mt-transform-image [altova:]

altova:mt-transform-image(Base64Image *as Base64BinaryString*, **Size** *as item()+*, **Rotation** *as xs:integer*, **Quality** *as xs:integer*) を **Base64BinaryString** とする XP3.1 XQ3.1

Base64-エンコード イメージを最初の引数として、変換されたBase64-エンコード イメージを返します。第 2、第 3、第 4 引数は変換された以下のイメージパラメーターです: サイズ、回転、およびクオリティ。

- サイズ引数には3 つのサイズ変更のオプションがあります。

(X,Y)	絶対ピクセルの値。アスペクト率は保持されません。高さと幅は自動的にイメージの長いおよび短い辺に逢わされるため、高さと幅の指示は関係ありません。2 つの整数アイテムのシーケンスとして値が入力されます。かっこが必要です。
X	X をピクセルの新しい長い辺として、イメージの縦横比のサイズ変更が行われます。アスペクト率は保持されます。値は整数で引用符なしで入力されます。
'X%'	元のディメンションの与えられたパーセンテージにイメージのサイズを変更します。値は文字列として引用符を付けて入力される必要があります。

- 回転は以下の値を持つことができます: 90、180、270、-90、-180、-270。これらの値は回転の度数です。正の値はイメージを時計回りに回転させます。負の値はイメージを反時計回りに回転させます。 Altova Exif 属性 **OrientationDegree** を使用して、イメージの現在の回転の度数(0, 90, 180, 270) をイメージの Exif **Orientation** タグから取得することができます。ですが、**OrientationDegree** 属性はデータの **Orientation** タグから取得されるため、Exif データ内に **Orientation** タグが存在する場合のみ使用することができます。(*下の OrientationDegree 説明を参照してください*)。
- クオリティは、0 から 100 の値で、JPEG 圧縮の IJG クオリティスケールの値を参照していますが、クオリティのパーセンテージのインジケーターではありません。サイズとクオリティのどちらかを優先すると、もう一方の優先度が下がります。フルカラーソースのイメージでは、75 が通常最高値と見なされています。もし、75 が満足のいく結果をもたらさない場合は、値を上げてください。

メモ Exif データが元のイメージに存在する場合、変換時に削除され、変換されたイメージには Exif データは存在しません。

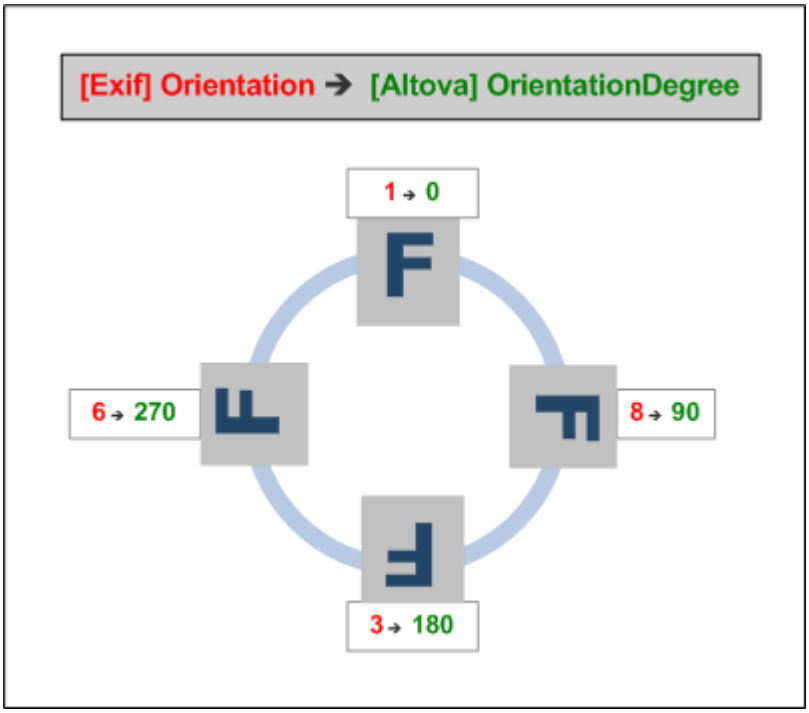
サンプル

- **mt-transform-image**(Images/Image[@id='43'], '50%', 90, 75)
  関数は、43 の @id 値を持つイメージ子孫ノード内で Base64-エンコード 文字列として保管されているイメージを入力とします。関数は変換されたイメージを返します。変換されたイメージは 50% まで、サイズ変更され、時計回りに 90 度回転され、75 のクオリティレベルを与えられます。
- **mt-transform-image**(Images/Image[@id='43'], 400, 90, 75)
  関数は前のサンプルと同じ結果を出しますが、長い辺は 400 ピクセルの特定の値に設定されています。元のイメージのアスペクト率は保持されます。
- **mt-transform-image**(Images/Image[@id='43'], (400, 280), image-exif-data($XML1/$XML1/Images/ReferenceImage)/@OrientationDegree, 75)
  このサンプルは、前のサンプルと同じイメージを選択し、同じクオリティ値 (75) を設定します。イメージのサイズは 400x280 ピクセルに設定されています。回転値は、ノード内の Base64-エンコード イメージの **@OrientationDegree** 属性から得られます。

Altova Exif 属性: OrientationDegree

Altova XPath/XQuery エンジンはカスタム属性 **OrientationDegree** を Exif メタデータタグ **Orientation** から生成します。

OrientationDegree は標準 Exif タグ Orientation を正数の値 (1、8、3 または 6) からそれぞれ対応する度数の値 (0, 90, 180, 270) へ下の図に示されているように変換します。2、4、5、7 の Orientation 値の変換はないことに注意してください。(これらの向きはイメージ 1 を垂直方向中央軸で反転して、2 の値を持つイメージを取得します。そして、このイメージを 90-度ごとに時計回りにジャンプさせそれぞれ 7、4、および 5 の値を取得します。)。



標準 Exif メタタグ

- 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
- 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

image-exif-data [altova:]

altova:image-exif-data(Base64BinaryString *as string*) を **element?** とする **XP3.1** **XQ3.1**

Base64 エンコード JPEG イメージを引数として、イメージの Exif メタデータを含む **Exif** という名の要素を返します。Exif メタデータは `Exif` 要素の属性の値ペアとして作成されます。属性名は、Base64 エンコード内で検出された Exif データタグです。Exif 仕様タグのリストは以下の通りです。ベンダー特有のタグが Exif データ内に存在する場合、タグとその値も属性値のペアとして返されます。標準 Exif メタデータタグに追加して（*下のリスト参照*）Altova 特有の属性値のペアもまた生成されます。これらの Altova Exif 属性は以下のとおりです。

サンプル

- 属性にアクセスするには、以下の関数を使用します:
  `image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude`
  `image-exif-data(//MyImages/Image20141130.01)/@Geolocation`
- 全ての属性にアクセスするには、以下の関数を使用します:
  `image-exif-data(//MyImages/Image20141130.01)/@*`
- 全ての属性の名前にアクセスするには、以下の式を使用します:
  `for $i in image-exif-data(//MyImages/Image20141130.01)/@* return name($i)`
  関数により返される属性の名前を検出するために役に立ちます。

Altova Exif 属性: 位置情報

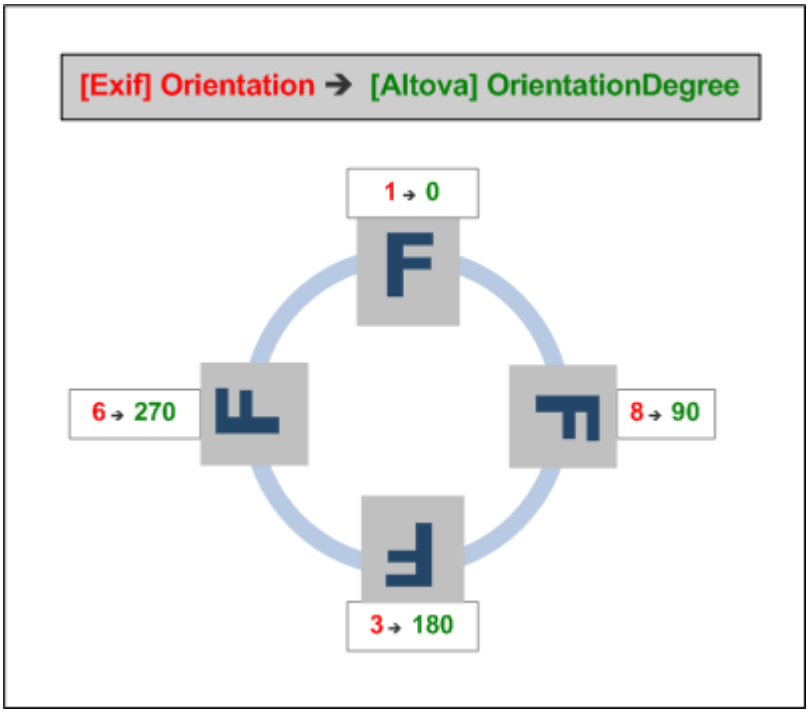
Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つの Exif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

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

Altova Exif 属性: OrientationDegree

Altova XPath/XQuery エンジンはカスタム属性 **OrientationDegree** を Exif メタデータタグ **Orientation** から生成します。

OrientationDegree は標準 Exif タグ Orientation を正数の値 (1、8、3 または 6) からそれぞれ対応する度数の値 (0, 90, 180, 270) へ下の図に示されているように変換します。2、4、5、7 の Orientation 値の変換はないことに注意してください。(これらの向きはイメージ 1 を垂直方向中央軸で反転して、2 の値を持つイメージを取得します。そして、このイメージを 90-度ごとに時計回りにジャンプさせそれぞれ 7、4、および 5 の値を取得します。)。



標準 Exif メタデータ タグのリスト

- 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
- 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

[トップ 355]

8.2.1.5 XPath/XQuery 関数: 数値

Altova の数値拡張関数は XPath と XQuery 内で使用することができ、データを更に処理するための追加機能を提供します。このセクションの関数は Altova の **XPath 3.0** と **XQuery 3.0** エンジンと使用することができます。XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

自動付番関数

generate-auto-number [altova:]

altova:generate-auto-number(ID *as xs:string*, **StartsWith** *as xs:double*, **Increment** *as xs:double*, **ResetOnChange** *as xs:string*) を xs:integer とする XP1 XP2 XQ1 XP3.1 XQ3.1

関数が呼び出される度に番号を生成します。関数が初めて呼び出される際に生成される最初の番号は StartsWith 引数により指定されます。その後の関数の呼び出しは新しい番号を生成します。この番号は前に生成された番号を、Increment 引数で指定された値ごとにインクリメントします。実質的には、altova:generate-auto-number 関数は、ID 引数により指定されたカウンターを作成し、このカウンターが関数が呼び出されるごとにインクリメントされます。ResetOnChange 引数の値が、前の関数呼び出しから変更されると、生成される番号の値が StartsWith 値にリセットされます。自動付番は altova:reset-auto-number [396] 関数を使用してリセットすることができます。

サンプル

- **altova:generate-auto-number**("ChapterNumber", 1, 1, "SomeString") は、関数が呼び出される度に番号を 1 つ返します。1 から始まり、関数が呼び出される度に 1 ずつインクリメントします。その後の呼び出しの 4 番目の"SomeString"引数がである限り、インクリメントは継続されます。4 番目の引数の値が変更されると、(ChapterNumber と呼ばれる) カウンターは 1 にリセットされます。ChapterNumber の値も altova:reset-auto-number [396] 関数の呼び出しによりの altova:reset-auto-number [396] ("ChapterNumber") ようにリセットされます。

reset-auto-number [altova:]

altova:reset-auto-number(ID *as xs:string*) XP1 XP2 XQ1 XP3.1 XQ3.1

この関数は、ID 引数内で名づけられた自動付番カウンターの番号をリセットします。引数内のカウンター名を作成した関数の引数により指定された数値にリセットされます。altova:generate-auto-number [396]

サンプル

- **altova:reset-auto-number**("ChapterNumber") は、altova:generate-auto-number [396] 関数により作成された ChapterNumber という名の自動付番カウンターをリセットします。
- ChapterNumber を作成した altova:generate-auto-number [396] 関数の StartsWith 引数の値に数値をリセットします。

[[トップ] [363]]

数値関数

hex-string-to-integer [altova:]

altova:hex-string-to-integer(HexString *as xs:string*) を **xs:integer** とする **XP3.1** **XQ3.1**
10 進のシステム(10進)内の正数の16 進の同値の文字列引数を必要とし、10 進の整数を返します。

サンプル

- **altova:hex-string-to-integer**('1') は1 を返します。
- **altova:hex-string-to-integer**('9') は9 を返します。
- **altova:hex-string-to-integer**('A') は10 を返します。
- **altova:hex-string-to-integer**('B') は11 を返します。
- **altova:hex-string-to-integer**('F') は15 を返します。
- **altova:hex-string-to-integer**('G') はエラーを返します。
- **altova:hex-string-to-integer**('10') は16 を返します。
- **altova:hex-string-to-integer**('01') は1 を返します。
- **altova:hex-string-to-integer**('20') は32 を返します。
- **altova:hex-string-to-integer**('21') は33 を返します。
- **altova:hex-string-to-integer**('5A') は90 を返します。
- **altova:hex-string-to-integer**('USA') はエラーを返します。

integer-to-hex-string [altova:]

altova:integer-to-hex-string(Integer *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
正数を引数として必要とし、文字列として自身のベース16 の同値を返します。

サンプル

- **altova:integer-to-hex-string**(1) は'1' を返します。
- **altova:integer-to-hex-string**(9) は'9' を返します。
- **altova:integer-to-hex-string**(10) は'A' を返します。
- **altova:integer-to-hex-string**(11) は'B' を返します。
- **altova:integer-to-hex-string**(15) は'F' を返します。
- **altova:integer-to-hex-string**(16) は'10' を返します。
- **altova:integer-to-hex-string**(32) は'20' を返します。
- **altova:integer-to-hex-string**(33) は'21' を返します。
- **altova:integer-to-hex-string**(90) は'5A' を返します。

[[トップ] 363]

数値フォーマット関数

[[トップ] 363]

8.2.1.6 XPath/XQuery 関数: スキーマ

下にリストされる Altova 拡張関数はスキーマの情報を返します。以下は関数の詳細 (i) サンプル(ii) スキーマコンポーネントのリストとパースペクティブのプロパティです。 Altova の**XPath 3.0** と**XQuery 3.0** エンジンと共に使用することができ XPath/XQuery コンテキスト内で見つけることができます。

スキーマドキュメントからのスキーマ情報
関数 **altova:schema** には以下の2つの引数が存在します: 1つのゼロ引数と他の2つの引数。ゼロ引数関数はスキーマ全体を返します。この開始点からスキーマ内をナビゲートし必要とするスキーマコンポーネントをロケートします。2-引数関数は自身のQNameにより識別される特定のコンポーネントの型を返します。両方の場合、戻り値は関数です。返されたコンポーネント内をナビゲートするには、特定のコンポーネントのプロパティを選択する必要があります。プロパティが非動的アイテム(すなわちコンポーネントの場合) このコンポーネントのプロパティをさらに選択してナビゲートすることが可能です。選択されたプロパティが動的なアイテムの場合、アイテムの値が返され、ナビゲートを更に行うことはできません。

メモ Xquery 式ではスキーマは明示的にインポートされる必要があります。XPath 式では、スキーマは処理環境にインポートされる必要があります。例えば、XSLT には **xslt:import** 命令を使用してインポートします。

XML ノードからのスキーマ情報
関数 **altova:type** はXMLドキュメントのノードを送信しノードの型情報をPSVI から返します。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内の XPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内の XQuery 式で使用):	**XQ1** **XQ3.1**

▼ Schema (引数無し)

altova:schema() as (function(xs:string) as item()*)? **XP3.1** **XQ3.1**
schema コンポーネント全体を返します。**schema** コンポーネントを **schema** コンポーネントのプロパティの一つを選択して更にナビゲートすることができます。

- このプロパティがコンポーネントの場合、このコンポーネントのプロパティの1つを選択して更に深く他のステップをナビゲートすることができます。このステップは更にスキーマをナビゲートするために繰り返すことができます。
- コンポーネントが動的な値の場合、動的な値が返され更にナビゲートすることはできません。

schema コンポーネントのプロパティ:

```
"type definitions"
"attribute declarations"
"element declarations"
"attribute group definitions"
"model group definitions"
"notation declarations"
"identity-constraint definitions"
```

(**schema** 以外の)他のすべてのコンポーネント型のプロパティが下にリストされています。

メモ Xquery 式ではスキーマは明示的にインポートされる必要があります。XPath 式では、スキーマは処理環境にインポートされる必要があります。例えば、XSLT には **xslt:import** 命令を使用してインポートします。

⊟ *サンプル*

- `import schema "" at "C:\Test\ExpReport.xsd"; for $typedef in altova:schema() ("type definitions") return $typedef ("name")` はスキーマ内のすべての単純型または複合型の名前を返します
- `import schema "" at "C:\Test\ExpReport.xsd"; altova:schema() ("type definitions")[1]("name")` はスキーマ内の単純型または複合型の最初の名前を返します

コンポーネントとそのプロパティ

Assertion

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Assertion"
test	XPath プロパティレコード	

Attribute Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Declaration"
name	文字列	属性のローカル名
target namespace	文字列	属性の名前空間 URI
type definition	Simple Type または Complex Type	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	存在する場合、プロパティを持つ関数 (″class″: ″Value Constraint″、″variety″: ″fixed″ または″default″, ″value″: atomic value, ″lexical form″: string。″value″ プロパティは namespace-sensitive 型のために使用することはできないことに注意してください。	
inheritable	ブール値	

Attribute Group Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Group Definition"
name	文字列	属性グループのローカル名
target namespace	文字列	属性グループの名前空間 URI
attribute uses	(Attribute Use) のシーケンス	
attribute wildcard	任意の属性のワイルドカード	

Attribute Use

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Use"
required	ブール値	属性が必要な場合はtrue 、任意の場合はfalse
value constraint	Attribute Declaration を参照	
inheritable	ブール値	

Attribute Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″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	文字列 (″strict″|″lax″|″skip″)	

Complex Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
base type definition	Complex Type Definition	
final	文字列のシーケンス ("restriction"|"extension")	
context	空のシーケンス(not implemented)	
derivation method	文字列 (″restriction″|″extension″)	
abstract	ブール値	
attribute uses	Attribute Use のシーケンス	
attribute wildcard	任意の属性のワイルドカード	
content type	プロパティを持つ関数: (″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	文字列のシーケンス	

(C) 2015-2021 Altova GmbH

substitutions	("restriction"|"extension")	
assertions	アサーションのシーケンス	

Element Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
type definition	Simple Type または Complex Type	
type table	プロパティを持つ関数 (″class″:″Type Table″, "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	Attribute Declaration を参照	
nillable	ブール値	
identity-constraint definitions	Identity Constraint のシーケンス	
substitution group affiliations	Element Declaration のシーケンス	
substitution group exclusions	文字列のシーケンス ("restriction"|"extension")	
disallowed substitutions	文字列のシーケンス ("restriction"|"extension"|"substitution")	
abstract	ブール値	

Element Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″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	文字列 (″strict″|″lax″|″skip″)	

Facet

プロパティ名	プロパティ型	プロパティ値

kind	文字列	ファセットの名前、例えば、"minLength" または "enumeration"
value	ファセットによる	ファセットの値
fixed	ブール値	
typed-value	ファセットの列挙のみ、Array(xs:anyAtomicType	列挙値を含む配列。それぞれがアトミック値のシーケンスである場合があります。(メモ列挙ファセットに関しては、実際の型にかかわらず "value" 値プロパティは文字列のシーケンスです)

Identity Constraint

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Identity-Constraint Definition"
name	文字列	制約のローカル名
target namespace	文字列	制約の名前空間 URI
identity-constraint category	文字列 ("key"|"unique"|"keyRef")	
selector	XPath プロパティレコード	
fields	XPath プロパティレコード のシーケンス	
referenced key	(keyRef のためのみ): Identity Constraint	対応するキー制約

Model Group

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group"
compositor	文字列 ("sequence"|"choice"|"all")	
particles	Particle のシーケンス	

Model Group Definition

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group Definition"
name	文字列	モデルグループのローカル名
target namespace	文字列	モデルグループの名前空間 URI
model group	Model Group	

Notation

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Notation Declaration"
name	文字列	表記のローカル名

target namespace	文字列	表記の名前空間 URI
system identifier	anyURI	
public identifier	文字列	

Particle

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Particle"
min occurs	整数	
max occurs	整数または文字列("unbounded")	
term	Element Declaration、Element Wildcard、または ModelGroup	

Simple Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Simple Type Definition"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
final	文字列のシーケンス ("restriction"|"extension"|"list"|"union")	
context	含まれるコンポーネント	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	(実装されていない)空のシーケンス	
variety	文字列 ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(リスト型のみのための) Simple Type	
member type definitions	(ユニオン型のための) Simple Type のシーケンス	

Type Alternative

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Type Alternative"
test	XPath プロパティレコード	
type definition	Simple Type または Complex Type	

XPath Property Record

プロパティ名	プロパティ型	プロパティ値

namespace bindings	プロパティを持つ関数のシーケンス(″prefix″: string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	XPath 式の静的ベースURI
expression	文字列	文字列としてのXPath 式

▼ Schema (2つの引数)

altova:schema(ComponentKind *as xs:string*, **Name** *as xs:QName*) **as (function(xs:string) as item()*)? XP3.1 XQ3.1**

2番目の引数内で与えられている名前と同じ名前を持つ最初の引数内で指定されているコンポーネントの型を返します。コンポーネントのプロパティの一つを選択して更にナビゲートすることができます。

- このプロパティがコンポーネントの場合、このコンポーネントのプロパティの1つを選択して更に深く他のステップをナビゲートすることができます。このステップは更にスキーマをナビゲートするために繰り返すことができます。
- コンポーネントが動的な値の場合、動的な値が返され更にナビゲートすることはできません。

メモ Xquery 式ではスキーマは明示的にインポートされる必要があります。XPath 式では、スキーマは処理環境にインポートされる必要があります。例えば、XSLT には**xslt:import** 命令を使用してインポートします。

⊟ *サンプル*

- ```
  import schema "" at "C:\Test\ExpReport.xsd";
  altova:schema("element declaration", xs:QName("OrgChart"))("type definition")
  ("content type")("particles")[3]!.("term")("kind")
  ```
  は3番目の`particles` コンポーネントの `kind` プロパティを返します。 この`particles` コンポーネントは`OrgChart` の `Qname` を持つ要素宣言の子孫です。

- ```
  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")]
  ```
  は各 **emailType** コンポーネントの各 **facet** にそのファセットの型と値を含む列挙を返します。

コンポーネントとそのプロパティ

⊟ Assertion

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Assertion"
test	XPath プロパティレコード	

⊟ Attribute Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Declaration"
name	文字列	属性のローカル名

target namespace	文字列	属性の名前空間 URI
type definition	Simple Type または Complex Type	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	存在する場合、プロパティを持つ関数 (″class″: ″Value Constraint″、″variety″: ″fixed″ または″default″, ″value″: atomic value, ″lexical form″: string。″value″ プロパティは namespace-sensitive 型のために使用することはできないことに注意してください。	
inheritable	ブール値	

Attribute Group Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Group Definition"
name	文字列	属性グループのローカル名
target namespace	文字列	属性グループの名前空間 URI
attribute uses	(Attribute Use) のシーケンス	
attribute wildcard	任意の属性のワイルドカード	

Attribute Use

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Use"
required	ブール値	属性が必要な場合は true 、任意の場合は false
value constraint	Attribute Declaration を参照	
inheritable	ブール値	

Attribute Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″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	文字列 (″strict″|″lax″|″skip″)	

Complex Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
base type definition	Complex Type Definition	
final	文字列のシーケンス ("restriction"|"extension")	
context	空のシーケンス(not implemented)	
derivation method	文字列 ("restriction"|"extension")	
abstract	ブール値	
attribute uses	Attribute Use のシーケンス	
attribute wildcard	任意の属性のワイルドカード	
content type	プロパティを持つ関数: ("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	文字列のシーケンス ("restriction"|"extension")	
assertions	アサーションのシーケンス	

Element Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
type definition	Simple Type または Complex Type	
type table	プロパティを持つ関数 ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	プロパティを持つ関数 ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	Attribute Declaration を参照	
nillable	ブール値	
identity-constraint	Identity Constraint のシーケンス	

(C) 2015-2021 Altova GmbH

definitions		
substitution group affiliations	Element Declaration のシーケンス	
substitution group exclusions	文字列のシーケンス ("restriction"|"extension")	
disallowed substitutions	文字列のシーケンス ("restriction"|"extension"|"substitution")	
abstract	ブール値	

Element Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″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	文字列 (″strict″|″lax″|″skip″)	

Facet

プロパティ名	プロパティ型	プロパティ値
kind	文字列	ファセットの名前、例えば、″minLength″ または″enumeration″
value	ファセットによる	ファセットの値
fixed	ブール値	
typed-value	ファセットの列挙のみ、Array(xs:anyAtomicType	列挙値を含む配列。それぞれがアトミック値のシーケンスである場合があります。(メモ 列挙ファセットに関しては、実際の型にかかわらず″value″値プロパティは文字列のシーケンスです)

Identity Constraint

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Identity-Constraint Definition"
name	文字列	制約のローカル名
target namespace	文字列	制約の名前空間 URI
identity-constraint category	文字列 (″key″|″unique″|″keyRef″)	
selector	XPath プロパティレコード	
fields	XPath プロパティレコード のシーケンス	

referenced key	(keyRef のためのみ): Identity Constraint	対応するキー制約

Model Group

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group"
compositor	文字列 ("sequence"|"choice"|"all")	
particles	Particle のシーケンス	

Model Group Definition

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group Definition"
name	文字列	モデルグループのローカル名
target namespace	文字列	モデルグループの名前空間 URI
model group	Model Group	

Notation

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Notation Declaration"
name	文字列	表記のローカル名
target namespace	文字列	表記の名前空間 URI
system identifier	anyURI	
public identifier	文字列	

Particle

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Particle"
min occurs	整数	
max occurs	整数または文字列("unbounded")	
term	Element Declaration、Element Wildcard、または ModelGroup	

Simple Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Simple Type Definition"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
final	文字列のシーケンス	

	("restriction"|"extension"|"list"|"union")	
context	含まれるコンポーネント	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	(実装されていない)空のシーケンス	
variety	文字列 ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(リスト型のみのための) Simple Type	
member type definitions	(ユニオン型のための) Simple Type のシーケンス	

Type Alternative

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Type Alternative"
test	XPath プロパティレコード	
type definition	Simple Type または Complex Type	

XPath Property Record

プロパティ名	プロパティ型	プロパティ値
namespace bindings	プロパティを持つ関数のシーケンス("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	XPath 式の静的ベースURI
expression	文字列	文字列としてのXPath 式

型

altova:type(Node *as item?*) **as (function(xs:string) as item()*)? XP3.1 XQ3.1**
関数 **altova:type** はXMLドキュメントの要素または属性ノードを送信しノードの型情報をPSVI から返します。

メモ スキーマが参照可能になるためにXMLドキュメントはスキーマ宣言を持つ必要があります。

サンプル

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type
  ```
  はノードの型情報を含む関数を返します。

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type ("kind")
  ```

はノードの型のコンポーネント（単純型または複合型）を取りコンポーネントの`kind` property の値を返します。

コンポーネントとそのプロパティ

- Assertion

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Assertion"
test	XPath プロパティレコード	

- Attribute Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Declaration"
name	文字列	属性のローカル名
target namespace	文字列	属性の名前空間 URI
type definition	Simple Type または Complex Type	
scope	プロパティを持つ関数（"class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	存在する場合、プロパティを持つ関数（"class": "Value Constraint"、"variety": "fixed" または "default", "value": atomic value, "lexical form": string。"value" プロパティは namespace-sensitive 型のために使用することはできないことに注意してください。	
inheritable	ブール値	

- Attribute Group Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Group Definition"
name	文字列	属性グループのローカル名
target namespace	文字列	属性グループの名前空間 URI
attribute uses	(Attribute Use) のシーケンス	
attribute wildcard	任意の属性のワイルドカード	

- Attribute Use

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Use"
required	ブール値	属性が必要な場合は true 、任意の場合は false

 (C) 2015-2021 Altova GmbH

value constraint	Attribute Declaration を参照	
inheritable	ブール値	

Attribute Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″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	文字列 (″strict″|″lax″|″skip″)	

Complex Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
base type definition	Complex Type Definition	
final	文字列のシーケンス ("restriction"|"extension")	
context	空のシーケンス(not implemented)	
derivation method	文字列 (″restriction″|″extension″)	
abstract	ブール値	
attribute uses	Attribute Use のシーケンス	
attribute wildcard	任意の属性のワイルドカード	
content type	プロパティを持つ関数: (″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	文字列のシーケンス ("restriction"|"extension")	
assertions	アサーションのシーケンス	

Element Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
type definition	Simple Type または Complex Type	
type table	プロパティを持つ関数 (″class″:″Type Table″, "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	Attribute Declaration を参照	
nillable	ブール値	
identity-constraint definitions	Identity Constraint のシーケンス	
substitution group affiliations	Element Declaration のシーケンス	
substitution group exclusions	文字列のシーケンス ("restriction"|"extension")	
disallowed substitutions	文字列のシーケンス ("restriction"|"extension"|"substitution")	
abstract	ブール値	

- Element Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″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	文字列 (″strict″|″lax″|″skip″)	

- Facet

プロパティ名	プロパティ型	プロパティ値
kind	文字列	ファセットの名前、例えば、″minLength″ または″enumeration″
value	ファセットによる	ファセットの値
fixed	ブール値	

typed-value	ファセットの列挙のみ、Array(xs:anyAtomicType	列挙値を含む配列。それぞれがアトミック値のシーケンスである場合があります。(メモ列挙ファセットに関しては、実際の型にかかわらず "value" 値プロパティは文字列のシーケンスです)

Identity Constraint

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Identity-Constraint Definition"
name	文字列	制約のローカル名
target namespace	文字列	制約の名前空間 URI
identity-constraint category	文字列 ("key"|"unique"|"keyRef")	
selector	XPath プロパティレコード	
fields	XPath プロパティレコード のシーケンス	
referenced key	(keyRef のためのみ): Identity Constraint	対応するキー制約

Model Group

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group"
compositor	文字列 ("sequence"|"choice"|"all")	
particles	Particle のシーケンス	

Model Group Definition

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group Definition"
name	文字列	モデルグループのローカル名
target namespace	文字列	モデルグループの名前空間 URI
model group	Model Group	

Notation

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Notation Declaration"
name	文字列	表記のローカル名
target namespace	文字列	表記の名前空間 URI
system identifier	anyURI	
public identifier	文字列	

Particle

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Particle"
min occurs	整数	
max occurs	整数または文字列("unbounded")	
term	Element Declaration、Element Wildcard、または ModelGroup	

Simple Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Simple Type Definition"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
final	文字列のシーケンス ("restriction"|"extension"|"list"|"union")	
context	含まれるコンポーネント	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	(実装されていない)空のシーケンス	
variety	文字列 ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(リスト型のみのための) Simple Type	
member type definitions	(ユニオン型のための) Simple Type のシーケンス	

Type Alternative

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Type Alternative"
test	XPath プロパティレコード	
type definition	Simple Type または Complex Type	

XPath Property Record

プロパティ名	プロパティ型	プロパティ値
namespace bindings	プロパティを持つ関数のシーケンス("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	XPath 式の静的ベース URI
expression	文字列	文字列としての XPath 式

8.2.1.7 XPath/XQuery 関数: シーケンス

Altova のシーケンス拡張関数はXPath とXQuery 式で使用することができ、XML スキーマの異なる日付および時刻データ型で保存されているデータを処理するための追加機能を提供します。このセクションの関数は、Altova の**XPath 3.0** および**XQuery 3.0** エンジンで使用することができます。これらの関数は、XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることに注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

attributes [altova:]

altova:attributes(AttributeName *as xs:string*) を**attribute()*** とする XP3.1 XQ3.1
入力引数 AttributeName 内で与えられた名前と同じローカル名を持つすべての属性を返します。検索は大文字と小文字を区別し、attribute:: 軸に対して行われます。これは、コンテキストノードが親要素ノードである必要があることを意味します。

サンプル

- **altova:attributes**("MyAttribute") はMyAttribute()* を返します。

altova:attributes(AttributeName *as xs:string*, **SearchOptions** *as xs:string*) **asattribute()*** XP3.1 XQ3.1
入力引数 AttributeName 内で与えられた名前と同じローカル名を持つすべての属性を返します。検索は大文字と小文字を区別し、attribute:: 軸に対して行われます。コンテキストノードが親要素ノードである必要があります。第 2 引数はオプションのフラッグを含みます。使用することのできるフラッグは以下の通りです:
r = 正規表現検索を切り替えます; AttributeName は正規表現検索文字列である必要があります;
f = このオプションが指定されている場合、AttributeName は完全一致を提供します。それ以外の場合は、AttributeName は属性名に部分的に一致するとその属性を返します。例えば **f** が指定されていない場合、MyAtt は MyAttribute を返します。
i = 大文字と小文字を一致させる検索に切り替えます。
p = 検索に名前空間プレフィックスを含みます。AttributeName は、名前空間プレフィックスを含みます。例えば altova:MyAttribute。
フラッグは順序に関わりなく書き込むことができます。無効なフラッグはエラーを生成します。1 つまたは複数のフラッグを省略することができます。空の文字列は許可されていますが、引数を1 つしか持たない関数と同じ効果を持ちます(*前の署名*)、ですが、空のシーケンスは第 2 の引数としては許可されていません。

サンプル

- **altova:attributes**("MyAttribute", "rfip") はMyAttribute()* を返します。

- **altova:attributes**("MyAttribute", "pri") はMyAttribute()* を返します。
- **altova:attributes**("MyAtt", "rip") はMyAttribute()* を返します。
- **altova:attributes**("MyAttributes", "rfip") は不一致を返します。
- **altova:attributes**("MyAttribute", "") はMyAttribute()* を返します。
- **altova:attributes**("MyAttribute", "Rip") 認識されないフラッグエラーを返します。
- **altova:attributes**("MyAttribute",) 見つからない第 2 引数を返します。

elements [altova:]

altova:elements(ElementName *as xs:string*) を**element()*** とする **XP3.1** **XQ3.1**
入力引数 ElementName で与えられた名前と同じローカル名を持つすべての要素を返します。検索は大文字と小文字を区別して child:: 軸に対して実行されます。コンテキストノードは、検索される要素の親ノードである必要があります。

サンプル

- **altova:elements**("MyElement") はMyElement()* を返します。

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string*) **aselement()*** **XP3.1** **XQ3.1**
入力引数 ElementName 内で与えられた名前と同じローカル名を持つすべての属性を返します。検索は大文字と小文字を区別し、child:: 軸に対して行われます。コンテキストノードが親要素ノードである必要があります。第 2 引数はオプションのフラッグを含みます。使用することのできるフラッグは以下の通りです:
r = 正規表現検索を切り替えます; ElementName は正規表現検索文字列である必要があります;
f = このオプションが指定されている場合、ElementName は完全一致を提供します。それ以外の場合は、ElementName は属性名に部分的に一致するとその属性を返します。例えば **f** が指定されていない場合、MyElem はMyElement を返します。
i = 大文字と小文字を一致させる検索に切り替えます。
p = 検索に名前空間プレフィックスを含みます。ElementName は、名前空間プレフィックスを含みます。例えば altova:MyElement。
フラッグは順序に関わりなく書き込むことができます。無効なフラッグはエラーを生成します。1 つまたは複数のフラッグを省略することができます。空の文字列は許可されていますが、引数を1 つしか持たない関数と同じ効果を持ちます(*前の署名*)、ですが、空のシーケンスは第 2 の引数としては許可されていません。

サンプル

- **altova:elements**("MyElement", "rip") はMyElement()* を返します。
- **altova:elements**("MyElement", "pri") はMyElement()* を返します。
- **altova:elements**("MyElement", "") はMyElement()* を返します。
- **altova:attributes**("MyElem", "rip") はMyElement()* を返します。
- **altova:attributes**("MyElements", "rfip") 不一致を課wします。
- **altova:elements**("MyElement", "Rip") 認識されないフラッグエラーを返します。
- **altova:elements**("MyElement",) 見つからない第 2 引数を返します。

find-first [altova:]

altova:find-first((Sequence *as item()**), **(Condition(Sequence-Item** *as xs:boolean*)) を **item()?** とする **XP3.1** **XQ3.1**
この関数は2 つの引数を必要とします。最初の引数は1 つ、または1 つ以上のデータ型のアイテムのシーケンスです。第 2 の引数 Condition は、(1 のアリティを持つ) 1 つの引数 を必要とし、boolean を返す XPath 関数に対する参照です。Condition で参照された関数の代わりに、**Sequence** の各アイテムが提出されます。(*注意:* この関数は1 つの引数のみを必要とします。) **Condition** 内の関数に**true()**と評価させる最初の **Sequence** アイテムは、**altova:find-first**、反復の終了の結果として返されます。

サンプル

- **altova:find-first**(5 to 10, function($a) {$a mod 2 = 0}) は xs:integer 6 を返します。**Condition** 引数は、**$a** という名のインライン関数を宣言し、定義します。XPath 3.0 インライン関数 **function()** を参照します。**altova: Sequence** 引数内の各アイテムでは、**find-first** がパスされ、代わりに、**$a** を入力値とします。入力値は関数定義($a mod 2 = 0)内の条件に対してテストされます。この条件を満たす最初の入力値が **altova:find-first**（この場合は 6）の結果として返されます。

- **altova:find-first**((1 to 10), (function($a) {$a+3=7})) は xs:integer 4 を返します。

更なるサンプル
ファイル **C:\Temp\Customers.xml** が存在する場合:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) は xs:string C:\Temp\Customers.xml を返します。

ファイル **C:\Temp\Customers.xml** が存在せず、**http://www.altova.com/index.html** が存在する場合:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) は xs:string http://www.altova.com/index.html を返します。

ファイル **C:\Temp\Customers.xml** が存在せず、**http://www.altova.com/index.html** も存在しない場合:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) 結果無しを返します。

上のサンプルについての注意点

- XPath 3.0 関数 doc-available は URI として使用され、ドキュメントノードが提出された URI で検出される場合 true を返する単一の引数を必要とします。（ですから、提出された URI でのドキュメントは XML ドキュメントである必要があります。）
- doc-available 関数は、altova:find-first の第 2 引数である、**Condition** で使用することができます。これは、1 つの引数（アリティ=1）のみを必要とするからであり、item() を入力（ URI として使用される文字列）として、boolean の値を返すからです。
- doc-available 関数は、参照されているだけで、呼び出されていない点に注意してください。アタッチされている #1 サフィックスは関数が 1 つのアリティであることを表示するためです。doc-available#1 の意味は以下のとおりです: アリティ=1 を持つ doc-availabe() 関数を使用し、最初のシーケンスの各アイテムの代わりに、単一引数として パスします。この結果、 2 つの文字列の各自つは、文字列をURIとして使用し、URIにドキュメントノードが存在するかテストする **doc-available()** にパスされます。1 つが各当する場合、**doc-available()** 関数は true() を評価し、シーケンス内のその文字列のインデックス ポジションは、 **altova:find-first** 関数の結果として返されます。doc-available() 関数に関しての注意点: 相対パスは、デフォルトで関数がロードされる XML ドキュメントの現在のベース URI に対して相対的に解決されます。

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*)) を **item()*** とする **XP3.1** **XQ3.1**
この関数は 3 つの引数を必要とします:

- 最初の2 つの引数 Seq-01 とSeq-02, は、1 つまたは1 つ以上のデータ型のアイテムです。
- 第 3 の引数 Condition は、(2 のアリティを持つ) 2 つの引数を必要とし、boolean を返す XPath 関数に対しての参照です。

Seq-01 とSeq-02 のアイテムは指定された組み合わせ(各シーケンスからの1 つずつのアイテムで構成されるペア)で、Condition 内の関数の引数として パスされました。組み合わせは以下のように指定されています。

```
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)
```

Condition 関数にtrue() と評価するように指示する最初のペアはaltova:find-first-combination の結果として返されます。以下の点に注意してください: (i) Condition 関数が提出された引数ペア内で繰り返され、true()を評価しない場合、altova:find-first-combination は、結果を返しません; (ii) altova:find-first-combination の結果が常に(データ型の)アイテムのペアである場合またはアイテムでない場合。

サンプル

- altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) はxs:integers (11, 21) のシーケンスを返します。
- altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) はxs:integers (11, 22) のシーケンを返します。
- altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 34}) は xs:integers (11, 23) のシーケンを返します。

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*)) をitem()* とする XP3.1 XQ3.1
この関数は3 つの引数を必要とします:

- 最初の2 つの引数 Seq-01 とSeq-02, は、1 つまたは1 つ以上のデータ型のアイテムです。
- 第 3 の引数 Condition は、(2 のアリティを持つ) 2 つの引数を必要とし、boolean を返す XPath 関数に対しての参照です。

Seq-01 とSeq-02 のアイテムは指定された組み合わせで、Condition 内の関数の引数として パスされました。組み合わせは以下のように指定されています。

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

Condition 関数にtrue() と評価するように指示する最初のペアはaltova:find-first-pair の結果として返されます。以下の点に注意してください: (i) Condition 関数が提出された引数ペア内で繰り返され、true()を評価しない場合、altova:find-first-pair は、結果を返しません; (ii) altova:find-first-combination の結果が常に(データ型の) アイテムのペアである場合またはアイテムでない場合。

サンプル

- altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) は xs:integers (11, 21) のシーケンスを返します。
- altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) は*結果無し*を返します。

上の2 つのサンプルに表示される通り、ペアの順序は以下の通りです: (11, 21) (12, 22) (13, 23)...(20,

30) 。(33 を返す指示されたペアがないため) この理由で第 2 のサンプルは *結果無し*を返します。

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*)**)** を**xs:integer** とする **XP3.1** **XQ3.1**
この関数は3 つの引数を必要とします:

- 最初の2 つの引数 **Seq-01** と**Seq-02**, は、1 つまたは1 つ以上のデータ型のアイテムです。
- 第 3 の引数 **Condition** は、(2 のアリティを持つ) 2 つの引数を必要とし、boolean を返す XPath 関数に対しての参照です。

Seq-01 と**Seq-02** のアイテムは指定された組み合わせで、**Condition** 内の関数の引数としてパスされました。組み合わせは以下のように指定されています。

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

Condition 関数に**true()**を評価させる、最初に指示されたペアのインデックス ポジションは、は**altova:find-first-pair-pos** の結果として返されます。関数が提出された引数ペア内で繰り返され、**true()**を一度も評価しない場合、**altova:find-first-pair-pos** *結果無し*が返します。

サンプル

- **altova:find-first-pair-pos**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) は 1 を返します
- **altova:find-first-pair-pos**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) は*結果無し*を返します。

上の2 つのサンプルに表示される通り、ペアの順序は以下の通りです: (11, 21) (12, 22) (13, 23)...(20, 30)。最初のサンプルでは、最初のペアは**Condition** 関数に**true()**を評価させ、シーケンス内のインデックス ポジションには**1** が返されます。第 2 のサンプルでは、33 を返すペアがないため、*結果無し*が返します。

find-first-pos [altova:]

altova:find-first-pos((Sequence *as item()**)**, (Condition(Sequence-Item** *as xs:boolean*)**)** を**xs:integer** とする **XP3.1** **XQ3.1**
この関数は2 つの引数を必要とします。最初の引数は1 つ、または1 つ以上のデータ型のアイテムのシーケンスです。第 2 の引数 Condition は、(1 のアリティを持つ) 1 つの引数 を必要とし、boolean を返す XPath 関数に対する参照です。Condition で参照された関数の代わりに、**Sequence** の各アイテムが提出されます。(*注意:* この関数は1 つの引数のみを必要とします。) **Condition** 内の関数に**true()**と評価させる最初の **Sequence** アイテムは、**altova:find-first-pos** の結果として返された**Sequence** 内インデックス ポジションを持ちます。

サンプル

- **altova:find-first-pos**(5 to 10, function($a) {$a mod 2 = 0}) は**xs:integer** 2 を返します。
  Condition 引数は、**$a** という名のインライン関数を宣言し、定義します。XPath 3.0 インライン関数 **function()**を参照します。 **Sequence** 引数内の各アイテムでは、**find-first-pos** がパスされ、代わりに、**$a** を入力値とします。入力値は関数定義($a mod 2 = 0)内の条件に対してテストされます。この条件を満たす最初の入力値が **altova:find-first-pos**((シーケンス内で条件を満たす最初の値である6 がシーケンスのインデックス位置 2 にあるためこの場合は2、) の結果として返されます。

- **altova:find-first-pos**((2 to 10), (function($a) {$a+3=7})) は xs:integer 3 を返します。

更なるサンプル
ファイル **C:\Temp\Customers.xml** が存在する場合:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) は 1 を返します。

ファイル **C:\Temp\Customers.xml** が存在せず、**http://www.altova.com/index.html** が存在する場合:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) は 2 を返します。

ファイル **C:\Temp\Customers.xml** が存在せず、**http://www.altova.com/index.html** も存在しない場合:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) 結果無しを返します。

上のサンプルについての注意点

- XPath 3.0 関数 doc-available は URI として使用され、ドキュメントノードが提出された URI で検出される場合 true を返する単一の引数を必要とします。(ですから、提出された URI でのドキュメントは XML ドキュメントである必要があります。)
- doc-available 関数は、altova:find-first-pos の第 2 引数である、**Condition** で使用することができます。これは、1 つの引数 (アリティ=1) のみを必要とするからであり、item() を入力(URI として使用される文字列) として、boolean の値を返すからです。
- doc-available 関数は、参照されているだけで、呼び出されていない点に注意してください。アタッチされている #1 サフィックスは関数が 1 つのアリティであることを表示するためです。doc-available#1 の意味は以下のとおりです: アリティ=1 を持つ doc-availabe() 関数を使用し、最初のシーケンスの各アイテムの代わりに、単一引数として パスします。この結果、 2 つの文字列の各自つは、文字列をURIとして使用し、URIにドキュメントノードが存在するかテストする **doc-available()** にパスされます。1 つが各当する場合、**doc-available()** 関数は true() を評価し、シーケンス内のその文字列のインデックス ポジションは、 **altova:find-first-pos** 関数の結果として返されます。doc-available() 関数に関しての注意点: 相対パスは、デフォルトで関数がロードされる XML ドキュメントの現在のベース URI に対して相対的に解決されます。

for-each-attribute-pair [altova:]

altova:for-each-attribute-pair(Seq1 *as element()?*, **Seq2** *as element()?*, **Function** *as function()*) **asitem()*** **XP3.1** **XQ3.1**

ペアの一つの属性が最初の要素を取得し、もう一つの属性が2番目の要素から取得された箇所で、最初の2つの引数は、2つの要素、および、属性ペアを作成するために使用された属性を識別します。要素ペアは同じ名前を持つことをベースに選択され、ペアは名前別に アルファベット順にセットに並べ替えられます。属性のために他の要素上に対応する属性が存在しない場合、ペアのジョイントは解除されます。これは、一つのメンバーにより飲み構成されることを意味します。関数アイテム(3番目の引数 Function) はアイテムのシーケンスである出力であるペアのシーケンス内のペア(ジョイント、ジョイントの解除) に個別に適用されます。

サンプル

- **altova:for-each-attribute-pair**(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b}) は以下 を返します。...

```
(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" />
```

メモ: 結果 (2, 6) は次のアクションにより取得されます: (1+1, ()+2, 3+3, 4+()) 。演算子の一つが空のシーケンスの場合、アイテム 2 と 4 の場合同様、結果は空のシーケンスになります。

- **altova:for-each-attribute-pair**(/Example/Test-A, /Example/Test-B, concat#2) は以下を返します。

```
(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}) を item()* とする** **XP3.1** **XQ3.1**

第 3 の引数として与えられた、インライン関数を返します。FirstSequence 内の各 $i を SecondSequence内の各 $j と結合します。出力はこれらのアイテムのシーケンスです。

サンプル

- **altova:for-each-combination**(('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j}) は ('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3') を返します。

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**

最初の2 つの引数内の2 つのシーケンスのアイテムは最初のシーケンスの各アイテムが2 番目のシーケンスになるように結合されるように結合されます。3 番目の引数として与えられる関数は結果のシーケンス内の組み合わせに適用され、アイテムのシーケンスである出力内で出力されます（*サンプル参照*）。

サンプル

- **altova:for-each-combination**(('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j}) は ('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3') を返します。

▼ substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()*) を **item()*** とする **XP3.1** **XQ3.1**
FirstSequence が空の場合、SecondSequence を返します。FirstSequence が空でない場合、FirstSequence を返します。

サンプル

- **altova:substitute-empty**((1,2,3), (4,5,6)) は (1,2,3) を返します。
- **altova:substitute-empty**((), (4,5,6)) は (4,5,6) を返します。

8.2.1.8 XPath/XQuery 関数: 文字列

Altova の文字列拡張関数は XPath と XQuery 式で使用することができ、XML スキーマの異なる日付および時刻データ型で保存されているデータを処理するための追加機能を提供します。このセクションの関数は、Altova の **XPath 3.0** および **XQuery 3.0** エンジンで使用することができます。これらの関数は、XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は **Altova 拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることに注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内の XPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内の XQuery 式で使用):	**XQ1** **XQ3.1**

▼ camel-case [altova:]

altova:camel-case(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
入力文字列を **InputString** をキャメルケースで返します。文字列は(空白スペースのショートカットである) 正規表現 **'\s'** を使用して分析されます。空白文字または連続する空白文字のシーケンスの後の最初の非空白スペース文字は大文字です。出力文字列の最初の文字は大文字です。

サンプル

- **altova:camel-case**("max") Max を返します。
- **altova:camel-case**("max max") Max Max を返します。
- **altova:camel-case**("file01.xml") File01.xml を返します。
- **altova:camel-case**("file01.xml file02.xml") File01.xml File02.xml を返します。
- **altova:camel-case**("file01.xml file02.xml") File01.xml File02.xml を返します。
- **altova:camel-case**("file01.xml -file02.xml") File01.xml -file02.xml を返します。

altova:camel-case(InputString *as xs:string*, **SplitChars** *as xs:string*, **IsRegex** *as xs:boolean*) を **xs:string** とする **XP3.1** **XQ3.1**
SplitChars を使用して、次の大文字をトリガーする文字を決定し、入力文字列を **InputString** キャメルケースに変換しま

す。 **SplitChars** は、**IsRegex = true()** の場合、または、**IsRegex = false()** の場合、プレーン文字は正規表現として使用されます。出力文字列の最初の文字は大文字です。

サンプル

- **altova:camel-case**("setname getname", "set|get", true()) setName getName を返します。
- **altova:camel-case**("altova\documents\testcases", "\", false()) Altova\Documents\Testcases を返します。

char [altova:]

altova:char(Position *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**

xs:string.Position に対してのコンテキスト アイテムの値を変換することにより得られた文字列内の Position 引数により指定されたポジションにある文字を含む文字列を返します。引数により提出されたインデックスに文字が存在しない場合、結果文字列は空です。

サンプル

コンテキスト アイテムが **1234ABCD** の場合:

- **altova:char**(2) は 2 を返します。
- **altova:char**(5) は A を返します。
- **altova:char**(9) は空の文字列を返します。
- **altova:char**(-2) は空の文字列を返します。

altova:char(InputString *as xs:string*, **Position** *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**

引数として提出された文字列内の Position 引数により指定されたポジションでの文字を含む文字列を返します。Position 引数により提出されたインデックスに文字が存在しない場合、結果文字列は空です。

サンプル

- **altova:char**("2014-01-15", 5) は - を返します。
- **altova:char**("USA", 1) は U を返します。
- **altova:char**("USA", 10) は空の文字列を返します。
- **altova:char**("USA", -2) は空の文字列を返します。

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**

HashAlgo 引数により指定されているハッシュアルゴリズムを使用して InputString からハッシュ文字列を生成します。次のハッシュアルゴリズムは（大文字、または、小文字で指定されている可能性があります: **MD5**, **SHA-224**, **SHA-256**, **SHA-384**, **SHA-512**。（最初の署名を参照してください） 2番目の引数が指定されていない場合、**SHA-256** ハッシュアルゴリズムが使用されます。

サンプル

- **altova:create-hash-from-string**('abc') は **SHA-256** ハッシュアルゴリズムを使用して生成されたハッシュ文字列 を返します。
- **altova:create-hash-from-string**('abc', 'md5') は **MD5** ハッシュアルゴリズムを使用して生成されたハッシュ文字列 を返します。
- **altova:create-hash-from-string**('abc', 'MD5') は **MD5** ハッシュアルゴリズムを使用して生成された

ハッシュ文字列 を返します。

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
xs:string に対するコンテキストアイテムの値を変換することにより得られた最初の X-Number 文字を含む文字列を返します。

サンプル

コンテキストアイテムが **1234ABCD** の場合:

- **altova:first-chars**(2) は 12 を返します。
- **altova:first-chars**(5) は 1234A を返します。
- **altova:first-chars**(9) は 1234ABCD を返します。

altova:first-chars(InputString *as xs:string*, **X-Number** *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
InputString 引数として提出された文字列の最初の文字を含む文字列を返します。

サンプル

- **altova:first-chars**("2014-01-15", 5) は 2014- を返します。
- **altova:first-chars**("USA", 1) は U を返します。

format-string [altova:]

altova:format-string(InputString *as xs:string*, **FormatSequence** *as item()**) **as xs:string** **XP3.1** **XQ3.1**
入力文字列（最初の引数）には、配置パラメーター（%1, %2, etc）が含まれています。各パラメーターは、（2番目の引数として提出されている）フォーマットシーケンス内の対応するポジションでロケートされる文字列アイテムと置き換えられます。フォーマットシーケンス内の最初のアイテムは、配置パラメーター %1 を置換、二番目のアイテムは %2 を置き換えます。関数は、書式設定された代替を持つ文字列を返します。配置パラメーターに文字列が存在しない場合は、配置パラメーターが返されます。これは、配置パラメーターのインデックスが書式シーケンス内のアイテムの数より大きい場合発生します。

サンプル

- **altova:format-string**('Hello %1, %2, %3', ('Jane','John','Joe')) は "Hello Jane, John, Joe" を返します。
- **altova:format-string**('Hello %1, %2, %3', ('Jane','John','Joe', 'Tom')) は "Hello Jane, John, Joe" を返します。
- **altova:format-string**('Hello %1, %2, %4', ('Jane','John','Joe', 'Tom')) は "Hello Jane, John, Tom" を返します。
- **altova:format-string**('Hello %1, %2, %4', ('Jane','John','Joe')) は "Hello Jane, John, %4" を返します。

last-chars [altova:]

altova:last-chars(X-Number *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
xs:string に対してのコンテキストアイテムの値の変換より取得された文字列の最後の X-Number 文字を含んでいる文字列を返します。

サンプル

コンテキストアイテムが **1234ABCD** の場合:

- **altova:last-chars**(2) は CD を返します。
- **altova:last-chars**(5) は 4ABCD を返します。
- **altova:last-chars**(9) は 1234ABCD を返します。

altova:last-chars(InputString *as xs:string*, **X-Number** *as xs:integer*) **as xs:string** とする **XP3.1** **XQ3.1**
引数として提出された文字列の最後の X-Number 文字を含んでいる文字列を返します。

サンプル

- **altova:last-chars**("2014-01-15", 5) は 01-15 を返します。
- **altova:last-chars**("USA", 10) は USA を返します。

pad-string-left [altova:]

altova:pad-string-left(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
PadCharacter 引数は1文字です。文字列の左側にパッドされ、この数が、StringLength 引数の整数の値と等しくなるように StringToPad の文字の数を増やします。StringLength 引数は任意の整数の値（正数または負数）を持つことができますが、パディングは、StringLength の値が StringToPad 内の文字数より多い場合の場合のみ発生します。もし、StringToPad が StringLength の値より多くの文字数も持つ場合、StringToPad は変更されません。

サンプル

- **altova:pad-string-left**('AP', 1, 'Z') は 'AP' を返します。
- **altova:pad-string-left**('AP', 2, 'Z') は 'AP' を返します。
- **altova:pad-string-left**('AP', 3, 'Z') は 'ZAP' を返します。
- **altova:pad-string-left**('AP', 4, 'Z') は 'ZZAP' を返します。
- **altova:pad-string-left**('AP', -3, 'Z') は 'AP' を返します。
- **altova:pad-string-left**('AP', 3, 'YZ') は[パッド文字が長すぎます] エラーを返します。

pad-string-right [altova:]

altova:pad-string-right(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
PadCharacter 引数は1文字です。文字列の右側にパッドされ、この数が、StringLength 引数の整数の値と等しくなるように StringToPad の文字の数を増やします。StringLength 引数は任意の整数の値（正数または負数）を持つことができますが、パディングは、StringLength の値が StringToPad 内の文字数より多い場合の場合のみ発生します。もし、StringToPad が StringLength の値より多くの文字数も持つ場合、StringToPad は変更されません。

サンプル

- **altova:pad-string-right**('AP', 1, 'Z') を 'AP' を返します。
- **altova:pad-string-right**('AP', 2, 'Z') を 'AP' を返します。
- **altova:pad-string-right**('AP', 3, 'Z') を 'APZ' を返します。
- **altova:pad-string-right**('AP', 4, 'Z') を 'APZZ' を返します。
- **altova:pad-string-right**('AP', -3, 'Z') を 'AP' を返します。
- **altova:pad-string-right**('AP', 3, 'YZ') は[パッド文字が長すぎます] エラーを返します。

repeat-string [altova:]

altova:repeat-string(InputString *as xs:string*, **Repeats** *as xs:integer*) を **xs:string** とする

XP2 XQ1 XP3.1 XQ3.1
最初の InputString 引数により構成される文字列、Repeats 回繰り返してを生成します。

サンプル

- **altova:repeat-string**("Altova #", 3) は "Altova #Altova #Altova #" を返します。

substring-after-last [altova:]

altova:substring-after-last(MainString *as xs:string*, **CheckString** *as xs:string*) を **xs:string** とする XP3.1 XQ3.1
CheckString が MainString 内で検出された場合、MainString 内の CheckString が発生した後のサブ文字列が返されます。MainString 内で CheckString が検出されない場合、空の文字列が返されます。CheckString が空の文字列の場合、MainString 全体が返されます。一度以上発生する場合、CheckString の最後の発生の後の サブ文字列が返されます。

サンプル

- **altova:substring-after-last**('ABCDEFGH', 'B') は 'CDEFGH' を返します。
- **altova:substring-after-last**('ABCDEFGH', 'BC') は 'DEFGH' を返します。
- **altova:substring-after-last**('ABCDEFGH', 'BD') は '' を返します。
- **altova:substring-after-last**('ABCDEFGH', 'Z') は '' を返します。
- **altova:substring-after-last**('ABCDEFGH', '') は 'ABCDEFGH' を返します。
- **altova:substring-after-last**('ABCD-ABCD', 'B') は 'CD' を返します。
- **altova:substring-after-last**('ABCD-ABCD-ABCD', 'BCD') は '' を返します。

substring-before-last [altova:]

altova:substring-before-last(MainString *as xs:string*, **CheckString** *as xs:string*) を **xs:string** とする XP3.1 XQ3.1
CheckString が MainString 内で検出された場合、MainString 内の CheckString が発生する前のサブ文字列が返されます。MainString 内で CheckString が一度以上発生する場合、CheckString の最後の発生の前の サブ文字列が返されます。

サンプル

- **altova:substring-before-last**('ABCDEFGH', 'B') は 'A' を返します。
- **altova:substring-before-last**('ABCDEFGH', 'BC') は 'A' を返します。
- **altova:substring-before-last**('ABCDEFGH', 'BD') は '' を返します。
- **altova:substring-before-last**('ABCDEFGH', 'Z') は '' を返します。
- **altova:substring-before-last**('ABCDEFGH', '') は '' を返します。
- **altova:substring-before-last**('ABCD-ABCD', 'B') は 'ABCD-A' を返します。
- **altova:substring-before-last**('ABCD-ABCD-ABCD', 'ABCD') は 'ABCD-ABCD-' を返します。

substring-pos [altova:]

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*) を **xs:integer** とする XP3.1 XQ3.1
StringToCheck 内での StringToFind の最初の発生の文字位置を整数として返します。StringToCheck の最初の文字は、位置 1 にあります。StringToFind が StringToCheck 内で発生しない場合、整数 0 が返されます。第 2 またはその後の StringToCheck, 発生を確認するには、この関数の次の署名を確認してください。

サンプル

- **altova:substring-pos**('Altova', 'to') は 3 を返します。

- **altova:substring-pos**('Altova', 'tov') は 3 を返します。
- **altova:substring-pos**('Altova', 'tv') は 0 を返します。
- **altova:substring-pos**('AltovaAltova', 'to') は 3 を返します。

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*, **Integer** *as xs:integer*) を **xs:integer** とする **XP3.1** **XQ3.1**
StringToCheck 内での StringToFind の最初の発生の文字位置を返します。Integer 引数により与えられた、文字位置から StringToFind の検索が開始されます。この位置の前の文字サブ文字列は検索されません。しかし、返された整数は、全体文字列 StringToCheck の検索された文字列の位置です。この署名は、StringToCheck 内で複数回発生する、第 2 または後の発生を検索する際に役に立ちます。 StringToFind が StringToCheck 内で発生しない場合、整数 0 が返されます。

サンプル

- **altova:substring-pos**('Altova', 'to', 1) は 3 を返します。
- **altova:substring-pos**('Altova', 'to', 3) は 3 を返します。
- **altova:substring-pos**('Altova', 'to', 4) は 0 を返します。
- **altova:substring-pos**('Altova-Altova', 'to', 0) は 3 を返します。
- **altova:substring-pos**('Altova-Altova', 'to', 4) は 10 を返します。

trim-string [altova:]

altova:trim-string(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
この関数は xs:string 引数を必要とし、先頭または後続の空白を削除し、トリミングされた xs:string を返します。

サンプル

- **altova:trim-string**(" Hello World ")) は "Hello World" を返します。
- **altova:trim-string**("Hello World ")) は "Hello World" を返します。
- **altova:trim-string**(" Hello World")) は "Hello World" を返します。
- **altova:trim-string**("Hello World")) は "Hello World" を返します。
- **altova:trim-string**("Hello World")) は "Hello World" を返します。

trim-string-left [altova:]

altova:trim-string-left(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
この関数は xs:string 引数を必要とし、先頭または後続の空白を削除し、トリミングされた xs:string を返します。

サンプル

- **altova:trim-string-left**(" Hello World ")) は "Hello World " を返します。
- **altova:trim-string-left**("Hello World ")) は "Hello World " を返します。
- **altova:trim-string-left**(" Hello World")) は "Hello World" を返します。
- **altova:trim-string-left**("Hello World")) は "Hello World" を返します。
- **altova:trim-string-left**("Hello World")) は "Hello World" を返します。

trim-string-right [altova:]

altova:trim-string-right(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
この関数は xs:string 引数を必要とし、先頭または後続の空白を削除し、トリミングされた xs:string を返します。

サンプル

- **altova:trim-string-right**(" Hello World ")) は" Hello World" を返します。
- **altova:trim-string-right**("Hello World ")) は"Hello World" を返します。
- **altova:trim-string-right**(" Hello World")) は" Hello World" を返します。
- **altova:trim-string-right**("Hello World")) は"Hello World" を返します。
- **altova:trim-string-right**("Hello World")) は"Hello World" を返します。

8.2.1.9 XPath/XQuery 関数: その他

XPath/XQuery 拡張関数の以下の一般的な目的は現在のRaptorXML Server バージョンでサポートされており(i) XSLT コンテキスト内のXPath 式 または(ii) Xquery ドキュメント内のXQuery 式内で使用することができます。。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova **拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内のXPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内のXQuery 式で使用):	**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**
指定されたエンコードを使用して送信されたbase64Binary 入力を文字列にデコードします。エンコードが指定されていない場合UTF-8 エンコードが使用されます。以下のエンコードがサポートされます: US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4

サンプル

- **altova:decode-string**($XML1/MailData/Meta/b64B) はbase64Binary 入力をUTF-8 エンコード済み文字列として返します。
- **altova:decode-string**($XML1/MailData/Meta/b64B, "UTF-8") はbase64Binary 入力をUTF-8-エンコード済み文字列として返します。
- **altova:decode-string**($XML1/MailData/Meta/b64B, "ISO-8859-1") はbase64Binary 入力をISO-8859-1 エンコード済み文字列として返します。

encode-string [altova:]

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
与えらえている場合指定されているエンコードを使用して送信された文字列をエンコードします。エンコードが与えられていない場合 UTF-8 エンコードが使用されます。エンコードされた文字列はbase64Binary 文字に変換され、エンコードされたbase64Binary 値が返されます。最初にUTF-8 エンコードがサポートされ、サポートは以下のエンコードに拡張されます: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

サンプル

- **altova:encode-string**("Altova") はUTF-8 エンコード済み文字列 "Altova" の等価である base64Binary を返します。
- **altova:encode-string**("Altova", "UTF-8") はUTF-8 エンコード済み文字列 "Altova" の等価であるbase64Binary のを返します。

get-temp-folder [altova:]

altova:get-temp-folder() を xs:string とする XP2 XQ1 XP3.1 XQ3.1
この関数は引数を必要としません。この関数は現在のユーザーの一時的なフォルダーへのパスを返します。

サンプル

- **altova:get-temp-folder**() は、マシーン上で、xs:string として C:\Users\<UserName>\AppData\Local\Temp\ と類似したパスを返します。

generate-guid [altova:]

altova:generate-guid() asxs:string XP2 XQ1 XP3.1 XQ3.1
ユニークな文字列 GUID 文字列を生成します。

サンプル

- **altova:generate-guid**() は(例えば) 85F971DA-17F3-4E4E-994E-99137873ACCD を返します。

high-res-timer [altova:]

altova:high-res-timer() asxs:double XP3.1 XQ3.1
秒数でシステムハイれぞリューションタイマーの値を返します。ハイれぞリューションタイマーがシステム内で存在すると、必要とされる場合、高度に正確な時間の計算を有効化します(例えば、アニメーションと正確なコード実行の時間の決定などが例として挙げられます)。この関数は、システムのタイマーに精度を与えます。

サンプル

- **altova:high-res-timer**() は '1.16766146154566E6' などを返します。

parse-html [altova:]

altova:parse-html(HTMLText *as xs:string*) **asnode()** XP3.1 XQ3.1
HTMLText 引数は、HTML ドキュメントのテキストを含む文字列です。関数は、文字列からHTML ツリーを作成します。提供された文字列は、HTML 要素を含む、または、含まない場合があります。いずれの場合でも、ツリーのルート要素は、**HTML** と名付けられます。提出された文字列内のHTML コードが有効な HTMLであることを確認することが奨励されます。.

サンプル

- **altova:parse-html**("<html><head/><body><h1>Header</h1></body></html>") は提供された文字列からHTML ツリーを作成します。

▼ sleep[altova:]

altova:sleep(Millisecs *as xs:integer*) **asempty-sequence()** **XP2** **XQ1** **XP3.1** **XQ3.1**
Millisecs により与えられるミリ秒で示される期間のための実行を延期します。

サンプル

- **altova:sleep**(1000) は1000 ミリ秒のための実行を延期します。

[[トップ] 396]

8.2.1.10 チャート関数

以下に示されるチャート関数により、チャートをイメージとして作成、生成、保存することができます。これらの機能は、お使いの Altova 製品で、以下に記される方法によりサポートされます。しかしながら、将来使用されるバージョンの製品において、以下にある関数のサポートが打ち切られたり、個々の振る舞いが変化する可能性があることに留意してください。将来のリリースにおける Altova 拡張関数のサポートについては、そのバージョンのドキュメンテーションを参照ください。

(XSLT 関数ではない) チャートに関する XPath 関数は、2 つのグループに分けられています::

- チャートの生成と保存を行う関数 398
- チャートの作成を行う関数 399

メモ: チャート関数は、**Enterprise** ならびにサーバーエディションの Altova 製品でしかサポートされていないことに注意してください。

メモ: サーバーエディションのチャートでサポートされるイメージのフォーマットは jpg、png、および bmp です。無損失圧縮のため、推奨されるオプションは png です。Enterprise エディションでサポートされるフォーマットは jpg. png、bmp、および gif です。

チャートの生成と保存を行う関数

これらの関数は、(チャート作成関数により得られた) チャートオブジェクトを受け取り、イメージの生成を行うか、イメージをファイルへ保存します。

altova:generate-chart-image ($chart, $width, $height, $encoding) as atomic

以下の説明を参照ください

- $chart は altova:create-chart 関数により得られたチャート拡張アイテムです。
- $width ならびに $height により大きさを指定する必要があります。
- $encoding は x-binarytobase64 または x-binarytobase16 であることができます。

関数からは、指定されたエンコーディングならびにイメージフォーマットにてチャートのイメージが返されます。

altova:generate-chart-image ($chart, $width, $height, $encoding, $imagetype) as atomic

以下の説明を参照ください

- $chart は altova:create-chart 関数により得られたチャート拡張アイテムです。

- $width ならびに $height により大きさを指定する必要があります。
- $encoding は x-binarytobase64 または x-binarytobase16 であることができます。
- $imagetype は次のフォーマットから選択することができます: png、gif、bmp、jpg、jpeg。gif はサーバー製品でサポートされていないことご注意してください。*ページの上のメモを参照してください。*

関数からは、指定されたエンコーディングならびにイメージフォーマットにてチャートのイメージが返されます。

altova:save-chart-image ($chart, $filename, $width, $height) as empty() *(Windows のみ)*

以下の説明を参照ください

- $chart は altova:create-chart 関数により得られたチャート拡張アイテムです。
- $filename は保存されるチャートイメージのファイルパスならびにファイル名です。
- $width ならびに $height により大きさを指定する必要があります。

関数により $filename で指定されたファイルにチャートイメージが保存されます。

altova:save-chart-image ($chart, $filename, $width, $height, $imagetype) as empty() *(Windows のみ)*

以下の説明を参照ください

- $chart は altova:create-chart 関数により得られたチャート拡張アイテムです。
- $filename は保存されるチャートイメージのファイルパスならびにファイル名です。
- $width ならびに $height により大きさを指定する必要があります。
- $imagetype は次のフォーマットから選択することができます: png、gif、bmp、jpg、jpeg。gif はサーバー製品でサポートされていないことご注意してください。*ページの上のメモを参照してください。*

指定されたイメージフォーマットで $filename により指定されたファイルにチャートイメージが保存されます。

チャートの作成を行う関数

以下の関数によりチャートの作成を行うことができます

altova:create-chart($chart-config, $chart-data-series*) as chart extension item

以下の説明を参照ください

- $chart-config は altova:create-chart-config 関数または altova:create-chart-config-from-xml 関数により取得された chart-config 拡張アイテムとなります。
- $chart-data-series は altova:create-chart-data-series 関数または altova:create-chart-data-series-from-rows 関数により取得された chart-data-series 拡張アイテムとなります。

関数により、引数により与えられたデータから作成されたチャート拡張アイテムが返されます。

altova:chart($chart-config, $chart-data-series*) as chart extension item

以下の説明を参照ください

- $chart-config はチャート構成拡張アイテムです。順序の決まっていない 4 つのキーのシリーズです: 4 つのキーが "width"、"height"、"title"、および "kind" である値ペアです。width と height の値は整数で、チャートの幅と高さを

ピクセルで指定します。kind の値は以下のとおりです: Pie、 Pie3d、 BarChart、 BarChart3d、 BarChart3dGrouped、 LineChart、 ValueLineChart、 RoundGauge、 BarGauge。
- $chart-data-series は各配列がチャートデータ系列を定義するサイズ 3 の配列です。各配列は以下により構成されています: (i) データ系列の名前 (ii) X-軸の値 (iii) Y-軸の値。複数のデータ系列は提供される場合があります。このサンプルでは、例えば、2 つのデータ配列はそれぞれ月々の最高および最低気をんを提供します。

巻数は引数を介して提供されるデータから作成されたチャート拡張アイテムを返します。

サンプル 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

以下の説明を参照ください

- $type-name により、以下から作成される種類のチャートが選択されます: Pie、Pie3d、 BarChart、BarChart3d、 BarChart3dGrouped、 LineChart、ValueLineChart、RoundGauge、BarGauge
- $title によりチャートの名前が指定されます。

関数により、チャートの構成情報が含まれる chart-config 拡張アイテムが返されます。

altova:create-chart-config-from-xml($xml-struct) as chart-config extension item

以下の説明を参照ください

- $xml-struct はチャートの構成情報が含まれる XML 構造です。

関数により、チャートの構成情報が含まれる chart-config 拡張アイテムが返されます。この情報は XML データフラグメント 402 から与えられます。

altova:create-chart-data-series($series-name?, $x-values*, $y-values*) as chart-data-series extension item

以下の説明を参照ください

- $series-name により、系列の名前が指定されます。
- $x-values により、X-軸値のリストが与えられます。
- $y-values により、Y-軸値のリストが与えられます。

関数により、チャートの作成に必要なデータの情報(系列の名前と軸のデータ)が含まれる chart-data-series 拡張アイテムが返されます。

altova:create-chart-data-row(x, y1, y2, y3, ...) as chart-data-x-*N*y-row extension item

以下の説明を参照ください

- x はチャートデータ行の X-軸カラムにおける値です。
- yN は Y-軸カラムの値です。

関数により、1 つの系列の X-軸カラムならびに Y-軸カラムに対するデータが含まれる chart-data-x-*N*y-row 拡張アイテムが返されます。

altova:create-chart-data-series-from-rows($series-names as xs:string*, $row*) as chart-data-series extension item

以下の説明を参照ください

- $series-name は作成される系列の名前です。
- $row は系列として作成される chart-data-x-*N*y-row 拡張アイテムです。

関数により、系列のX-軸ならびにY-軸に関するデータが含まれる chart-data-series 拡張アイテムが返されます。

altova:create-chart-layer($chart-config, $chart-data-series*) as chart-layer extension item

以下の説明を参照ください

- $chart-config は altova:create-chart-config 関数または altova:create-chart-config-from-xml 関数により取得された $chart-config 拡張アイテムです。
- $chart-data-series は altova:create-chart-data-series 関数または altova:create-chart-data-series-from-rows 関数により取得された chart-data-series 拡張アイテムです。

関数により、チャートのレイヤーデータが含まれる **chart-layer** 拡張アイテムが返されます。

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*)

以下の説明を参照ください

- $chart-config は altova:create-chart-config 関数または altova:create-chart-config-from-xml 関数により取得された chart-config 拡張アイテムです。
- $chart-data-series は altova:create-chart-data-series 関数または altova:create-chart-data-series-from-rows 関数により取得された chart-data-series 拡張アイテムです。
- $chart-layer は altova:create-chart-layer 関数により取得された **chart-layer** 拡張アイテムです。

関数により、複数レイヤーのチャートアイテムが返されます。

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*, xs:boolean $mergecategoryvalues)

以下の説明を参照ください

- $chart-config は altova:create-chart-config 関数または altova:create-chart-config-from-xml 関数により取得された chart-config 拡張アイテムです。
- $chart-data-series は altova:create-chart-data-series 関数または altova:create-chart-data-series-from-rows 関数により取得された chart-data-series 拡張アイテムです。
- $chart-layer は altova:create-chart-layer 関数により取得された **chart-layer** 拡張アイテムです。
- $mergecategoryvalues は true の場合、複数のデータシリーズをマージします。false の場合はマージしません。

関数により、複数レイヤーのチャートアイテムが返されます。

8.2.1.10.1 チャートデータの XML 構造

以下にチャートデータのXML 構造、ならびにそれがAltova のチャート拡張関数398においてどのように表示されるかを示します。チャートの種類によっては、ここでの記述内容により外観が変化します。全ての要素が全種類のチャートに対して使用されるわけではなく、例えば〈Pie〉要素は棒グラフに対して無視されます。

メモ チャート関数は、**Enterprise** ならびに**サーバー エディション**のAltova 製品でしかサポートされていないことご注意してください。

```
<chart-config>
        <General
        SettingsVersion="1" must be provided
        ChartKind="BarChart"  Pie, Pie3d, BarChart, StackedBarChart, BarChart3d,
BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge, BarGauge,
CandleStick
        BKColor="#ffffff" Color
        BKColorGradientEnd="#ffffff" Color. In case of a gradient, BKColor and BKColorGradientEnd define
the gradient's colors
        BKMode="#ffffff" Solid, HorzGradient, VertGradient
        BKFile="Path+Filename" String. If file exists, its content is drawn over the background.
        BKFileMode="Stretch" Stretch, ZoomToFit, Center, Tile
        ShowBorder="1" Bool
        PlotBorderColor="#000000" Color
        PlotBKColor="#ffffff" Color
        Title="" String
        ShowLegend="1" Bool
        OutsideMargin="3.%" PercentOrPixel
        TitleToPlotMargin="3.%" PercentOrPixel
        LegendToPlotMargin="3.%" PercentOrPixel
        Orientation="vert" Enumeration: possible values are: vert, horz
        >
        <TitleFont
            Color="#000000" Color
            Name="Tahoma" String
            Bold="1" Bool
            Italic="0" Bool
            Underline="0" Bool
            MinFontHeight="10.pt" FontSize (only pt values)
            Size="8.%" FontSize />
        <LegendFont
            Color="#000000"
            Name="Tahoma"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="3.5%" />
        <AxisLabelFont
            Color="#000000"
            Name="Tahoma"
            Bold="1"
            Italic="0"
            Underline="0"
```

```
      MinFontHeight="10.pt"
      Size="5.%" />
</General>

<Line
      ConnectionShapeSize="1.%"   PercentOrPixel
      DrawFilledConnectionShapes="1"   Bool
      DrawOutlineConnectionShapes="0"   Bool
      DrawSlashConnectionShapes="0"   Bool
      DrawBackslashConnectionShapes="0"   Bool
/>

<Bar
      ShowShadow="1"   Bool
      ShadowColor="#a0a0a0"   Color
      OutlineColor="#000000"   Color
      ShowOutline="1"   Bool
/>

<Area
      Transparency="0"   UINT ( 0-255 ) 255 is fully transparent, 0 is opaque
      OutlineColor="#000000"   Color
      ShowOutline="1"   Bool
/>

<CandleStick
      FillHighClose="0"   Bool. If 0, the body is left empty. If 1, FillColorHighClose is used for
the candle body
      FillColorHighClose="#ffffff"   Color. For the candle body when close > open
      FillHighOpenWithSeriesColor="1"   Bool. If true, the series color is used to fill the
candlebody when open > close
      FillColorHighOpen="#000000"   Color. For the candle body when open > close and
FillHighOpenWithSeriesColor is false
/>

<Colors   User-defined color scheme: By default this element is empty except for the style and
has no Color attributes
      UseSubsequentColors ="1"  Boolean. If 0, then color in overlay is used. If 1, then
subsequent colors from previous chart layer is used
      Style="User"   Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"
      Colors="#52aca0"   Color: only added for user defined color set
      Colors1="#d3c15d"   Color: only added for user defined color set
      Colors2="#8971d8"   Color: only added for user defined color set
      ...
      ColorsN=""   Up to ten colors are allowed in a set: from Colors to Colors9
</Colors>

<Pie
      ShowLabels="1"   Bool
      OutlineColor="#404040"   Color
      ShowOutline="1"   Bool
      StartAngle="0."   Double
      Clockwise="1"   Bool
      Draw2dHighlights="1"   Bool
      Transparency="0"   Int (0 to 255: 0 is opaque, 255 is fully transparent)
```

```
            DropShadowColor="#c0c0c0"   Color
            DropShadowSize="5.%"   PercentOrPixel
            PieHeight="10.%"   PercentOrPixel. Pixel values might be different in the result because
of 3d tilting
            Tilt="40.0"   Double (10 to 90: The 3d tilt in degrees of a 3d pie)
            ShowDropShadow="1"   Bool
            ChartToLabelMargin="10.%"   PercentOrPixel
            AddValueToLabel="0"   Bool
            AddPercentToLabel="0"   Bool
            AddPercentToLabels_DecimalDigits="0"  UINT ( 0 – 2 )
            >
            <LabelFont
                  Color="#000000"
                  Name="Arial"
                  Bold="0"
                  Italic="0"
                  Underline="0"
                  MinFontHeight="10.pt"
                  Size="4.%"   />
      </Pie>

      <XY>
            <XAxis   Axis
                  AutoRange="1"   Bool
                  AutoRangeIncludesZero="1"   Bool
                  RangeFrom="0."   Double: manual range
                  RangeTill="1."   Double : manual range
                  LabelToAxisMargin="3.%"   PercentOrPixel
                  AxisLabel=""   String
                  AxisColor="#000000"   Color
                  AxisGridColor="#e6e6e6"   Color
                  ShowGrid="1"   Bool
                  UseAutoTick="1"   Bool
                  ManualTickInterval="1."   Double
                  AxisToChartMargin="0.px"   PercentOrPixel
                  TickSize="3.px"   PercentOrPixel
                  ShowTicks="1"   Bool
                  ShowValues="1"   Bool
                  AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom", "RightOrTop", "AtValue"
                  AxisPositionAtValue = "0"   Double
                  >
                  <ValueFont
                        Color="#000000"
                        Name="Tahoma"
                        Bold="0"
                        Italic="0"
                        Underline="0"
                        MinFontHeight="10.pt"
                        Size="3.%"   />
            </XAxis>
            <YAxis   Axis (same as for XAxis)
                  AutoRange="1"
                  AutoRangeIncludesZero="1"
                  RangeFrom="0."
                  RangeTill="1."
                  LabelToAxisMargin="3.%"
                  AxisLabel=""
```

```
                AxisColor="#000000"
                AxisGridColor="#e6e6e6"
                ShowGrid="1"
                UseAutoTick="1"
                ManualTickInterval="1."
                AxisToChartMargin="0.px"
                TickSize="3.px"
                ShowTicks="1"   Bool
                ShowValues="1"   Bool
                AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom", "RightOrTop", "AtValue"
                AxisPositionAtValue = "0"   Double
                >
                <ValueFont
                      Color="#000000"
                      Name="Tahoma"
                      Bold="0"
                      Italic="0"
                      Underline="0"
                      MinFontHeight="10.pt"
                      Size="3.%"/>
        </YAxis>
</XY>

<XY3d
        AxisAutoSize="1"   Bool: If false, XSize and YSize define the aspect ration of x and y
axis. If true, aspect ratio is equal to chart window
        XSize="100.%"   PercentOrPixel. Pixel values might be different in the result because of
3d tilting and zooming to fit chart
        YSize="100.%"   PercentOrPixel. Pixel values might be different in the result because of
3d tilting and zooming to fit chart
        SeriesMargin="30.%"   PercentOrPixel. Pixel values might be different in the result
because of 3d tilting and zooming to fit chart
        Tilt="20."   Double. -90 to +90 degrees
        Rot="20."   Double. -359 to +359 degrees
        FoV="50.">   Double. Field of view: 1-120 degree
        >
        <ZAxis
                AutoRange="1"
                AutoRangeIncludesZero="1"
                RangeFrom="0."
                RangeTill="1."
                LabelToAxisMargin="3.%"
                AxisLabel=""
                AxisColor="#000000"
                AxisGridColor="#e6e6e6"
                ShowGrid="1"
                UseAutoTick="1"
                ManualTickInterval="1."
                AxisToChartMargin="0.px"
                TickSize="3.px" >
                <ValueFont
                      Color="#000000"
                      Name="Tahoma"
                      Bold="0"
                      Italic="0"
                      Underline="0"
                      MinFontHeight="10.pt"
                      Size="3.%"/>
        </ZAxis>
```

```
    </XY3d>

    <Gauge
        MinVal="0."   Double
        MaxVal="100."   Double
        MinAngle="225"   UINT: -359-359
        SweepAngle="270"   UINT: 1-359
        BorderToTick="1.%"   PercentOrPixel
        MajorTickWidth="3.px"   PercentOrPixel
        MajorTickLength="4.%"   PercentOrPixel
        MinorTickWidth="1.px"   PercentOrPixel
        MinorTickLength="3.%"   PercentOrPixel
        BorderColor="#a0a0a0"   Color
        FillColor="#303535"   Color
        MajorTickColor="#a0c0b0"   Color
        MinorTickColor="#a0c0b0"   Color
        BorderWidth="2.%"   PercentOrPixel
        NeedleBaseWidth="1.5%"   PercentOrPixel
        NeedleBaseRadius="5.%"   PercentOrPixel
        NeedleColor="#f00000"   Color
        NeedleBaseColor="#141414"   Color
        TickToTickValueMargin="5.%"   PercentOrPixel
        MajorTickStep="10."   Double
        MinorTickStep="5."   Double
        RoundGaugeBorderToColorRange="0.%"   PercentOrPixel
        RoundGaugeColorRangeWidth ="6.%"   PercentOrPixel
        BarGaugeRadius="5.%"   PercentOrPixel
        BarGaugeMaxHeight="20.%"   PercentOrPixel
        RoundGaugeNeedleLength="45.%"   PercentOrPixel
        BarGaugeNeedleLength="3.%"   PercentOrPixel
        >
        <TicksFont
            Color="#a0c0b0"
            Name="Tahoma"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="4.%"
        />
        <ColorRanges>   User-defined color ranges. By default empty with no child element entries
            <Entry
                From="50. "   Double
                FillWithColor="1"   Bool
                Color="#00ff00"   Color
            />
            <Entry
                From="50.0"
                FillWithColor="1"
                Color="#ff0000"
            />
            ...
        </ColorRanges>
    </Gauge>
</chart-config>
```

8.2.1.10.2 チャート関数

以下のサンプルXSLTドキュメントでは、Altova のチャート拡張関数398の使用方法が示されます。更に下には、XMLドキュメントと、Altova XSLT 2.0 または3.0 エンジンとXSLTドキュメントによりXMLドキュメントが処理された結果生成される出力イメージのスクリーンショットが示されます。

メモ: チャート関数は、**Enterprise** ならびに**サーバー エディション**のAltova 製品でしかサポートされていないことに注意してください。

メモ: チャートデータテーブルの作成に関する詳しい情報については、Altova XMLSpy ならびにStyleVision 製品のドキュメンテーションを参照ください。

XSLT ドキュメント

以下にあるXSLTドキュメントでは、Altova チャート拡張関数を使用することで円グラフを生成します。更に下に記されているXMLドキュメントを処理するために使用することができます。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:altovaext="http://www.altova.com/xslt-extensions"
    exclude-result-prefixes="#all">
    <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>
                    <xsl:text>HTML Page with Embedded Chart</xsl:text>
                </title>
            </head>
            <body>
                <xsl:for-each select="/Data/Region[1]">
                    <xsl:variable name="extChartConfig" as="item()*">
                        <xsl:variable name="ext-chart-settings" as="item()*">
                            <chart-config>
                                <General
                                    SettingsVersion="1"
                                    ChartKind="Pie3d"
                                    BKColor="#ffffff"
                                    ShowBorder="1"
                                    PlotBorderColor="#000000"
                                    PlotBKColor="#ffffff"
                                    Title="{@id}"
                                    ShowLegend="1"
                                    OutsideMargin="3.2%"
                                    TitleToPlotMargin="3.%"
                                    LegendToPlotMargin="6.%"
                                    >
                                    <TitleFont
                                        Color="#023d7d"
                                        Name="Tahoma"
                                        Bold="1"
                                        Italic="0"
                                        Underline="0"
                                        MinFontHeight="10.pt"
                                        Size="8.%" />
```

```
                                    </General>
                                </chart-config>
                            </xsl:variable>
                            <xsl:sequence select="altovaext:create-chart-config-from-xml( $ext-chart-settings )"/>
                        </xsl:variable>
                        <xsl:variable name="chartDataSeries" as="item()*">
                            <xsl:variable name="chartDataRows" as="item()*">
                                <xsl:for-each select="(Year)">
                                    <xsl:sequence select="altovaext:create-chart-data-row( (@id), ( .) )"/>
                                </xsl:for-each>
                            </xsl:variable>
                            <xsl:variable name="chartDataSeriesNames" as="xs:string*" select=" ( (&quot;Series 1&quot;),
&apos;&apos; )[1]"/>
                            <xsl:sequence
                                select="altovaext:create-chart-data-series-from-rows( $chartDataSeriesNames,
$chartDataRows)"/>
                        </xsl:variable>
                        <xsl:variable name="ChartObj" select="altovaext:create-chart( $extChartConfig, ( $chartDataSeries),
false() )"/>
                        <xsl:variable name="sChartFileName" select="'mychart1.png'"/>
                        <img src="{$sChartFileName, altovaext:save-chart-image( $ChartObj, $sChartFileName, 400,
400 ) }"/>
                    </xsl:for-each>
                </body>
            </html>
        </xsl:template>
</xsl:stylesheet>
```

XML ドキュメント

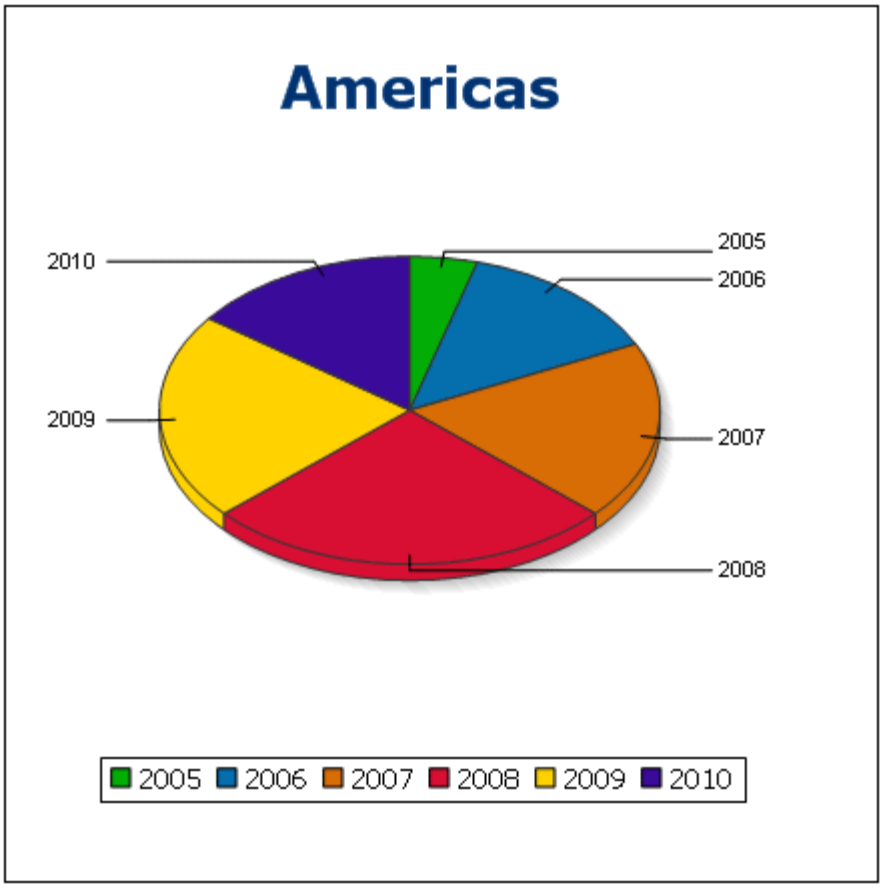
上にある XSLT ドキュメントにより以下の XML ドキュメントを処理することができます。XML ドキュメント内にあるデータを使用することで、下に示される円グラフを生成することができます。

```
<?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>

出力イメージ

上にある XSLT ドキュメントを使って XML ドキュメントを処理することで、以下にある円グラフが生成されます



8.2.1.11 バーコード関数

Altova XSLT エンジンはサードパーティー Java ライブラリを使用してバーコードを作成します。使用されるクラスや public メソッドを以下に記します。クラスは `<ProgramFilesFolder>\Altova\Common2021\jar` フォルダーにある `AltovaBarcodeExtension.jar` というパッケージに収められています。

使用される Java ライブラリは `<ProgramFilesFolder>¥Altova¥Common2021¥jar` フォルダーのサブフォルダー以下に収められています：

- `barcode4j\barcode4j.jar`（ウェブサイト: http://barcode4j.sourceforge.net/）
- `zxing\core.jar`（ウェブサイト: http://code.google.com/p/zxing/）

それぞれのフォルダーにはライセンスファイルも収められています。

Java 仮想マシン

バーコード関数を使用するには、Java 仮想マシンは使用中のマシンで使用することができます。マシンへのパスは以下で説明される通りです。

- Altova デスクトップ製品を使用している場合、Altova アプリケーションは、Java 仮想マシンへのパスを(以下の順序で)読み取ることにより自動的に検知しようとします：(i) Windows レジストリ(ii) **JAVA_HOME** 環境変数。アプリケーションのオプションダイアログ内にカスタムパスを追加することもできます。このエントリは自動的に検知された他の Java VM パス以上の優先順位を有します。
- Altova サーバー製品が Windows マシン上で作動している場合、Java 仮想マシンへのパスは Windows レジストリから最初に読み取られます。成功しない場合、**JAVA_HOME** 環境変数が使用されます。
- Altova サーバー製品を作動する場合、および、Linux または macOS マシン上でサーバー製品を作動する場合、Java 仮想マシンへのパスが **JAVA_HOME** 環境変数内に保管されていることを確認してください。**\bin\server** または **\bin\client** ディレクトリ内の **jvm.dll** ファイルを指している必要があります。

com.altova.extensions.barcode パッケージ

殆どの種類のバーコード生成には com.altova.extensions.barcode パッケージが使用されます。

以下のクラスが使用されます:

```
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** *動的に使用されるバーコードプロパティを保管するために使用されます*

```
   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** *により、*
org.krysalis.barcode4j.DefaultBarcodeClassResolver *にて登録されたクラスに加え、*qrcode *に対して*
com.altova.extensions.barcode.proxy.zxing.QRCodeBean *クラスが登録されます。*

com.altova.extensions.barcode.proxy.zxing パッケージ

QRCode バーコードの生成には com.altova.extensions.barcode.proxy.zxing パッケージが使用されます。

以下のクラスが使用されます:

class **QRCodeBean**

- org.krysalis.barcode4j.impl.AbstractBarcodeBean を拡張します

- com.google.zxing.qrcode.encoder に対して AbstractBarcodeBean インターフェースを作成します。

```
void generateBarcode(CanvasProvider canvasImp, String msg)
void setQRErrorCorrectionLevel(QRCodeErrorCorrectionLevel level)
BarcodeDimension calcDimensions(String msg)
double getVerticalQuietZone()
double getBarWidth()
```

class QRCodeErrorCorrectionLevel QRCode に対するエラー訂正レベル

```
static QRCodeErrorCorrectionLevel byName(String name)
"L" = ~7% correction
"M" = ~15% correction
"H" = ~25% correction
"Q" = ~30% correction
```

XSLT の例

以下にXSLT スタイルシートにてバーコード関数を使用するXSLT の例を示します

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:altova="http://www.altova.com"
   xmlns:altovaext=""http://www.altova.com/xslt-extensions"
   xmlns:altovaext-barcode="java:com.altova.extensions.barcode.BarcodeWrapper"
   xmlns:altovaext-barcode-
property="java:com.altova.extensions.barcode.BarcodePropertyWrapper">
   <xsl:output method="html" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
      <html>
         <head><title/></head>
         <body>
            <img alt="barcode" src="{altovaext:get-temp-folder()}barcode.png"/>
         </body>
      </html>
      <xsl:result-document
         href="{altovaext:get-temp-folder()}barcode.png"
         method="text" encoding="base64tobinary" >
         <xsl:variable name="barcodeObject"
            select="altovaext-barcode:newInstance(&apos;Code39&apos;,string(&apos;some
value&apos;),
            96,0, (altovaext-barcode-property:new( &apos;setModuleWidth&apos;, 25.4 div
96 * 2 ) ) )"/>
         <xsl:value-of select="xs:base64Binary(xs:hexBinary(string(altovaext-
barcode:generateBarcodePngAsHexString($barcodeObject)) ))"/>
      </xsl:result-document>
   </xsl:template>
</xsl:stylesheet>
```

8.2.2 その他の拡張関数

Java や C# などのプログラミング言語には、XPath 2.0 / XQuery 関数、または XSLT 2.0 関数として利用できない関数がいくつかあります。そのような関数の良い例として、Java で利用することのできる sin() や cos() といった数学関数があります。XSLT スタイルシートや XQuery のクエリにてこれらの関数が利用できるのであれば、スタイルシートやクエリの適用範囲を大幅に拡張することができ、スタイルシート作成タスクの負担が大幅に軽減されます。Altova 製品で使用されている Altova エンジン(XSLT 1.0、XSLT 2.0、XQuery 1.0)

では、Java 412 や .NET 420 および MSXSL scripts for XSLT 425 における拡張関数の使用がサポートされます。 このセクションでは、拡張機能および XSLT スタイルシート内で MSXSL スクリプト。および XQuery ドキュメントを使用する方法について記述します。使用できる拡張関数は以下のように構成されます:

- Java 拡張関数 412
- .NET 拡張関数 420
- XSLT に対する MSXSL スクリプト 425

記述の中では特に、(i) 関連するライブラリ内の関数がどのように呼ばれるか、(ii) 関数呼び出しを行う際に入力として使用される引数を変換するのにどのようなルールが適用され、return により値が返される際にどのような変換ルールが適用されるのか(XSLT/XQuery データオブジェクトに対する関数の結果)について説明されます。

必要条件

拡張関数のサポートを有効にするには、XSLT 変換や XQuery の実行を行うコンピューターに Java Runtime Environment (Java 関数にアクセスする場合)ならびに .NET Framework 2.0 以上(.NET 関数にアクセスする場合)がインストールされている、またはアクセスできる環境が整っている必要があります。

8.2.2.1 Java 拡張関数

Java 拡張関数は、XPath または XQuery 条件式にて使用することができるほか、Java のコンストラクターを呼び出したり、Java の(静的またはインスタンス)メソッドを呼び出すことができます。

Java クラスのフィールドは、引数を持たないメソッドとして扱われます。フィールドは静的またはインスタンスとして存在することができます。フィールドへのアクセス方法については、静的とインスタンスの両方について、以下のサブセクションにて記述されます。

このセクションは以下のサブセクションにより構成されます:

- Java : コンストラクター 417
- Java : 静的メソッドと静的フィールド 417
- Java : インスタンスメソッドとインスタンスフィールド 418
- データ型 : XPath/XQuery から Java へ 419
- データ型 : Java から XPath/XQuery へ 420

以下の点に注意してください

- Altova デスクトップ製品を使用している場合、Altova アプリケーションは、Java 仮想マシンへのパスを(以下の順序で)読み取ることにより自動的に検知しようとします : (i) Windows レジストリ(ii) **JAVA_HOME** 環境変数。アプリケーションのオプションダイアログ内にカスタムパスを追加することもできます。このエントリは自動的に検知された他の Java VM パス以上の優先順位を有します。
- Altova サーバー製品が Windows マシン上で作動している場合、Java 仮想マシンへのパスは Windows レジストリから最初に読み取られます。成功しない場合、**JAVA_HOME** 環境変数が使用されます。
- Altova サーバー製品を作動する場合、および、Linux または macOS マシン上でサーバー製品を作動する場合、Java 仮想マシンへのパスが **JAVA_HOME** 環境変数内に保管されていることを確認してください。**\bin\server** または **\bin\client** ディレクトリ内の **jvm.dll** ファイルを指している必要があります。

拡張関数のフォーム

XPath/XQuery 条件式における拡張関数では、`prefix:fname()` の形式を取る必要があります。

- `prefix:` 部により拡張関数が Java 関数として認識されます。`java:` から始まる URI のスコープ内の名前空間宣言に拡張関数を関連付けることで Java 関数であるという認識が行われます。名前空間の宣言により、例えば `xmlns:myns="java:java.lang.Math"` という Java クラスが特定されます。名前空間の宣言は、(コロン無しの) `xmlns:myns="java"` という形式で、Java クラスの識別子を拡張関数にある `fname()` 部の左型に配置することでも行うことができます。
- `fname()` 部により、呼び出されている Java メソッドが識別され、メソッドの引数が提供されます(以下の例を参照ください)。`prefix:` 部にて識別された名前空間 URI が Java クラスを識別できなかった場合、Java クラスの識別はクラスの前にくる `fname()` 部にて行うことになり、ピリオドによりクラスから分離されることになります(以下にある2番目の XSLT サンプルを参照)。

メモ 呼び出されるクラスはコンピューターのクラスパス上にある必要がありあす。

XSLT サンプル

以下に静的メソッドを呼び出す2つのサンプルを示します。最初のサンプルでは、クラス名 (`java.lang.Math`) が名前空間 URI に加えられており、`fname()` へ加えることはできません。2 番目のサンプルでは、`prefix:` 部に `java:` が与えられており、`fname()` 部にてクラスとメソッドが識別されます。

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

拡張関数内にあるメソッド名(上の例では `cos()`)は、名前付き Java クラス(上の例では `java.lang.Math`)の public な静的メソッドの名前に一致する必要があります。

XQuery サンプル

以下に XSLT のサンプルに似た XQuery のサンプルを示します:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

ユーザー定義された Java クラス

独自の Java クラスやメソッドを作成した場合、(i) JAR ファイル(または class ファイル)を介してこれらクラスファイルへアクセスしているか、(ii) これら(JAR または class)ファイルが、カレントディレクトリ(XSLT や XQuery ドキュメントが存在するディレクトリ)に配置されているかにより、これらクラスの呼び出し方法が変わってきます。これらファイルの特定方法については、ユーザー定義クラスファイル 413 ならびにユーザー定義 JAR ファイル 416 を参照ください。カレントディレクトリには無いクラスファイルや JAR ファイルへのパスは指定しなければならないことご注意してください。

8.2.2.1.1 ユーザー定義のクラスファイル

アクセスがクラスファイルを介したものである場合、4 つのケースが考えられます:

- クラスファイルはパッケージである。XSLT または XQuery ファイルが Java パッケージと同じ場所に収められている。(下のサンプルを参照 414)
- クラスファイルはパッケージでは無い。XSLT または XQuery ファイルが Java パッケージと同じ場所に収められている。(下のサンプルを参照 414)
- クラスファイルはパッケージである。XSLT または XQuery ファイルがランダムな場所に収められている。(下のサンプルを参照 415)
- クラスファイルはパッケージである。XSLT または XQuery ファイルがランダムな場所に収められている。(下のサンプルを参照 414)

クラスファイルがパッケージではなく、XSLT または XQuery ドキュメントと同じ場所に収められているケースを考えてみましょう。この場合、フォルダー内の全クラスを発見することができるため、ファイルの場所を指定する必要はありません。クラスの識別を行う構文は以下のようになります:

```
java:classname
```

ここで、

`java:` によりユーザー定義の Java 関数が呼ばれていることが示されます(デフォルトでカレントディレクトリにある Java クラスがロードされます)。
`classname` は目的となるメソッドのクラスが含まれているクラスの名前です。

クラスが名前空間 URI にて識別され、名前空間がメソッド呼び出しにて使用されます。

クラスファイルがパッケージで、XSLT/XQuery ファイルが Java パッケージと同じ場所に収められている

以下の例では、`com.altova.extfunc` パッケージにある `Car` クラスの `getVehicleType()` メソッドが呼び出されています。`com.altova.extfunc` パッケージは `JavaProject` という名前のフォルダーに置かれており、XSLT ファイルも同じフォルダーに配置されています。

```
<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>
```

クラスファイルが参照され、XSLT/XQuery ファイルがクラスファイルと同じフォルダーに収められている

下のサンプルでは `Car` クラスの `getVehicleType()` メソッドが呼び出されます。以下を述べることができます: (i) `Car` クラスは次のフォルダー内にあります: `JavaProject/com/altova/extfunc` 。(ii) このフォルダーは下のサンプルの現在のフォルダー内に存在します。XSLT ファイルもフォルダー `JavaProject/com/altova/extfunc` 内にあります。

```
<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>
```

クラスファイルがパッケージされXSLT/XQuery ファイルがランダムな場所に収められている

以下の例では、com.altova.extfunc パッケージにある Car クラスの getVehicleColor() メソッドが呼び出されています。com.altova.extfunc パッケージは JavaProject という名前のフォルダーに置かれており、XSLT ファイルが任意の場所に配置されています。この場合、以下のような構文でパッケージの場所をクエリ文字列として URI 内にて指定する必要があります:

```
java:classname[?path=uri-of-package]
```

ここで、

java: によりユーザー定義の Java 関数が呼ばれていることを表します。
uri-of-package は Java パッケージの URI です。
classname は目的のメソッドが含まれているクラス名です。

クラスは名前空間 URI により特定され、名前空間がメソッド呼び出しのプレフィックスで使用されます。以下の例ではカレントディレクトリ以外にあるクラスファイルへのアクセスを行うことができます。

```
<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>
```

クラスファイルがパッケージではなく、XSLT/XQuery ファイルがランダムな場所に収められている

以下の例では、com.altova.extfunc パッケージにある Car クラスの getCarColor() メソッドが呼び出されています。com.altova.extfunc パッケージは JavaProject という名前のフォルダーに置かれており、XSLT ファイルが任意の場所に配置されています。以下のような構文で、クラスファイルの場所をクエリ文字列として URI 内にて指定する必要があります:

```
java:classname[?path=uri-of-classfile]
```

ここで

java: によりユーザー定義の Java 関数が呼ばれていることを表します。
uri-of-classfile は Java パッケージの URI です。
classname は目的のメソッドが含まれているクラス名です。

クラスは名前空間 URI により特定され、名前空間はメソッド呼び出しのプレフィックスで使用されます。以下の例ではカレントディレクトリ以外にあるクラスファイルへのアクセスを行うことができます。

```
<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>
```

メモ パスが外部関数により与えられている場合、ClassLoader によりパスが追加されます。

8.2.2.1.2 ユーザー定義の JAR ファイル

JAR ファイル経由でアクセスが行われた場合、以下の構文により JAR ファイルの URI を指定する必要があります:

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

クラスの識別を行う名前空間 URI のプレフィックスを使用してメソッドが呼び出されます: `classNS:method()`

上の例に対する説明は以下のとおりです:

`java:` 関数が呼び出されていることを表します。
`classname` がユーザー定義されたクラスの名前になります。
`?` はクラス名とパスを分離するために使用されます。
`path=jar:` により、JAR ファイルへのパスが与えられていることを示します。
`uri-of-jarfile` は JAR ファイルの URI となります。
`!/` は、終了を表すデリミタとなります。
`classNS:method()` により、メソッドの呼び出しが行われます。

その他にも、メソッド名とともにクラス名を与えることができます。構文の例を以下に示します:

```
              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()
```

以下に JAR ファイルを使った Java 拡張関数を呼び出す XSLT サンプルを記します:

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

```
<xsl:template match="/">
      <xsl:variable name="myCar" select="car:Car1.new('red')" />
      <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

メモ: 拡張関数によりパスが与えられている場合、ClassLoader にパスが追加されます。

8.2.2.1.3 Java:コンストラクター

拡張関数を使用することで Java コンストラクターを呼び出すことができます。`new()` により全てのコンストラクターを呼び出すことができます。

Java コンストラクターの呼び出し結果を、黙示的にXPath/XQuery データ型へ変換 420 できる場合、Java 拡張関数により XPath/XQuery データ型のシーケンスが返されます。Java コンストラクターの呼び出し結果がXPath/XQuery データ型へ変換できない場合、値を返すクラス名でラップした Java オブジェクトがコンストラクターにより作成されます。例えば、`java.util.Date` クラスに対するコンストラクターが呼び出された場合(`java.util.Date.new()`)、`java.util.Date` を持ったオブジェクトが返されます。返されたオブジェクトのレキシカルフォーマットは、XPath データ型のレキシカルフォーマットにマッチしない場合もあり、目的のXPath データ型に対するレキシカルフォーマットへ値の変換を行い、その後目的のXPath データ型へ変換を行う必要があります。

コンストラクターにより作成された Java オブジェクトにより2つのことが行えます:

- 変数への割り当てを行うことができます:
  ```
  <xsl:variable name="currentdate" select="date:new()"
  xmlns:date="java:java.util.Date" />
  ```
- 拡張関数への受け渡しを行うことができます(*インスタンスメソッドならびにインスタンスフィールド* 418 を参照ください):
  ```
  <xsl:value-of select="date:toString(date:new())"
  xmlns:date="java:java.util.Date" />
  ```

8.2.2.1.4 Java:静的メソッドと静的フィールド

静的メソッドは、Java 名ならびにメソッドの引数により直接呼び出すことができます。`E` や `PI` といった定数の静的フィールド(引数を持たないメソッド)は、引数を指定することなくアクセスすることができます。

XSLT の例

静的メソッドならびにフィールドを呼び出す例を以下に示します:

```
<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)" />
```

上の拡張関数は prefix:fname() という形式を使用していることに注意してください。3つの例にあるプレフィックスは全て jMath: となっており、このプレフィックスは、java:java.lang.Math という名前空間 URI に関連付けられています。名前空間 URI は java: で開始しなければなりません。上の例では、クラス名(java.lang.Math)を含むように拡張されています。拡張関数の fname() 部は(java.lang.Math のような) public クラスにマッチする必要があり、その後 public な静的メソッドが引数とともに続くか(例: cos(3.14))、public な静的フィールドが続きます(例: PI())。

上の例では、クラス名が名前空間 URI に含まれています。クラス名が名前空間 URI に含まれていない場合、以下の例にあるように、拡張関数の fname() 部にて追加する必要があります:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery の例

XQuery における似たようなサンプルを以下に示します:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

8.2.2.1.5 Java:インスタンスメソッドとインスタンスフィールド

メソッド呼び出しの第一引数として パースされる Java オブジェクトが、インスタンスメソッドには与えられています。このような Java オブジェクトは通常、拡張関数を使うことで作成される(例:コンストラクターの呼び出し)か、スタイルシート パラメーター/変数により作成されます。以下に XSLT サンプルを示します:

```
<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>
```

上の例では、ノード enrollment/@type の値が以下のように作成されます:

1. java.util.Date クラスに対するオブジェクトがコンストラクター(date:new() コンストラクター)とともに作成されます。
2. jlang.Object.getClass メソッドの引数として Java オブジェクトがパースされます。
3. getClass メソッドにより得られたオブジェクトが jlang.Object.toString メソッドの引数としてパースされます。

結果(@type の値)は java.util.Date を持つた文字列となります。

インスタンスフィールドは、引数としてインスタンスフィールドへ渡される Java オブジェクトではないという点で、理論的にはインスタンスメソッドと異なります。パラメーターや変数がその代わりに引数として渡されますが、パラメーター/変数そのものに Java オブジェクトから返された値が含まれている場合もあります。例えば、CurrentDate パラメーターには java.util.Date クラスのコンストラクターから返された値が含まれます。この値は引数として、date:toString インスタンスメソッドへ渡され、/enrollment/@date の値として使用されます。

8.2.2.1.6 データ型：XPath/XQuery から Java へ

XPath/XQuery 条件式内部から Java 関数が呼び出された場合、複数ある同名の Java クラスのうち、どのクラスが呼び出されたのか決定するのに、関数へ渡される引数のデータ型が重要になります。

Java では、以下のルールが適用されます:

- 同名の Java メソッドが2つ以上あり、それぞれが違う数の引数を受け取る場合、呼び出しに使用されている引数の数に一番マッチするメソッドが選択されます。
- XPath/XQuery の文字列、数値、boolean データ型は、黙示的に対応する Java データ型へ変換されます（以下のリストを参照）。与えられた XPath/XQuery 型が2つ以上の Java 型へ変換できる場合（例: `xs:integer`）、選択されたメソッドにて宣言されている Java 型が使用されます。例えば、呼び出された Java メソッドが `fx(decimal)` で、与えられた XPath/XQuery データ型が `xs:integer` の場合、`xs:integer` が Java の `decimal` データ型へ変換されます。

以下のテーブルに、XPath/XQuery の文字列、数値、boolean 型から Java データ型への黙示的な変換リストを示します。

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean`（プリミティブ型）, `java.lang.Boolean`
`xs:integer`	`int`, `long`, `short`, `byte`, `float`, `double`, ならびに、`java.lang.Integer` のようなこれらのラッパークラス
`xs:float`	`float`（プリミティブ型）, `java.lang.Float`, `double`（プリミティブ型）
`xs:double`	`double`（プリミティブ型）, `java.lang.Double`
`xs:decimal`	`float`（プリミティブ型）, `java.lang.Float`, `double`（プリミティブ型）, `java.lang.Double`

上のリストにある XML スキーマデータ型（ならびに XPath や XQuery で使用されているデータ型）のサブタイプも、対応する祖先のサブタイプとして Java のデータ型へ変換されます。

場合によっては、与えられた情報から正しい Java メソッドを選択することができない場合もあります。例えば、以下のような場合を考えてみましょう:

- 与えられた引数が 10 という値を持った xs:untypedAtomic 型で、`mymethod(float)` メソッドへ渡されるのを意図している。
- しかし、そのクラスには別のデータ型を取る `mymethod(double)` というメソッドも存在する。
- メソッド名が同じで、与えられた型（`xs:untypedAtomic`）も `float` と `double` の両方に変換することができるため、`xs:untypedAtomic` が `float` ではなく `double` に変換される可能性もある。
- 結果として、意図したメソッドは選択されず、予期しない動作結果が招く可能性がある。この問題を回避するには、意図したメソッドを使用するユーザー定義のメソッドを別の名前で新たに作成する必要があります。

上のリストでカバーされていない型（例: `xs:date`）は変換されず、エラーとなります。しかし場合によっては、Java コンストラクターを使用して、目的の Java データ型を作成することが可能だということも留意してください。

8.2.2.1.7 データ型：Java から XPath/XQuery へ

Java メソッドにより値が返され、値のデータ型が文字列、数値、または boolean 型の場合、対応する XPath/XQuery 型への変換が行われます。例えば、Java の `java.lang.Boolean` や `boolean` データ型は `xsd:boolean` へ変換されます。

関数から返された一次元配列は、シーケンス（sequence）に展開されます。2 次元以上の配列は変換されることが無いため、ラップして使用するべきでしょう。

Java オブジェクトや文字列、数値、boolean 以外のデータ型がラップされて返された場合、最初に（例えば `toString` といった）Java メソッドを使用して Java オブジェクトを文字列へ変換することで、目的の XPath/XQuery 型への変換を行います。XPath/XQuery では、文字列を目的となる型のレキシカルフォーマットへ変換し、目的の型への変換を（例えば `cast as` 式を使用することで）行うことができます。

8.2.2.2 .NET 拡張関数

.NET プラットフォームにて作業を行なっている場合、.NET 言語（例えば C#）で記述された拡張関数を使用することができます。.NET 拡張関数は XPath や XQuery 条件式内部から使用することができ、.NET クラス内部にあるコンストラクターや（static または インスタンス変数）プロパティを呼び出すことができます。

`get_PropertyName()` 構文を使用することにより .NET クラスのプロパティを呼び出すことができます。

このセクションは、以下のサブセクションにより構成されています:

- .NET コンストラクター 422
- .NET：静的メソッドならびに静的フィールド 423
- .NET：インスタンスメソッドとインスタンスフィールド 423
- データ型：XPath/XQuery から .NET へ 424
- データ型：.NET から XPath/XQuery へ 425

拡張関数のフォーム

XPath/XQuery 条件式にある拡張関数は、`prefix:fname()` の形式を取る必要があります。

- `prefix:` 部は、呼び出されている .NET クラスを特定する URI となります。
- `fname()` 部により、.NET クラス内にあるコンストラクター、プロパティ、または（静的または インスタンス）メソッドが特定され、必要な場合は引数が与えられます。
- URI は `clitype:` で開始する必要があり、これにより関数が .NET 拡張関数であることが認識されます。
- 拡張関数の `prefix:fname()` 形式は、システムクラスならびにロードされたアセンブリとともに使用することもできます。しかしクラスをロードする必要がある場合、必要な情報が含まれるパラメーターが必要になります。

パラメーター

アセンブリをロードするには以下のパラメーターを使用してください:

`asm`	ロードするアセンブリの名前。
`ver`	バージョン番号（ピリオドにより分離された最大 4 桁の整数）。

sn	アセンブリ厳密名のキートークン(16新数の数値)。
from	ロードするアセンブリ(DLL) の場所を特定するURI。URI が相対パスの場合、XSLT やXQuery ドキュメントに対して相対的になります。このパラメーターが指定された場合、その他のパラメーターが無視されます。
partialname	アセンブリ名の一部。Assembly.LoadWith.PartialName() へ渡され、アセンブリへのロードが試みられます。 partialname が指定された場合、その他のパラメーターが無視されます。
loc	例えばen-US というロケール。デフォルトはneutral です。

アセンブリがDLL からロードされる場合、 from パラメータが使用して、sn パラメーターは使用しないでください。アセンブリがグローバルアセンブリキャッシュ(GAC) からロードされる場合、sn パラメータを使用して from パラメーターは使用しないでください。

最初のパラメーターの前に疑問符 (?) を挿入し、パラメーター同士はセミコロンで分離する必要があります。パラメーター名へ値を受け渡すには、統合符号 (=) を使用します(以下の例を参照ください)。

名前空間宣言の例

XSLT において、システムクラス System.Environment を特定する名前空間宣言の例を以下に示します:

```
xmlns:myns="clitype:System.Environment"
```

XSLT において、ロードするクラスを Trade.Forward.Scrip として特定する名前空間宣言の例を以下に示します:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

XQuery において、システムクラス MyManagedDLL.testClass を特定する名前空間宣言の例を以下に示します。2つのケースが考えられます:

1. アセンブリがGAC からロードされた場合:
```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
       ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```
2. アセンブリがDLL からロードされた場合(完全参照と一部の参照):
```
       declare namespace cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
       Projects/extFunctions/MyManagedDLL.dll;

declare namespace cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

XSLT の例

システムクラス System.Math 内の関数を呼び出すための完全な XSLT の例を以下に示します:

```
<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>
```

math 要素にある名前空間宣言により、math: プレフィックスとclitype:System.Math URI が関連付けられます。URI の最初にある clitype:により、それ以降の記述がシステムクラスまたはロードされたクラスを特定するものであることが示されます。XPath 条件式にあるmath: プレフィックスにより、拡張関数がURI（そしてクラス）System.Math に関連付けられます。拡張関数により、System.Math クラス内のメソッドが特定され、必要な場所に引数が与えられます。

XQuery の例

上のXSLT に対する例と同様のXQuery 例を以下に示します:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

上のXSLT と同様に、名前空間宣言により.NET クラス（この場合はシステムクラス）が特定されます。XQuery 式により呼び出されるメソッドが特定され、引数が与えられます。

8.2.2.2.1 .NET コンストラクター

拡張関数を使用することで、.NET コンストラクターを呼び出すことができます。new() により全てのコンストラクターを呼び出すことができます。クラス内に2つ以上のコンストラクターがある場合、与えられた引数の数が最もマッチするコンストラクターが選択されます。与えられた引数に対してマッチするコンストラクターが見つからない場合、"No constructor found'" エラーが返されます。

XPath/XQuery データ型を返すコンストラクター

.NET コンストラクター呼び出しの結果が XPath/XQuery データ型へ黙示的に変換 420 することができる場合、.NET 拡張関数から XPath/XQuery データ型のシーケンスが返されます。

.NET オブジェクトを返すコンストラクター

.NET コンストラクター呼び出しの結果がXPath/XQuery データ型へ適切に変換できない場合、値を返すクラス名でラップした.NET オブジェクトがコンストラクターにより作成されます。例えば、System.DateTime クラスのコンストラクターが（System.DateTime.new() により）呼ばれた場合 System.DateTime 型を持ったオブジェクトが返されます。

返されたオブジェクトのレキシカルフォーマットは、目的のXPath データ型と違っている場合があります。その場合、返された値を:（i）目的のXPath データ型のレキシカルフォーマットへ変換し、（ii）目的のXPath データ型へキャストする必要があります。

コンストラクターにより作成された.NET オブジェクトに対して3つのことを行うことができます:

- 変数内で使用することができます:
  ```
  <xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
  ```
- 拡張関数へ渡すことができます（*インスタンスメソッドとインスタンスフィールド* 418 を参照ください）:
  ```
  <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
  xmlns:date="clitype:System.DateTime" />
  ```
- 文字列、数値、またはboolean へ変換することができます:

```
<xsl:value-of select="xs:integer(date:get_Month(date:new(2008, 4, 29)))"
 xmlns:date="clitype:System.DateTime" />
```

8.2.2.2.2 .NET : 静的メソッドと静的フィールド

メソッド名と引数を与えることで、静的メソッドを直接呼び出すことができます。呼び出しに使用される名前は、クラス内にある public static メソッドと完全に一致する必要があります。関数の呼び出しに使用されたメソッド名と引数の数にマッチするものがクラス内に複数ある場合、与えられた引数が評価され、最もマッチするものが選択されます。マッチする結果が得られない場合、エラーが返されます。

メモ: .NET クラス内にあるフィールドは、引数を持たないメソッドとしてみなされます。プロパティは `get_PropertyName()` 構文により呼び出されます。

例

1つの引数とともにメソッド (`System.Math.Sin(arg)`) を呼び出す XSLT サンプルを以下に示します:

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

(引数なしのメソッドとしてみなされる) フィールド (`System.Double.MaxValue()`) を呼び出す XSLT サンプルを以下に示します:

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

(`get_PropertyName()` 構文を使って) プロパティ (`System.String()`) を呼び出す XSLT サンプルを以下に示します:

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

1つの引数とともにメソッド ((`System.Math.Sin(arg)`) を呼び出す XQuery サンプルを以下に示します:

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

8.2.2.2.3 .NET : インスタンスメソッドとインスタンスフィールド

インスタンスメソッドには、メソッド呼び出しの第一引数として .NET オブジェクトが渡されます。通常この .NET オブジェクトは、拡張関数 (例えばコンストラクター呼び出し) またはスタイルシート パラメーター/変数により作成されます。以下に XSLT の例を示します:

```
<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>
```

上の例では、System.DateTime コンストラクター(`new(2008, 4, 29)`) が、System.DateTime 型の.NET オブジェクトの作成に使用されます。このオブジェクトは、最初に`releasedate` 変数の値として、次に`System.DateTime.ToString()` メソッドの引数として作成されます。`System.DateTime.ToString()` インスタンスメソッドは、`System.DateTime` コンストラクターの (new(2008, 4, 29)) における引数として2度呼び出されます。これらインスタンスにおいて `releasedate` 変数が.NET オブジェクトを取得するのに使用されます。

インスタンスメソッドとインスタンスフィールド

インスタンスメソッドとインスタンスフィールドの違いは理論的なものです。インスタンスメソッドでは.NET オブジェクトが直接引数に渡され、インスタンスフィールドではパラメーターや変数が(.NET オブジェクトそのものを含めることはできるものの) 代わりに渡されます。例えば上の例では、`releasedate` 変数に.NET オブジェクトが含まれており、この変数が2番目のdate 要素コンストラクターにて `ToString()` の引数として渡されます。そのため、最初のdate 要素にある `ToString()` インスタンスがインスタンスメソッドであるのに対し、2番目はインスタンスフィールドとしてみなされます。両方のインスタンスで求められる結果は等価です。

8.2.2.2.4 データ型:XPath/XQuery から .NET へ

.NET 拡張関数がXPath/XQuery 条件式内部で使用された場合、複数ある.NET メソッドのうち、どれが呼び出されたのが決定するのに関数の引数に使用されるデータ型が重要になります。

.NET では、以下のルールが適用されます:

- クラス内に同名のメソッドが2つ以上ある場合、呼び出しに使用された引数の数がマッチするメソッドだけが、呼び出される関数の候補に狭められます。
- XPath/XQuery の文字列、数値、boolean データ型は黙示的に対応する.NET データ型へ変換されます(以下のリストを参照)。与えられたXPath/XQuery 型が2つ以上の.NET 型へ変換できる場合(例: `xs:integer`)、選択されたメソッドにて宣言されている.NET 型が使用されます。例えば、呼び出された.NET メソッドがfx(fouble) で、与えられたXPath/XQuery データ型がxs:integer の場合、`xs:integer` が.NET の`double` データ型へ変換されます。

以下のテーブルに、XPath/XQuery の文字列、数値、boolean 型から.NET データ型へ行われる黙示的な変換リストを示します。

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

上のリストにある XML スキーマ型(ならびに XPath や XQuery で使用されているデータ型)のサブタイプも、対応する祖先のサブタイプとして .NET のデータ型へ変換されます。

場合によっては、与えられた情報から正しい .NET メソッドを選択することができない場合もあります。例えば、以下のような場合を考えてみましょう:

- 与えられた引数が 0 ある xs:untypedAtomic 値で、mymethod(float) メソッドへ渡されるのを意図している。
- しかし、そのクラスには別のデータ型をとる mymethod(double) というメソッドも存在する。
- メソッド名が同じで、与えられた型 (xs:untypedAtomic) も float と double の両方に変換することができるため、xs:untypedAtomic が float ではなく double に変換される可能性もある。
- 結果として、意図したメソッドは選択されず、予期しない動作結果がを招く可能性がある。この問題を回避するには、意図したメソッドを使用するユーザー定義のメソッドを別の名前で新たに作成する必要があります。

上のリストでカバーされていない型(例: xs:date)は変換されずエラーとなります。

8.2.2.2.5 データ型 : .NET から XPath/XQuery へ

.NET メソッドにより値が返される際に値のデータ型が文字列、数値、または boolean 型の場合、対応する XPath/XQuery 型への変換が行われます。例えば、.NET の decimal データ型は xsd:decimal へ変換されます。

.NET オブジェクトや文字列、数値、boolean 以外のデータ型が返された場合、最初に(例えば System.DateTime.ToString() といった).NET メソッドを使用して .NET オブジェクトを文字列へ変換します。XPath/XQuery では、文字列を目的となる型のレキシカルフォーマットへ変換し、目的の型への変換を(例えば cast as 式を使用することで)行うことができます。

8.2.2.3 XSLT に対する MSXSL スクリプト

<msxsl:script> 要素にはユーザー定義の関数や変数が含まれており、XSLT スタイルシート内の XPath 条件式内部から呼び出しを行うことができます。<msxsl:script> はトップレベル要素で、<xsl:stylesheet> または <xsl:transform> の子要素である必要があります。

<msxsl:script> 要素は urn:schemas-microsoft-com:xslt 名前空間内に存在する必要があります(以下を参照ください)。

スクリプト言語と名前空間

ブロック内で使用されるスクリプト言語は <msxsl:script> 要素の language 属性にて指定され、XPath 条件式における関数の呼び出しに対して使用される名前空間は implements-prefix 属性により特定されます(以下を参照)。

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

<msxsl:script> 要素はWindows Scripting Runtime を使ったやりとりを行うため、お使いのコンピューターにインストールされた言語だけしか<msxsl:script> 要素では使用することができません。MSXSL スクリプトを使用するには.NET Framework 2.0 以上のプラットフォームをインストールする必要があります。結果として <msxsl:script> 言語から.NET スクリプト言語を使用することができます。

HTML の<script> 要素における language 属性と同じ値が language 属性では受理されます。language 属性が指定されていない場合、Microsoft JScript がデフォルトとして想定されます。

implements-prefix 属性には名前空間スコープ内で宣言されたプレフィックスが与えられます。通常この名前空間は関数ライブラリのために予約されたユーザーの名前空間となります。<msxsl:script>要素内で定義された全ての関数ならびに変数は、 implements-prefix 属性にて指定されたプレフィックスで特定される名前空間に収められます。XPath 条件式内部から関数が呼ばれる場合、完全修飾関数名が同じ名前空間内に関数として定義されていなければなりません。

サンプル

<msxsl:script> 要素内で定義された関数を使用するXSLT スタイルシートの例を以下に示します:

```
<?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>
```

データ型

スクリプトブロックとのやりとりに使用されるパラメーターの値はXPath データ型に限定されます。スクリプトブロック内にある関数にてやりとりされるデータや変数に、この制限はありません。

アセンブリ

msxsl:assembly 要素を使用することで、アセンブリをスクリプト内部へインポートすることができます。アセンブリは名前やURI により特定されます。アセンブリのインポートは、コンパイル時に行われます。以下に msxsl:assembly 要素の簡単な使用例を示します:

```
<msxsl:script>
       <msxsl:assembly name="myAssembly.assemblyName" />
       <msxsl:assembly href="pathToAssembly" />

       ...

</msxsl:script>
```

アセンブリ名は、以下のような完全な名前でも::

```
"system.Math, Version=3.1.4500.1 Culture=neutral PublicKeyToken=a46b3f648229c514"
```

`"myAssembly.Draw"` のような短い名前でも指定できます。

名前空間

msxsl:using 要素により名前空間の宣言を行うことができます。これにより、スクリプト内において名前空間無しでアセンブリクラスを使用することができ、タイピングの手間を軽減することができます。以下に `msxsl:using` 要素の簡単な使用例を示します:

```
<msxsl:script>
       <msxsl:using namespace="myAssemblyNS.NamespaceName" />

       ...

</msxsl:script>
```

`namespace` 属性の値は名前空間の名前となります。

インデックス

.

.NET Framework API, 314
.NET インターフェイス, 10
.NET 拡張関数,
  XSLT と XQuery, 420
  インスタンスメソッド、インスタンスフィールド, 423
  コンストラクター, 422
  データ型変換、.NET から XPath/XQuery へ, 425
  データ型変換、XPath/XQuery から .NET へ, 424
  概要, 420
  静的メソッド、静的フィールド, 423
.NET 内のXSLT と XQuery のための拡張機能,
  .NET 拡張関数を参照する, 420

A

Altova 拡張関数,
  チャート関数 (チャート関数を参照), 326

C

CLI 上のヘルプコマンド, 163
COM インターフェイス, 10
CoreCatalog.xml, 43
CustomCatalog.xml, 43

D

DTD とカタログ, 41

H

HTTP インターフェイス, 10, 200
  クライアントリクエスト, 211
  サーバーセットアップ, 201
  サーバーの構成, 203
  セキュリティの問題, 51

J

Java インターフェイス, 10
Java 拡張関数,
  XSLT と XQuery, 412
  インスタンスメソッド、インスタンスフィールド, 418
  コンストラクター, 417
  データ型変換、Java から XPath/XQuery へ, 420
  データ型変換、XPath/XQuery から Java へ, 419
  ユーザー定義の JAR ファイル, 416
  ユーザー定義のクラスファイル, 413
  概要, 412
  静的メソッド、静的フィールド, 417
JSON 構成ファイル,
  RaptorXMLServer Python モジュールのための, 309

L

License commands on CLI, 167
Linux,
  でのライセンス, 31
  へのインストール, 28
Linux でのライセンス, 31
Linux へのインストール, 28

M

macOS,
  でのライセンス, 37
  へのインストール, 35
macOS でのライセンス, 37
macOS へのインストール, 35
MSXSL スクリプト内の拡張関数, 425
msxsl:script, 425

P

pip コマンド, 309
Python,
セキュリティの問題, 51
Python API, 306
Python API よくある質問, 313
Python モジュール,
RaptorXML Server の, 309
Python ライブラリ,
RaptorXML Server の, 309
Pythonインターフェイス, 10

R

RaptorXML,
COM、Java、.NET を持つインターフェイス, 10
HTTP インターフェイス, 10
Pythonインターフェイス, 10
エディションとインターフェイス, 10
コマンドラインインターフェイス, 10
サポートされる仕様, 17
システムの必要条件, 14
はじめに, 9
機能, 15
RaptorXML Server APIs, 305
RaptorXML Server上の Python スクリプト, 311
RaptorXML 内のカタログ, 43
RaptorXMLServer Python モジュールのインストール, 309
RootCatalog.xml, 43, 309

V

Visua Studio と Python スクリプト, 311
Visual Studio コード内のスクリプトの Python デバッグ, 311

W

Windows,
でのライセンス, 24
へのインストール, 21
Windows でのライセンス, 24
Windows へのインストール, 21

X

XML カタログ, 40
XQuery,
拡張関数, 411
XQuery コマンド, 82
XQuery ドキュメント 検証, 95
XQuery 実行, 82
XSLT,
拡張関数, 411
XSLT コマンド, 107
XSLT と XQuery のためのJava 拡張関数,
Java 拡張関数を参照する, 412
XSLT と XQuery のための拡張関数, 411
XSLT ドキュメント 検証, 114
XSLT 変換, 107
XSLT/XQuery 内のスクリプト,
拡張関数で参照する, 411

Z

インターフェイス,
概要, 10
カタログ, 40
カタログと環境変数, 47
カタログのカスタム化, 45
カタログメカニズムの概要, 41
カタログ内で使用される環境変数, 43
グローバルリソース, 49
コア, 20
コマンドライン,
XQuery, 82
オプション, 184
使用方法の概要, 52
サーバーの構成, 203
サーバー側の Python スクリプトのデバッグ, 311
スキーマ,
カタログを介したルックアップ, 45
スキーマとカタログ, 41

セキュリティの問題, 51
セットアップ,
Linux での, 28
Mac OS X での, 35
チャート関数,
サンプル, 407
チャートデータ構造, 402
リスト, 398
プロセッサーコア, 20
ライセンスとプロセッサーコア, 20
ローカライズ, 164
環境変数, 47
検証,
DTD, 62
DTD を使用して XML インスタンスの, 53
XQuery ドキュメント, 95
XSD, 66
XSD を持つ XML インスタンス, 56
XSLT ドキュメント, 114
整形式チェック コマンド, 72
設定,
Windows 上, 21
単一スレッドの実行, 20