Altova RaptorXML+XBRL Server 2017

ユーザーマニュアル

Altova RaptorXML+XBRL Server 2017ユーザーマニュアル

発行日 :2017

目次

チャプター 1

RaptorXML+XBRL Server についての説明

1 RaptorXML+XBRL Server についての説明

Altova RaptorXML+XBRL 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。

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のセクションで説明されています。RaptorXML は、Windows、Linux、およびMac OS X 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 およびXBRL 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、機能のセクションを参照してください。サポートされる仕様 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。

このドキュメント

このドキュメントは、Altova Web 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ぐ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)
- RaptorXML のセットアップ
- コマンドライン インターフェイス
- HTTP インターフェイス
- Python インターフェイス
- Java インターフェイス
- COM/.NET インターフェイス
- XSLT とXQuery エンジンの情報
- XSLT とXPath/XQuery 関数
- Altova LicenseServer

最終更新日: 2017年 04月 27日

1.1 エディションとインターフェイス

RaptorXML には以下のエディションがあります :

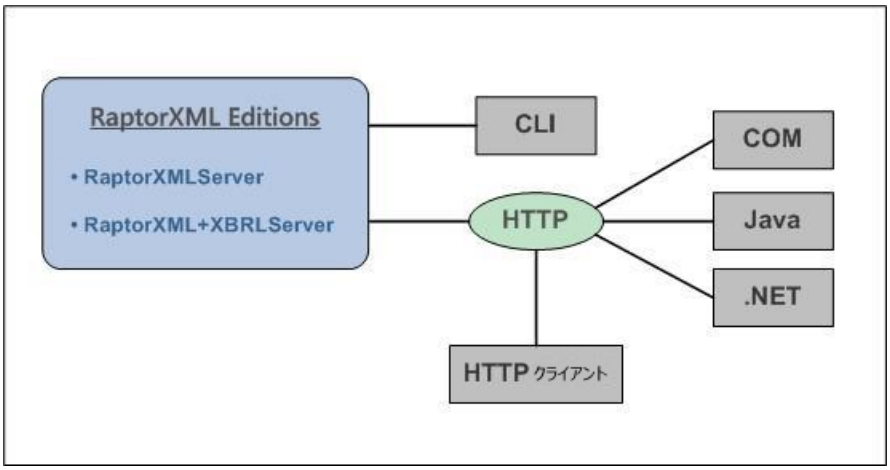
- *RaptorXML Server* は、XML、XML スキーマ、XSLT、XPath、XQuery などをサポートする高速の XML 処理 エンジンです。
- *RaptorXML+XBRL Server* は、RaptorXML Server の全ての機能と、追加処理機能および、XBRL 一連の標準の検証をサポートします。

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)
- Windows システム上の COM インターフェイス。
- Windows システム上の .NET インターフェイス。
- Windows、Linux、および MacOS システム上の Java インターフェイス。
- HTTP クライアントからアクセスすることのできる HTTP インターフェイス。
- Python API を介して、Python スクリプトがドキュメント部分にアクセスし処理することのできる、Python インターフェイス。スクリプトは CLI または HTTP を介して送信することができます。更に、RaptorXML 機能は Python パッケージとして使用することができます。これにより、RaptorXML 機能が、他の第三者パッケージと共に Python コード内で使用することができるようになります。
- .NET Framework API

インターフェイスを介して RaptorXML へアクセスする方法が下の図に表示されています。



COM、Java、および .NET インターフェイスは、サービスエディションに接続するために、HTTP プロトコールを使用することに注意してください。Python スクリプトは、コマンドラインと HTTP インターフェイスを介して、サーバーエディションに送信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)

XML (そして他のドキュメント) 検証 ,XSLT 変換、および XQuery 実行のためのコマンドラインの使用方法を提供します。使用情報のためのコマンドラインのセクションを参照してください。

HTTP インターフェイス

サーバーエディションの全ての機能はHTTP インターフェイスにより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。サーバーは、クライアントに応答し、ジョブに関連するすべての情報を含みます。ます。 HTTP インターフェイスのセクションを参照してください。

Python API
CLI コマンドまたはHTTP リクエストとともに、コマンドまたはリクエスト内で指定されたドキュメントをアクセスするための Python スクリプトを送信することができます。ドキュメントへのアクセスは、XML、XSD、およびXBRLのためのPython API により提供されます。使用方法の説明 および異なる種類の API に関しては Python インターフェイスのセクションを参照してください。RaptorXML 機能はPython パッケージとして使用することができます。これにより、RaptorXML 機能が、他の第三者パッケージと共にPython コード内で使用することができるようになります。

.NET Framework API
.NET Framework API により、NET Framework をサポートするアプリケーションRaptorXML+XBRL Server に埋め込むことができます。API に関する詳細は、.NET Framework API を参照してください。

COM インターフェイス
RaptorXML は、COM インターフェイスを介して使用されることができます。ですから、COMをサポートするアプリケーションとスクリプト言語により使用されることができます。COM インターフェイスは、実装された行とDispatch をサポートします。入力データは、スクリプト内とアプリケーションデータ内のファイルとしてまたはテキスト文字列として提供されることができます。

Java インターフェイス
RaptorXML 機能は、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。 例えば、XML 検証、XSLT 変換、およびXQuery 実行 機能を提供するJava クラスがあります。

.NET インターフェイス
DLL ファイルは、以下のようにラッパー としてRaptorXML の周りに構築されます。そして、 NET ユーザーにRaptorXML 機能への接続を許可します。RaptorXML は、Altova により署名された、プライマリ相互運用機能アセンブリです。入力データは、ファイルとして、またはアプリケーションデータ内で、ファイルとしてまたはテキスト文字列として提供されることができます。

1.2 システムの必要条件

RaptorXML+XBRL 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*
  Windows Vista, Windows 7/8/10

- *Windows Server*
  Windows Server 2008 R2 または以降

- *Linux*
  - CentOS 6 または以降
  - RedHat 6 または以降
  - Debian 7 または以降
  - Ubuntu 12.04 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+XBRL Server	qt4, krb5-libs, qt-x11	libqtcore4, libqtgui4, libgssapi-krb5-2	libqtcore4, libqtgui4, libgssapi-krb5-2

- *Mac OS X*
  OS X 10.10、 10.11、macOS 10.12 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。

1.3 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、XBRL、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 および XBRL 検証

- 与えられた XML または XBRL 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。
- XBRL タクソノミ および XBRL ドキュメントを XBRL タクソノミに対して検証。
- XBRL フォーミュラ および 検証アサーションの実行
- XBRL テーブルの表示
- XBRL 2.1、Dimensions 1.0、および Formula 1.0 仕様、および Table Linkbase 1.0 へのサポート。
- インライン XBRL へのサポート
- XBRL タクソノミ パッケージへのサポート

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 ビットと 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 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 と XBRL 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、XBRL、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 および Altova Web サイト のセクションを参照してください。

1.4 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 Standards

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
- Apache Avro Schema

XBRL 勧告

Web サイト:拡張可能な事業報告言語 Extensible Business Reporting Language (XBRL)

- XBRL 2.1
- Dimensions 1.0
- Extensible Enumerations 1.0
- Formula Specifications 1.0
  - Aspect Cover Filters
  - Assertion Severity 1.0
  - Boolean Filters
  - Concept Filters
  - Concept Relation Filters
  - Consistency Assertions
  - Custom Function Implementation
  - Dimension Filters
  - Entity Filters
  - Existence Assertions
  - Formula
  - Function Registry
  - General Filters
  - Generic Messages
  - Implicit Filters
  - Match Filters
  - Period Filters
  - Relative Filters
  - Segment Scenario Filters
  - Tuple Filters
  - Unit Filters
  - Validation
  - Validation Messages
  - Value Assertions
  - Value Filters
  - Variables
- Table Linkbase 1.0
- Function Registry 1.0
- Generic Links 1.0
  - Generic References
  - Generic Labels
- Units Registry 1.0
- Inline XBRL 1.0 and 1.1
- Taxonomy Packages 1.0

チャプター 2

RaptorXML のセットアップ

2 RaptorXML のセットアップ

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

- Windows、Linux および Mac OS X システムへの RaptorXML のインストールとライセンス。
- XML カタログの使用方法。
- Altova グローバルリソースとの作業方法。
- RaptorXML に関連したセキュリティの問題

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 および Altova グローバルリソース 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のセクションで説明されています。

2.1 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+XBRL Server のインストールとライセンスについて説明します。

Windows へのインストール

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

Windows でのライセンス

- ServiceController の開始
- LicenseServer の開始
- RaptorXML+XBRL Serverの開始
- RaptorXML+XBRL Serverの登録
- ライセンスの割り当て

2.1.1 Windows へのインストール

RaptorXML+XBRL 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*
    Windows Vista, Windows 7/8/10
  - *Windows Server*
    Windows Server 2008 R2 または以降

- RaptorXML+XBRL Server のインストール
  RaptorXML+XBRL 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+XBRL Server と称されます。RaptorXML+XBRL Server をインストールするには、RaptorXML+XBRL 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+XBRL 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+XBRL 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+XBRL Server と FlowForce Server のインストーラーは Altova Web サイトで入手でき、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+XBRL Server はデフォルトで以下で見つけることができます：

  ```
  <ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2017\bin\RaptorXMLXBRL.exe
  ```

  COM インターフェイスおよび .NET 環境を介して RaptorXML+XBRL 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+XBRL Server の COM サーバーオブジェクトとしての登録、(Java インターフェイス使用のための) `RaptorXMLLib.dll` の `WINDIR\system32\` ディレクトリ内でのインストール、`Altova.RaptorXML.dll` ファイルを to the .NET リファレンスライブラリへの追加などが含まれます。

- Altova LicenseServer
  - RaptorXML+XBRL 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+XBRL 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+XBRL 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+XBRL 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。

  RaptorXML+XBRL 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 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+XBRL 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+XBRL Server の特定のバージョンに適切なLicenseServer のバージョンが RaptorXML+XBRL 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+XBRL Server の全ての古いバージョンと作動します。
  - RaptorXML+XBRL 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: 2.3

- トライアル ライセンス

  インストール中、30 日間のRaptorXML+XBRL 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 XP	`C:\Program Files\Altova\`
Windows Vista、Windows 7/8	`C:\Program Files\Altova\`
64-bit OS 上の 32-bit バージョン	`C:\Program Files (x86)\Altova\`

2.1.2 Windows でのライセンス

RaptorXML+XBRL 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 2 つのステップから構成されています：

1. LicenseServer に**RaptorXML+XBRL Server** を**登録**します。RaptorXML+XBRL Server から登録することができます。
2. RaptorXML+XBRL Server へ**ライセンスを割り当て**ます。LicenseServer からライセンスを割り当てることができます。

必要な手順は以下に説明されています。

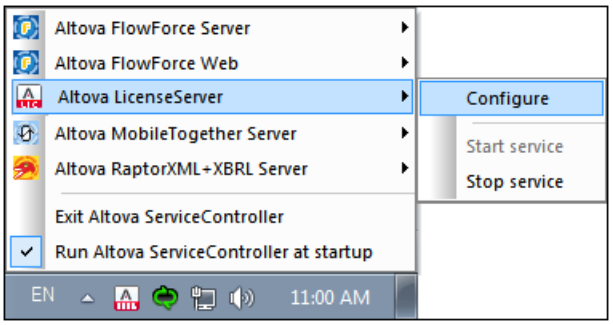
▾ ServiceController の開始

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

Altova ServiceController (略して ServiceController) は **Windows 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 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) と共にインストールされます。**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 *Altova サーバー製品がサービスとしてインストールされている* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server) **スタート** メニューフォルダーでも利用可能です。) 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*)。

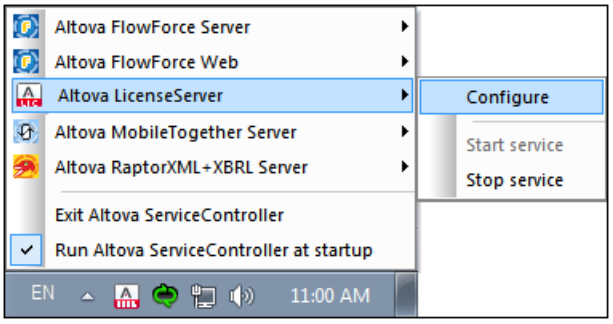
システムログイン時に ServiceController の自動開始を指定するには、システムトレイの **ServiceController** アイコンをクリックして **ServiceController** メニューを表示します (*下部スクリーンショット*)。 **スタートアップ時に Altova ServiceController を作動する (Run Altova ServiceController at Startup)** コマンドに切り替えます。 (このコマンドはデフォルトで切り替えられています。)ServiceController を終了するには、システムトレイの **ServiceController**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

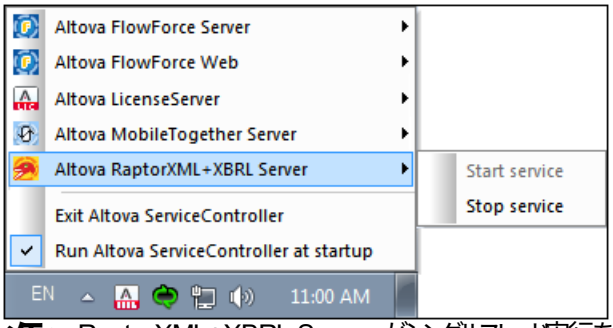
LicenseServer を開始するにはシステムトレイの、**[ServiceController]** アイコンをクリックします。メニューの

[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+XBRL Server の開始

  RaptorXML+XBRL 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+XBRL 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+XBRL 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+XBRL 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+XBRL 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+XBRL 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+XBRL 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+XBRL 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+XBRL Server の登録

  - *FlowForce Server を介しての RaptorXML+XBRL Server の登録*

    RaptorXML+XBRL 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+XBRL 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 Web 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 ; (ii) 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+XBRL Server へライセンスを割り当てます。

- *スタンドアロン RaptorXML+XBRL Server の登録*

  以下を介しての RaptorXML+XBRL Server の登録 :

  - `licenseserver` コマンドを利用した CLI です :

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

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

    ```
    RaptorXMLXBRL licenseserver localhost
    ```

  登録に成功した後、LicenseServer 構成ページの Server Management (サーバー管理)タブに移動して、RaptorXML+XBRL 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+XBRL Server の登録に成功した後、LicenseServer の構成ページの Server Management (サーバーの管理タブ) リストされます。移動して RaptorXML+XBRL Server にライセンスの割り当てを行います。

コアとライセンスについてのメモ

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。

MobileTogether Server ライセンス

MobileTogether Server ライセンスには 2 つの種類があります。カスタマーは必要に応じてライセンスの種類を選択することができます。

- *コア ライセンス* : サーバー マシンのコア数をベースにして MobileTogether Servers に割り当てられます。上の例を参照してください。上の説明を参照してください。コア ライセンスは、無制限の数量の MobileTogether クライアント デバイスによりサーバーへの接続を許可します。しかしながら、「単一スレッドの実行」チェックボックスがチェックされていると、1 度に MobileTogether Server に接続できるモバイル デバイスは 1 台です。これは、評価と小さい規模のテストを行う際に役に立ちます。この場合、2 台目のモバイル デバイスが MobileTogether Sever に接続される場合、ライセンスは 2 台目が使用するようになります。最初のデバイスは、接続できないようになり、エラー メッセージが表示されます。

- *デバイスライセンス：* MobileTogether Server にいつでも接続することのできる MobileTogether Client デバイスの最高使用数を指定します。

2.2 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+XBRL Server のインストールとライセンスについて説明します。

Linux へのインストール

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

Linux でのライセンス

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

環境についてのメモ

2.2.1 Linux へのインストール

RaptorXML+XBRL 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。

- システム必要条件
  - *Linux*
    - CentOS 6 または以降
    - RedHat 6 または以降
    - Debian 7 または以降
    - Ubuntu 12.04 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+XBRL Server	qt4, krb5-libs, qt-x11	libqtcore4, libqtgui4, libgssapi-krb5-2	libqtcore4, libqtgui4, libgssapi-krb5-2

- FlowForce Server への統合

  RaptorXML+XBRL 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+XBRL 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/RaptorXMLXBRLServer2017/etc/*.tool /opt/Altova/
FlowForceServer2017/tools
```

  このコマンドは**.tool** ファイルをRaptorXML+XBRL 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 +XBRL Server 実行可能ファイルへのパスを含みます。FlowForce Server をRaptorXML+XBRL 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。

- ルートユーザーのメモ

  RaptorXML+XBRL 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

  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+XBRL Server がインストールされていない場合、以下の*RaptorXML+XBRL Server のインス*

*ツール*で説明されている手順を踏んでください。

RaptorXML+XBRL Server が既にインストールされており、RaptorXML+XBRL 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 raptorxmlxbrlserver
[CentOS, RedHat]:   sudo rpm -e raptorxmlxbrlserver
```

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
```

Linux パッケージのダウンロード

以下のRaptorXML+XBRL 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。

ディストリビューション	パッケージ拡張子
Debian 6 と以降	.deb
Ubuntu12.04 と以降	.deb
CentOS 6 と以降	.rpm
RedHat 6 と以降	.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+XBRL Server を作動するためには Altova LicenseServer が必要なため、Altova Web サイトから RaptorXML+XBRL 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+XBRL 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+XBRL Server インストールする:

```
[Debian]:   sudo dpkg --install raptorxmlxbrlserver-2017-debian.deb
[Ubuntu]:   sudo dpkg --install raptorxmlxbrlserver-2017-ubuntu.deb
[CentOS]:   sudo rpm -ivh raptorxmlxbrlserver-2017-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh raptorxmlxbrlserver-2017-1.x86_64.rpm
```

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

```
/opt/Altova/RaptorXMLXBRLServer2017
```

Altova LicenseServer

RaptorXML+XBRL 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 Web 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+XBRL 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-2.3-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-2.3-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-2.3-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-2.3-1.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+XBRL 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 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+XBRL 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+XBRL Server の特定のバージョンに適切な LicenseServer のバージョンが RaptorXML+XBRL 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+XBRL Server の全ての古いバージョンと作動します。
- RaptorXML+XBRL 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: 2.3

トライアルライセンス

インストール中、RaptorXML+XBRL Server の 30 日間トライアルライセンスのオプションが与えられます。トライアルライセンスのリクエストが送信されると、登録された電子メールアドレスにライセンスが送付されます。

2.2.2 Linux でのライセンス

RaptorXML+XBRL 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+XBRL Server** にLicenseServer を**登録**します。RaptorXML+XBRL Server から登録することができます。
2. RaptorXML+XBRL Server の**ライセンスを割り当て**ます。LicenseServer からライセンスを割り当てることができます。

必要な手順は以下に説明されています。

▾root ユーザーのメモ

RaptorXML+XBRL 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+XBRL 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+XBRL Serverの開始

RaptorXML+XBRL 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/ raptorxmlxbrlserver start`
[≥ Debian 8]	`sudo systemctl start raptorxmlxbrlserver`

[< CentOS 7]	sudo initctl start raptorxmlxbrlserver
[≥ CentOS 7]	sudo systemctl start raptorxmlxbrlserver
[< Ubuntu 15]	sudo initctl start raptorxmlxbrlserver
[≥ Ubuntu 15]	sudo systemctl start raptorxmlxbrlserver
[RedHat]	sudo initctl start raptorxmlxbrlserver

▾ RaptorXML+XBRL Server の登録

以下を使用しての RaptorXML+XBRL Server の登録：

- `licenseserver` コマンドを使用した CLI：

```
sudo /opt/Altova/RaptorXMLXBRLServer2017/bin/raptorxmlxbrl
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/RaptorXMLXBRLServer2017/bin/raptorxmlxbrl
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+XBRL Server 実行可能の場所を確認してください：

```
/opt/Altova/RaptorXMLXBRLServer2017/bin/
```

登録に成功した後、LicenseServer 構成ページの Server Management (サーバー管理) タブに移動して、RaptorXML+XBRL 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+XBRL Server の登録に成功した後、LicenseServer の構成ページの Server Management (サーバーの管理タブ) リストされます。移動して RaptorXML+XBRL Server にライセンスの割り当て

コアとライセンスについてのメモ

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。

MobileTogether Server ライセンス
MobileTogether Server ライセンスには2つの種類があります。カスタマーは必要に応じてライセンスの種類を選択することができます。

- *コアライセンス*: サーバーマシンのコア数をベースにして MobileTogether Servers に割り当てられます。上の例を参照してください。上の説明を参照してください。コアライセンスは、無制限の数量の MobileTogether クライアントデバイスによりサーバーへの接続を許可します。しかしながら、「単一スレッドの実行」チェックボックスがチェックされていると、1度に MobileTogether Server に接続できるモバイルデバイスは1台です。これは、評価と小さい規模のテストを行う際に役に立ちます。この場合、2台目のモバイルデバイスが MobileTogether Sever に接続される場合、ライセンスは2台目が使用するようになります。最初のデバイスは、接続できないようになり、エラーメッセージが表示されます。
- *デバイスライセンス*: MobileTogether Server にいつでも接続することのできる MobileTogether Client デバイスの最高使用数を指定します。

2.3 Mac OS X でのセットアップ

このセクションは RaptorXML+XBRL Server の Mac OS X システムへの インストール と ライセンス について説明します。

Mac OS X へのインストール

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

Mac OS でのライセンス

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

環境についてのメモ

2.3.1 Mac OS X へのインストール

RaptorXML+XBRL Server の Mac OS X 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*
  - *Mac OS X*

    OS X 10.10、 10.11、macOS 10.12 またはそれ以降

- FlowForce Server 統合

  RaptorXML+XBRL 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+XBRL 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 /usr/local/Altova/RaptorXMLXBRLServer2017/etc/*.tool /usr/local/Altova/
  FlowForceServer2017/tools
  ```

  このコマンドは **.tool** ファイルを RaptorXML+XBRL 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+XBRL Server 実行可能へのパスを含みます。FlowForce Server を RaptorXML+XBRL 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 ユーザーのメモ

  RaptorXML+XBRL 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+XBRL 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.RaptorXMLXBRLServer2017.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+XBRL 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/RaptorXMLXBRLServer2017/
  ```

  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+XBRL Server の説明と同じ手順でアンインストールします。

- Mac OS X パッケージのダウンロード

  Altova Web サイトから MacOS X パッケージのダウンロード後、Mac OS X システムに直接コピーしてください。

RaptorXML+XBRL Server を作動するためには Altova LicenseServer が必要なため、Altova Web サイトから、RaptorXML+XBRL 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。Mac OS X インストーラーファイルは拡張子 `.pkg` を持ちます。

RaptorXML+XBRL Server のインストール

ターミナルウィンドウで、インストーラーファイルをコピーしたディレクトリに切り替え、ダブルクリックします。インストーラーウィザードの手順を踏みます。これらのステップは説明不要ですが、ステップの 1 つは使用許諾契約書に合意しなければなりません。

RaptorXML+XBRL Server は以下のフォルダーにインストールされます：
`/usr/local/Altova/RaptorXMLXBRLServer2017`

アプリケーションターミナルの RaptorXML+XBRL Server アイコンをクリックすると、画面ヘルプ(このドキュメンテーション)がポップアップします。

Altova LicenseServer

RaptorXML+XBRL 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。

Mac OS X 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。

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

LicenseServer versions

- Altova サーバー製品はインストールされた RaptorXML+XBRL 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+XBRL Server の特定のバージョンに適切な LicenseServer のバージョンが RaptorXML+XBRL 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+XBRL Server の全ての古いバージョンと作動します。
- RaptorXML+XBRL 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：23

- トライアルライセンス

  インストール中、RaptorXML+XBRL Serverの 30日間トライアルライセンスのオプションが与えられます。トライアルライセンスのリクエストが送信されると、登録された電子メールアドレスにライセンスが送付されます。

2.3.2 Mac OS X でのライセンス

RaptorXML+XBRL 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+XBRL Server** にLicenseServer を**登録**します。RaptorXML+XBRL Server から登録することができます。
2. RaptorXML+XBRL Server の**ライセンスを割り当て**ます。LicenseServer からライセンスを割り当てることができます。

必要な手順は以下に説明されています。

- root ユーザーのメモ

  RaptorXML+XBRL 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+XBRL Server をLicenseServer に正しく登録しライセンスを与えるためには、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+XBRL Serverの開始

  RaptorXML+XBRL 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 :

  ```
  sudo launchctl load /Library/LaunchDaemons/
  com.altova.RaptorXMLXBRLServer2017.plist
  ```

  RaptorXML+XBRL Server を停止する場合、以下を使用します :

  ```
  sudo launchctl unload /Library/LaunchDaemons/
  com.altova.RaptorXMLXBRLServer2017.plist
  ```

- RaptorXML+XBRL Serverの登録

  以下を介してのRaptorXML+XBRL Server の登録 :

  - `licenseserver` コマンドを使用したCLI :

    ```
    sudo /usr/local/Altova/RaptorXMLXBRLServer2017/bin/RaptorXMLXBRL
    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/RaptorXMLXBRLServer2017/bin/RaptorXMLXBRL
    licenseserver localhost
    ```

    上記のコマンドで、`localhost` がLicenseServer がインストールされたサーバーの名前です。RaptorXML

+XBRL Server 実行可能の場所を確認してください:
```
/usr/local/Altova/RaptorXMLXBRLServer2017/bin/
```

登録に成功した後、LicenseServer 構成ページの Server Management (サーバー管理)タブに移動して、RaptorXML+XBRL 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+XBRL Server の登録に成功した後、LicenseServer の構成ページの Server Management (サーバーの管理タブ) リストされます。移動して RaptorXML+XBRL Server にライセンスの割り当てを行います。

コアとライセンスについてのメモ
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 6CPU 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。

MobileTogether Server ライセンス
MobileTogether Server ライセンスには2つの種類があります。カスタマーは必要に応じてライセンスの種類を選択することができます。

- *コアライセンス*:サーバーマシンのコア数をベースにして MobileTogether Servers に割り当てられます。上の例を参照してください。上の説明を参照してください。コアライセンスは、無制限の数量の MobileTogether クライアントデバイスによりサーバーへの接続を許可します。しかしながら、「単一スレッドの実行」チェックボックスがチェックされていると、1 度に MobileTogether Server に接続できるモバイルデバイスは 1台です。これは、評価と小さい規模のテストを行う際に役に立ちます。この場合、2台目のモバイルデバイスが MobileTogether Sever に接続される場合、ライセンスは 2台目が使用するようになります。最初のデバイスは、接続できないようになり、エラーメッセージが表示されます。
- *デバイスライセンス*: MobileTogether Server にいつでも接続することのできる MobileTogether Client デバイスの最高使用数を指定します。

2.4 XML カタログ

XML カタログ メカニズムにより、ローカルフォルダーからファイルを取得することが可能になります。 カタログファイルの URI のみが変更されるため、処理スピード全体を向上し、ドキュメントのポータブル性を向上することができます。詳細に関しては、カタログの機能のセクションを参照してください。

AltovaXML 製品はカタログメカニズムを使用して、 DTD および XML スキーマなどの、一般に使用されるファイルに素早くアクセスしロードします。この カタログ メカニズムは、ユーザーによりカスタム化、および拡張することができます。AltovaXML カタログ メカニズムに詳細が説明されています。ファイル システム内の場所のための変数 のセクションは、共通システムロケーションのための Windows 変数 をリストしています。これらの変数は、よく使用されるフォルダーを検索するために カタログファイルで使用することができます。

このセクションは、以下のサブセクションに整理されています :

- カタログの機能
- Altova XML カタログ メカニズム
- ファイル システム内の場所のための変数

カタログに関する詳細は、XML カタログ仕様を参照してください。

2.4.1 カタログの機能

このセクション:

カタログは、リモートリソースをローカル URL にダイレクトする際に役に立ちます。これは、カタログファイル内、パブリックまたはシステム識別子、URI、または必要なローカル URL の識別子または URI の一部のマッピングにより達成されます。

パブリックおよびシステム識別子をローカル URL へマッピングする

XML ファイル内の DTD の `DOCTYPE` 宣言が読み込まれると、宣言のパブリックまたはシステム識別子が必要なリソースを検索します。識別子がリモートリソースを選択、または識別子がロケーターでない場合、ローカルリソースのカタログエントリを介して、マップすることが可能です。

例えば、次の SVG ファイルを考慮すると:

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

パブリック識別子は以下の通りです: `-//W3C//DTD SVG 1.1//EN`
システム識別子は以下の通りです: `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`

カタログエントリは、ローカル URL を検索するためにパブリック識別子を以下のようにマップすることができます:

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

または、カタログエントリは、ローカル URL を検索するためにシステム識別子を以下のようにマップすることができます:

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

カタログ内でパブリックまたはシステム識別子の一致がある場合、マップされている URL が使用されます。(相対パスは、カタログ要素のダイレクト内の `xml:base` 属性への参照と共に解決されます。フォールバックベースの URL はカタログファイルの URL です。) カタログ内でパブリックまたはシステム識別子の一致がない場合、XML ドキュメントの URL が使用されます (上のサンプルでは、`http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`)。

相対および絶対ファイルパス、Web URL、または名前のみをローカル URL へマッピングする

`uri` 要素を相対または絶対ファイルパスを、Web URL、または任意の名前、をローカル URL にマップする際に以下のように使用することができます:

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

name 値が発生した場合、uri 属性内の指定されたリソースにマップされます。カタログと共に、同じ名前が異なるリソースにマップされることもできます。例えば：

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

通常、属性の値の URI 部分 (上のサンプルで太字で表示) は、実際のスキーマロケーションのパスです。スキーマがカタログを介して参照されている場合、URI 部分は、実際の XML スキーマをポイントする必要はありません。しかし、xsi:schemaLocation 属性の構文の有効性を保持するために、存在する必要があります。foo の値は、(Orgchart.xsd の代わりに) 例えば、xsi:schemaLocation 属性の値の URI 部分に十分です。xsi:schemaLocation 属性の値 の名前空間部分によりスキーマは、カタログ内で検索されます。上のサンプルでは、名前空間の部分は以下の通りです：http://www.altova.com/schemas/orgchart。

カタログ内で、次のエントリは、スキーマを名前空間の部分をベースにして検索します。

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

詳細に関しては、XML カタログ仕様を参照してください。

2.4.2 Altova XML カタログ メカニズム

このセクション:

- ルートカタログファイル、Rootcatalog.xml、は、 RaptorXML が検索するカタログファイルを含みます。
- Altova カタログ拡張ファイル: Corecatalog.xml、Customcatalog.xml、および catalog.xml。
- サポートされるカタログサブセット

Rootcatalog.xml

デフォルトでは、RaptorXML は、使用するカタログファイルのリストのため Rootcatalog.xml (以下にリストされる) ファイルを検索します。Rootcatalog.xml は以下のフォルダーにあります:

```
<ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2017\etc
```

ルートカタログとして他のファイルを使用するには、コマンドライン上の --catalog オプション、Java インターフェイスの setcatalog メソッド、または COM インターフェイスの catalog メソッドを使用します。

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

  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/Customcatalog.xml"/>
  <nextCatalog catalog="Corecatalog.xml"/>

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

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

検索する追加カタログファイルは、それぞれ nextCatalog 要素内にリストされています。追加する数に制限はありません。それぞれのカタログファイルが検索され、その中のマッピングが解決されます。各カタログファイルは検索され、その中のマッピングは解決されます。

上のリストでは、2 つのカタログは直接以下を参照しています: Corecatalog.xml および Customcatalog.xml。更に、Schemas および XBRL フォルダーのサブフォルダーの最初のレベルにある catalog.xml という名前のカタログも参照されています。(%AltovaCommonFolder% 変数の値はファイル システム内の場所のための変数のセクションで与えられています。)

Altova Common フォルダー内の カタログファイルは、(XML スキーマおよび XHTML などの) よく使用されるスキーマの定義済みのパブリックおよびシステム識別子を、対応するスキーマのローカルコピーをポイントする URI にマップします。
RaptorXML がインストールされると、これらのスキーマは Altova Common フォルダーにインストールされます。

Corecatalog.xml、Customcatalog.xml、および catalog.xml

カタログファイル Corecatalog.xml と Customcatalog.xml は、Rootcatalog.xml にリストされています:

- Corecatalog.xml は、Altova Common フォルダー内のスキーマを検索するための Altova 固有のマッピングを含みます。
- Customcatalog.xml は、独自のマッピングを作成することのできるスケルトンファイルです。 Altova Common フォルダー内のカタログファイルによりアドレス指定されていない必要な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 メカニズムにサポートされる要素を使用して、上記を行ってください。 (*以下を参照*).
- catalog.xml ファイルが固有のスキーマのフォルダー内または Altova Common フォルダー内の XBRL タクソノミ内にあり、それぞれパブリックおよび/またはシステム識別子を、ローカルで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 にマップします。

Corecatalog.xml と Customcatalog.xml は、フォルダー *<ProgramFilesFolder>*\Altova\RaptorXMLXBRLServer2017\etc にあります。catalog.xml ファイルは、特定のスキーマフォルダーで、これらのスキーマフォルダーは %AltovaCommonFolder%\Schemas および %AltovaCommonFolder%\XBRL フォルダーの内部にあります。

サポートされるカタログサブセット

RaptorXML が使用するカタログファイル内のエントリを作成する場合、次の OASIS カタログ仕様の要素のみを使用します。下にリストされる個々の要素は、属性の値の説明を伴います。詳細については、 XML カタログの仕様を参照してください。

- <**public** publicId="PublicID of Resource" uri="URL of local file"/>
- <**system** systemId="System ID 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 System ID" rewritePrefix="Replacement string to locate resource locally"/>

パブリック識別子が存在しない場合、system 要素を介して、システム識別子が直接 URL にマップされます。また、uri 要素を使用して URI を他の URI にマップすることもできます。rewriteURI と rewriteSystem 要素は、URI の開始部分またはシステム識別子の書き換えを有効化することができます。これによりファイルパスの開始を置き換えて、結果的に他のディレクトリをターゲットにすることができます。

メモ: 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。xml:base 要素が存在しない場合は、カタログファイルの URI がベース URI として扱われます。

詳細に関しては、XML カタログの仕様を参照してください。

2.4.3 Windows システム内の場所のための変数

シェル環境変数は、各種 Windows システムロケーションのパスを指定するために、カタログファイルで使用することができます。以下の変数がサポートされます：

%AltovaCommonFolder%	C:\Program Files\Altova\Common2017
%DesktopFolder%	現在のユーザーのためのデスクトップフォルダーへのフルパス。
%ProgramMenuFolder%	現在のユーザーのためのプログラムメニューフォルダーへのフルパス。
%StartMenuFolder%	現在のユーザーのためのスタートメニューフォルダーへのフルパス。
%StartUpFolder%	現在のユーザーのためのスタートアップフォルダーへのフルパス
%TemplateFolder%	現在のユーザーのためのテンプレートフォルダーへのフルパス
%AdminToolsFolder%	現在のユーザーのため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%	現在のユーザーのためのアプリケーションデータフォルダへのフルパス。
%CommonAppDataFolder%	全てのユーザーのため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%	現在のユーザーのためのお気に入りフォルダへのフルパス。
%PersonalFolder%	現在のユーザーのための個人用フォルダへのフルパス。
%SendToFolder%	現在のユーザーのための送信済みフォルダへのフルパス。
%FontsFolder%	システムフォントフォルダへのフルパス。
%ProgramFilesFolder%	現在のユーザーのための プログラムファイルフォルダへのフルパス。
%CommonFilesFolder%	現在のユーザーのための共有ファイルへのフルパス
%WindowsFolder%	現在のユーザーのためのWindows フォルダへのフルパス。
%SystemFolder%	現在のユーザーのためのシステムフォルダへのフルパス。
%LocalAppDataFolder%	ローカルアプリケーション (ローミングしない) のためのデータレポジトリであるでシステムディレクトリへのフルパス。
%MyPicturesFolder%	マイピクチャフォルダへのフルパス。

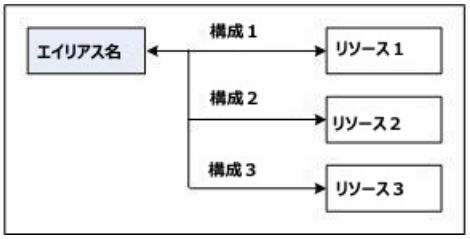
2.5 グローバルリソース

このセクション:

- グローバルリソースに関して
- グローバルリソースの使用方法

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:

```
raptorxmlxbrl 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:

```
raptorxmlxbrl 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：

```
raptorxmlxbrl 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ご注意してください。例：**altova://folder_resource/AliasName/input.xml**。

2.6 セキュリティの問題

このセクション:

RaptorXML+XBRL 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 インターフェイスに関連するセキュリティの問題

HTTP 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+XBRL 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` 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+XBRL 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の `server.script-root-dir` 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` のサブディレクトリである)ジョブ出力ディレクトリ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。

チャプター 3

コマンドライン インターフェイス

3 コマンドライン インターフェイス

コマンドライン インターフェイス (CLI) と共に使用されるRaptorXML+XBRL Server 実行ファイルは、デフォルトで以下にあります :

Windows	`<ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2017\bin\RaptorXMLXBRL.exe`
Linux	`/opt/Altova/RaptorXMLXBRLServer2017/bin/raptorxmlxbrl`
Mac	`/usr/local/Altova/RaptorXMLXBRLServer2017/bin/raptorxmlxbrl`

▼ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows および Mac のみでしか使用できません。
* 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てください。

使用方法

コマンドラインの構文は以下の通りです :

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

`RaptorXMLXBRL`	Windows プラットフォーム上でアプリケーションを呼び出す
`raptorxmlxbrl`	Unix プラットフォーム (Linux および Mac) 上でアプリケーションを呼び出す
`--h | --help`	ヘルプテキストの表示
`--version`	アプリケーションのバージョン番号の表示
`<command>`	実行するコマンド。下のリストを参照してください。このセクションのサブセクションで、各コマンドはオプションや引数と共に説明されています。
`[options]`	コマンドのオプション。対応するコマンドと共にリストされ、オプションのセクションで詳細と共に説明されています
`[arguments]`	コマンドの引数。対応するコマンドと共にリストおよび説明されています

CLI コマンド

使用可能なCLI コマンドは、下にリストされており、機能別に整理されています。このセクションのサブセクションで、詳細が説明されています。(検証コマンドの一部は、下のリスト内で 1 つ以上のグループで表示されることにご注意ください。)

すべての検証コマンド

valdtd | dtd	DTD ドキュメントを検証します。
valxsd | xsd	W3C XML スキーマドキュメントを検証します。
valxml-withdtd | xml	DTD に対して XML ドキュメントを検証します。
valxml-withxsd | xsi	XML スキーマに対して XML ドキュメントを検証します。
valxslt	XSLT ドキュメントを検証します。
valxquery	XQuery ドキュメントを検証します。
valxbrl | xbrl	XBRL インスタンスドキュメントを検証します。(.xbrl 拡張子)
valinlinexbrl | ixbrl	インライン XBRL (iXBRL) ドキュメントを検証します。
valxbrltaxonomy | dts	XBRL タクソノミスキーマ ドキュメントを検証します。(.xsd 拡張子)
valtaxonomypackage	XBRL タクソノミ パッケージを検証します。
valjson	JSON ドキュメントを検証します。
valjsonschema	JSON スキーマドキュメントをスキーマ仕様に対して検証します。
valavroschema	Avro バイナリファイルを Avro スキーマ仕様に対して検証します。
valavrojson	JSON データファイルを Avro スキーマに対して検証します。
valavro	Avro バイナリ内のデータを自身の Avro スキーマに対して検証します。
valany	前のコマンドにより検証される型のドキュメントを検証します。ドキュメントの型は自動的に検出されます。

整形式チェック コマンド

wfjson	JSON ドキュメントの整形式をチェックします。
wfxml	XML ドキュメントの整形式をチェックします。
wfdtd	DTD ドキュメントの整形式をチェックします。
wfany	XML または DTD ドキュメントの整形式をチェックします。

XBRL 検証コマンド

valxbrl | xbrl	XBRL インスタンスドキュメント (.xbrl 拡張子) を検証します。
valinlinexbrl | ixbrl	インライン XBRL (iXBRL) ドキュメントを検証します。
valxbrltaxonomy | dts	XBRL タクソノミ (スキーマ) ドキュメント (.xsd 拡張子) を検証します。
valtaxonomypackage	XBRL タクソノミ パッケージを検証します。

XSLT コマンド

xslt	引数により与えられたXSLT ファイルを使用して変換を行います。
valxslt	XSLT ドキュメントを検証します。

XQuery コマンド

xquery	引数により与えられたXQuery ファイルを使用してXQuery を行います。
valxquery	XQuery ドキュメントを検証します。

JSON/Avro コマンド

avroextractschema	Avro バイナリファイルからAvro スキーマを抽出します。
valavro	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	Avro バイナリファイルからデータを抽出し、データをJSON としてシリアル化します。
valavroschema	Avro スキーマ仕様に対してAvro スキーマを検証します。
valjsonschema	JSON スキーマドキュメントの有効性をチェックします。
valjson	JSON ドキュメントの有効性をチェックします。
wfjson	JSON ドキュメントの整形式をチェックします。

3.1 XML、DTD、XSD 検証コマンド

XML 検証コマンドは次の種類のドキュメントを検証するために使用することができます :

- *XML:* DTD に対して XML インスタンスドキュメント (`valxml-withdtd | xml`) または XML スキーマ 1.0/1.1 (`valxml-withxsd | xsi`) を検証します。
- *DTD:* DTD が整形式でエラーを含まないかをチェックします (`valdtd | dtd`)。
- *XSD:* XML スキーマ仕様 のルールに従い W3C XML スキーマ (XSD) ドキュメント (`valxsd | xsd`) を検証します。

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 | xml`	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 | xsi`	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 | dtd`	DTD ドキュメントを検証します。
`valxsd | xsd`	W3C XML スキーマ (XSD) ドキュメントを検証します。

メモ: XBRL インスタンス、XBRL タクソノミ、XSLT、XQuery、JSON、とAvro ドキュメントも検証することができます。これらの検証コマンドは対応する各セクションで説明されています :XBRL 検証コマンド; XSLT コマンド、XQuery コマンド、JSON/Avro コマンド。

3.1.1 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。

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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は（raptorxmlxbrl）全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、大文字と小文字（RaptorXMLXBRL）は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます：(i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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 \RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

  - **enable-globalresources**

    --enable-globalresources = true|false

グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、 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+XBRL 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 = *VALUE*
`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。

3.1.2 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W3C XML スキーマ定義言語 (XSD) 1.0 および 1.1。

Windows	**RaptorXMLXBRL valxml-withxsd | xsi [options] *InputFile***
Linux	**raptorxmlxbrl valxml-withxsd | xsi [options] *InputFile***
Mac	**raptorxmlxbrl 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=true|false は、XML ドキュメント内のXSD 参照が使用されるかどうかを示します。(ロケーションが使用され) デフォルトは true です。--xsd=FILE オプションは使用す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 (i) 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 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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。

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。* 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 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 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がある場合、--schema-mapping オプションの値が使用されます。(XBRL オプションおよびXML/XSD オプション) 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またはXBRL内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 のスキーマの場所のURL 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がある場合、--schema-mapping オプションの値が使用されます。(XBRL オプションおよびXML/XSD オプション) 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

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 **2.3** 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。

- **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
  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。

- **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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、 valany --h。(または、 help コマンドお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+XBRL 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 = *VALUE*
`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。

3.1.3 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 or XML 1.1 に従い検証します。

Windows **RaptorXMLXBRL valdtd | dtd [options]** ***InputFile***

Linux **raptorxmlxbrl valdtd | dtd [options]** ***InputFile***

Mac **raptorxmlxbrl 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 (i) 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 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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`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+XBRL 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 = *VALUE*
`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。

3.1.4 valxsd (xsd)

valxsd | xsd コマンドは、1つまたは複数のW3C XML スキーマ定義言語 (XSD) 1.0 or 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。

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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプション および XML/XSD オプション）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 または XBRL 内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所の URL 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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプション および XML/XSD オプション）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`
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 **`2.3`** 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。

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`
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。

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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`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+XBRL 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 = *VALUE*
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。

3.2 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	XML ドキュメントの整形式をチェックします。
wfdtd	DTD の整形式をチェックします。
wfany	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。

3.2.1 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。

Windows **RaptorXMLXBRL wfxml [options]** ***InputFile***

Linux **raptorxmlxbrl wfxml [options]** ***InputFile***

Mac **raptorxmlxbrl 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 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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2 つのグループに分けられています。値は、2 つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (ii) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、 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+XBRL 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 = *VALUE*
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。

3.2.2 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。

Windows **RaptorXMLXBRL wfdtd [options]** ***InputFile***

Linux **raptorxmlxbrl wfdtd [options]** ***InputFile***

Mac **raptorxmlxbrl 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 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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2 つのグループに分けられています。値は、2 つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (ii) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。`*` 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

`--enable-globalresources = true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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`
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、`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+XBRL 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 =` *`VALUE`*
`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。

3.2.3 wfany

wfany 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 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。

Windows	**RaptorXMLXBRL wfany [options]** ***InputFile***
Linux	**raptorxmlxbrl wfany [options]** ***InputFile***
Mac	**raptorxmlxbrl 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に注意してください。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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2 つのグループに分けられています。値は、2 つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 検証と処理

▾ **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。* 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

- **enable-globalresources**

  --enable-globalresources = `true|false`
  グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

- **globalresourcefile [gr]**

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

- 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*
    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、`valany --h`。(または、`help` コマンドお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+XBRL 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 = *VALUE*
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。

3.3 XBRL 検証コマンド

XBRL 検証コマンドは、XBRL インスタンスドキュメントおよびXBRL タクソノミをXBRL 2.1、Dimensions 1.0 およびFormula 1.0 仕様に従い検証するために使用することができます。使用可能な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：

valxbrl | xbrl	XBRL インスタンスドキュメントを検証します。(.xbrl 拡張子)。
valinlinexbrl (ixbrl)	1つまたは複数のインラインXBRL (iXBRL) ドキュメントをインラインXBRL 1.0 、または、インラインXBRL 1.1 仕様に従い検証します。
valxbrltaxonomy | dts	XBRL タクソノミ(スキーマ) ドキュメントを検証します。(.xsd 拡張子)。
valtaxonomypackage | taxpkg	1つまたは複数のXBRL タクソノミパッケージをタクソノミ 1.0 パッケージ仕様に従い検証します。

3.3.1 valxbrl (xbrl)

valxbrl | xbrl コマンドは、1つまたは複数のXBRL インスタンス ドキュメントをXBRL 2.1、Dimensions 1.0 およびFormula 1.0 仕様に従い検証します。

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

InputFile 引数は、検証するXBRL 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) 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 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 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: XBRL インスタンス ドキュメントは他のXML ドキュメントとネストされていてはならず、xbrl 要素をルート要素として持たなければなりません。
<xbrl xmlns="http://www.xbrl.org/2003/instance"> ... </xbrl>

EDGAR 検証

EDGAR (Electronic Data Gathering, Analysis, and Retrieval) は、企業により米国証券取引委員会 (Securities and Exchange Commission)(SEC) に提出された金融書類の自動化された収集、検証、および 分類をするシステムです。raptorxmlxbrl は、Raptor のPython API を介したEDGAR 検証をサポートします。EDGAR 検証をXBRL インスタンス ファイルで実行するには、--script オプションを使用してEDGAR 検証 Python スクリプトを実行します。raptorxmlxbrl 内では、このスクリプト**efm-validation.py** はアプリケーション フォルダー内のetc\scripts\sec-edgar-tools フォルダーにあります：

```
valxbrl --script="C:\Program Files\Altova\RaptorXMLXBRLServer2015\etc
\scripts\sec-edgar-tools\efm-validation.py" myinstance.xbrl
```

サンプル

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

▾ コマンドライン上の文字種とスラッシュ

*Windows 上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。

* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

- XBRL 検証と処理
  - **dimensions**

    --dimensions = true|false
    XBRL Dimension 1.0 拡張子を有効化します。デフォルトは 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。

  - **dts**

    --dts = *FILE*
    FILE 内のDTS エントリポイントをプリロードすることにより、ユーザーはインスタンスファイルの検証に使用することができます。オプションを複数回追加することにより、1つ以上のエントリポイントを指定することができます。コマンドは、同じタクソノミを参照する複数のインスタンスファイルをバッチ検証するために使用されます。 --dts オプションはエンジンに、ファイルリスト内の各インスタンスではなく、一回のみDTS をロードするように指定します。(ファイルリストは、CLI 上または --listfile オプションが true に設定されているとファイルリストとして指定されているテキストファイルに与えられます。) ファイルリスト内のインスタンスファイルが異なるタクソノミを参照する場合、警告が与えられます。このコマンドは、多数の大きなサポートDTS を持つ小さいインスタンスファイルを検証する速度を上げることができます。単一のインスタンスファイルを検証する際の利用では、利点はありません。

  - **extensible-enumerations**

    --extensible-enumerations = true|false
    true の場合 XBRL Extensible Enumerations 1.0 拡張子を有効化します。デフォルト:true。

  - **inconsistencies-limit**

    --inconsistencies-limit = *VALUE*
    値が 1-65535 の範囲である、XBRL 不整合性の制限を指定します。処理は、制限に到達しても継続されますが、更なる不整合性は報告されません。デフォルトの値 :100.

  - **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。

preload-xbrl-schemas

--preload-xbrl-schemas = true|false
XBRL 2.1 仕様のスキーマをプリロードします。デフォルトは 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。

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。* 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 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 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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプション および XML/XSD オプション）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 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。

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。

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 または XBRL 内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所のURL 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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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。

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

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 **2.3** 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。

taxonomy-package

--taxonomy-package = *FILE*

Taxonomy Package 1.0 および Taxonomy Packages 1.0 作業ドラフトで説明されているとおり、追加タクソノミパッケージへの絶対パスを指定します。FILE の値は、タクソノミパッケージの場所を与えます。1つ以上のタクソノミパッケージを指定するには、オプションを複数回追加してください。

taxonomy-packages-config-file

--taxonomy-packages-config-file = *FILE*
XBRL タクソノミパッケージをロードするために使用される **TaxonomyPackagesConfig.json** ファイルへのパスを指定します。ファイルの便利な点は、このファイルがタクソノミパッケージのカタログとして使用できることです。JSON ファイルの構造は、したのサンプルのリスティングのとおりです。 **uri** キーの値は、パッケージの位置を示します。(パッケージのセット) **active** キーは、パッケージの使用法を切り替えます。

```
{"taxonomies":
  {"EIOPA Solvency II XBRL Taxonomy 2.1.0":
    {"packages":[
      {"uri":"C:\\test\\XBRL\
\EIOPA_SolvencyII_XBRL_Taxonomy_2.1.0.zip"},
      {"uri":"C:\\test\\XBRL\\AdditionalTestPkg.zip"}
      ],"active":true
    },"Test Taxonomy":
      {"packages":[{"uri":"C:\\test\\XBRL\\test.zip"}],"active":true
      }
  }
}
```

▼ **treat-inconsistencies-as-errors**

--treat-inconsistencies-as-errors = true|false
XBRL 2.1 仕様の定義に対する不整合がファイルに含まれる場合、XBRL 検証の失敗を引き起こします。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。

▼ **utr**

--utr = true|false
true の場合、XBRL Unit Registry 1.0 拡張子を有効化します。デフォルト:false。

▼ **validate-dts-only**

--validate-dts-only = true|false
DTS は、XBRL インスタンスドキュメントから検出されます。参照されたすべてのタクソノミスキーマとリンクベースは検出され、検証されます。残りの XBRL インスタンスドキュメントは無視されます。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。

▼ **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。

▼ XBRL フォーミュラとアサーション

▼ **assertion-severity**

--assertion-severity = true|false
Assertion Severity 1.0 の拡張子を有効化します。デフォルト:true.
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。

▼ **assertions-output**

--assertions-output = *FILE*
指定された*FILE* に対してのアサーション評価の出力を書き込みます。設定されると、自動的に--formula-execution=true を指定します。

assertions-output-format

--assertions-output-format = json|xml
アサーション評価の出力フォーマットを指定します。デフォルトはjson です。

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = true|false
false の場合、フォーミュラアサーションテーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト:true。

formula

--formula = true|false
XBRL Formula 1.0 拡張子を有効化します。デフォルトは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。

formula-assertion-set [[DEPRECATED]]

--formula-assertion-set = *VALUE*
フォーミュラの実行を与えられたアサーションのセットに制限します。1つ以上のアサーションセットを指定するため、オプションを複数回追加します。短い形式は--as です。 ***VALUE*** は、@id 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。特別な値##none と##all も使用することができます。

formula-execution

--formula-execution = true|false
XBRL フォーミュラの評価を有効化します。デフォルトはtrue です。 true の場合自動的に--formula=true を指定します。
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。

formula-output

--formula-output = *FILE*
指定された*FILE* に対するフォーミュラ評価の出力を書き込みます。設定されると、自動的に--formula-execution=true を指定します。

formula-parameters

--formula-parameters = *JSON-ARRAY*
XBRL フォーミュラ評価のパラメーターをJSON マップの配列として直接 CLI 上に指定します。 詳細に関してはフォーミュラパラメーターのセクションを参照してください。

formula-parameters-file

--formula-parameters-file = *FILE*
フォーミュラ評価のパラメーターを含む*FILE* を指定します。ファイルはXML ファイルまたはJSON ファイルであることができます。フォーミュラパラメーターのセクションを参照してください。

ignore-assertion

--ignore-assertion = *VALUE*
実行から与えられたアサーションを除外します。 1つ以上のアサーションを指定するためには、このオプションを複数回追加してください。

ignore-assertions-file

--ignore-assertions-file = *FILE*
処理から除外されるアサーションのID/XPointer のリストを含むファイルへのパスを指定します。ファイル内では、一行につき1つのアサーションを入力してください。

ignore-formula

--ignore-formula = *VALUE*
実行から与えられたフォーミュラを除外します。 1つ以上のフォーミュラを指定するためには、このオプションを複数回追加してください。

ignore-formulas-file

--ignore-formulas-file = *FILE*
処理から除外されるフォーミュラID/XPointer のリストを含むファイルへのパスを指定します。ファイル内では、一行につき1つのフォーミュラを入力してください。

message-lang

--message-lang = *VALUE*
検証メッセージを表示する際に使用される言語を指定します。デフォルトの言語 :`en`. 使用することのできる他の言語 `de`, `es`, `ja`, それぞれ、ドイツ語、スペイン語、日本語を指します。

message-role

--message-role = *VALUE*
検証メッセージを表示するために使用される優先メッセージを指定します。デフォルト :`http://www.xbrl.org/2010/role/message`.

preload-formula-schemas

--preload-formula-schemas = `true|false`
XBRL Formula 1.0 仕様のスキーマをプリロードします。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。

process-assertion [a]

--a | --process-assertion = *VALUE*
フォーミュラの実行を与えられたアサーションに制限します。 1つ以上のアサーションを指定するため、オプションを複数回追加します。短い形式は `--a` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

process-assertion-set [as]

--as | --process-assertion-set = *VALUE*
フォーミュラの実行を与えられたアサーションのセットに制限します。1つ以上のアサーションセットを指定するため、オプションを複数回追加します。 短い形式は `--as` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

process-assertions-file

--process-assertions-file = *FILE*
実行するアサーションのID/XPointer のリストを含むファイルへのパスを指定します。ファイル内では、一行につき1つのアサーションを入力してください。

- **process-formula [f]**

  --f | --process-formula = *VALUE*
  フォーミュラの実行を与えられたフォーミュラに制限します。1つ以上のフォーミュラを指定するため、オプションを複数回追加します。短い形式は `--f` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

- **process-formulas-file**

  --process-formulas-file = *FILE*
  実行するフォーミュラID/XPointer のリストを含むファイルへのパスを指定します。ファイル内では、一行につき1つのフォーミュラを入力してください。

- **report-unsatisfied-assertion-evaluations**

  --report-unsatisfied-assertion-evaluations = `true|false`
  アサーション重要度レベルに従い、条件を満たさないアサーションの評価をエラーまたは警告としてレポートします。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。

- **variableset-execution-timeout**

  --variableset-execution-timeout = *VALUE*
  (**--formula-execution=true**) フォーミュラを実行する際に適用されます。単一の変数のセットの実行に許可される最長時間を指定します (フォーミュラ 値、存在、整合性アサーション)。時間は分数で指定されており、正の数である必要があります。 デフォルトは 30分です。特定の変数セットが実行をタイムアウト前に完了しない場合は中断されます。エラーメッセージが表示されます (詳細ログに入力されます)。しかし、タイムアウトのチェックは各変数セットの評価の後で実行され、各 XPath 式の実行中には実行されないことに注意してください。ですから、単一のXPath 式の実行に長い時間が要される場合、タイムアウトの制限を過ぎる可能性があります。変数セットの実行は、完全な変数セットの評価が実行されてから、中断されます。

- XBRL テーブル

  - **concept-label-linkrole**

    --concept-label-linkrole = *VALUE*
    コンセプトラベルをレンダする際、優先する拡張リンクロールを指定します。

  - **concept-label-role**

    --cconcept-label-role = *VALUE*
    コンセプトラベルをレンダする際、優先するラベルロールを指定します。デフォルト:`http://www.xbrl.org/2003/role/label`。

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

    --evaluate-referenced-parameters-only = `true|false`
    `false` の場合、フォーミュラ/アサーション/テーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト:`true`。

  - **generic-label-linkrole**

    --generic-label-linkrole = *VALUE*
    ジェネリックラベルをレンダする際、使用される優先する拡張されたリンクロールを指定します。

  - **generic-label-role**

    --generic-label-role = *VALUE*
    ジェネリックラベルをレンダする際、優先するラベルロールを指定します。デフォルト:http://www.xbrl.org/2003/

role/label。

label-lang

--label-lang = *VALUE*
ラベルをレンダする際使用する優先ラベル言語を指定します。デフォルト:en。

preload-table-schemas

--preload-table-schemas = true|false
XBRL Table 1.0 仕様のスキーマをプリロードします。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。

process-table [t]

--t | --process-table = *VALUE*
フォーミュラの実行を与えられたテーブルに制限します。1つ以上のテーブルを指定するため、オプションを複数回追加します。短い形式は --t です。***VALUE*** は、@id 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。##none や ##all などの特別な値も使用することができます。

table

--table = true|false
XBRL Table 1.0 拡張子を有効化します。デフォルト値はtrue です。true の場合自動的に--formula=true および--dimensions=true を指定します。
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。

table-elimination

--table-elimination = true|false
HTML 出力内の空のテーブル行 列の削除を有効化します。デフォルトは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。

table-execution

--table-execution = true|false
XBRL テーブルの評価を有効化します。デフォルトはfalse です。--table-output が指定されている場合、true に設定されます。true の場合自動的に--table=true を指定します。
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。

table-linkbase-namespace

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

前のドラフト仕様を使用して作成されたテーブルリンクベースのロードを有効化します。テーブルリンクベース検証、解像度、レイアウトはしかしながら、常に 2014年 3月 18日版 Table Linkbase 1.0 勧告に従い実行されます。##detect を使用して自動検出を有効化します。

table-output

--table-output = *FILE*
指定された *FILE* にテーブル出力を書き込みます。設定されると、自動的に--table-execution=true を指定します。

table-output-format

--table-output-format = xml|html
テーブル出力のフォーマットを指定します。デフォルトは xml 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`valany --h`。(または、`help` コマンドはお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+XBRL 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 = *VALUE*
`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。

3.3.2 valinlinexbrl (ixbrl)

valinlinexbrl | ixbrl コマンドは、1つまたは複数のインラインXBRL (iXBRL) ドキュメントをXBRL 1.0 またはインラインXBRL 1.1 仕様に従い検証します。XBRL ファイルは、インラインXBRL から生成され、保存することができます。

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

InputFile 引数は、検証されるインラインXBRL ドキュメント (通常、XHTML ドキュメント) を指定します。検証の最初のステップでは、コマンドの実行は、XHTML をXBRL にインラインXBRL を抽出することにより、変換します。最初のステップでエラーが生成されず、`--validate-xbrl` オプションが `true` に設定されている場合、生成されたXBRL は、インラインXBRL により指定されたXBRL タクソノミに対して検証されます。

複数のドキュメントを検証するには、以下の1つを行います : (i) CLI 上で検証されるファイルをリストします。個々のファイルは、スペースにより区切られています。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` 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*)。複数のインラインXBRL 入力ファイルが処理された場合でも、1つのXBRL ドキュメントのみが生成されることに注意してください。

サンプル

- **raptorxmlxbrl** `valinlinexbrl c:\MyIXBRL.xhtml`
- **raptorxmlxbrl** `valinlinexbrl --ixbrl-version=1.1 c:\MyIXBRL.xhtml`
- **raptorxmlxbrl** `ixbrl --validate-xbrl=true --xbrl-output=C:\MyOutXBRL.xbrl C:\MyIXBRL.xhtml`
- **raptorxmlxbrl** `ixbrl --validate-xbrl=false --xbrl-output=C:\MyOutXBRL.xbrl C:\MyIXBRL.xhtml`
- **raptorxmlxbrl** `ixbrl --xbrl-output=C:\MyOutXBRL.xbrl --document-set=true --listfile=true C:\MyFileList.txt`

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows およびMac のみでしか使用できません。
* 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*を使用してください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ XBRL 検証と処理

▾ **dimensions**

--dimensions = true|false
XBRL Dimension 1.0 拡張子を有効化します。デフォルトは 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。

dts

--dts = *FILE*
FILE 内のDTS エントリポイントをプリロードすることにより、ユーザーはインスタンスファイルの検証に使用することができます。オプションを複数回追加することにより、1つ以上のエントリポイントを指定することができます。コマンドは、同じタクソノミを参照する複数のインスタンスファイルをバッチ検証するために使用されます。 --dts オプションはエンジンに、ファイルリスト内の各インスタンスではなく、一回のみDTS をロードするように指定します。 (ファイルリストは、CLI 上または --listfile オプションが true に設定されているとファイルリストとして指定されているテキストファイルに与えられます。) ファイルリスト内のインスタンスファイルが異なるタクソノミを参照する場合、警告が与えられます。このコマンドは、多数の大きなサポートDTS を持つ小さいインスタンスファイルを検証する速度を上げることができます。単一のインスタンスファイルを検証する際の利用では、利点はありません。

extensible-enumerations

--extensible-enumerations = true|false
true の場合 XBRL Extensible Enumerations 1.0 拡張子を有効化します。デフォルト:true。

inconsistencies-limit

--inconsistencies-limit = *VALUE*
値が 1-65535 の範囲である、XBRL 不整合性の制限を指定します。処理は、制限に到達しても継続されますが、更なる不整合性は報告されません。デフォルトの値 :100.

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。

preload-xbrl-schemas

--preload-xbrl-schemas = true|false
XBRL 2.1 仕様のスキーマをプリロードします。デフォルトは 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

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。* 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 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 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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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 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。

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。

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 またはXBRL内の `xsi:schemaLocation` および `xsi:noNamespaceSchemaLocation` 属性 の スキーマの場所のURL 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, がある場合、`--schema-mapping` オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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。

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`

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 **`2.3`** 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。

taxonomy-package

`--taxonomy-package =` *FILE*

Taxonomy Package 1.0 および Taxonomy Packages 1.0 作業ドラフトで説明されているとおり、追加タクソノミパッケージへの絶対パスを指定します。FILE の値は、タクソノミパッケージの場所を与えます。1つ以上のタクソノミパッケージを指定するには、オプションを複数回追加してください。

taxonomy-packages-config-file

`--taxonomy-packages-config-file =` *FILE*

XBRL タクソノミパッケージをロードするために使用される **`TaxonomyPackagesConfig.json`** ファイルへのパスを指定します。ファイルの便利な点は、このファイルがタクソノミパッケージのカタログとして使用できることです。JSON ファイルの構造は、したのサンプルのリスティングのとおりです。 **`uri`** キーの値は、パッケージの位置を示します。(パッケージのセット) **`active`** キーは、パッケージの使用法を切り替えます。

```
{"taxonomies":
  {"EIOPA Solvency II XBRL Taxonomy 2.1.0":
    {"packages":[
      {"uri":"C:\\test\\XBRL\
\EIOPA_SolvencyII_XBRL_Taxonomy_2.1.0.zip"},
     {"uri":"C:\\test\\XBRL\\AdditionalTestPkg.zip"}
```

```
        ],"active":true
      },"Test Taxonomy":
        {"packages":[{"uri":"C:\\test\\XBRL\\test.zip"}],"active":true
        }
    }
}
```

▼ **treat-inconsistencies-as-errors**

--treat-inconsistencies-as-errors = true|false
XBRL 2.1 仕様の定義に対する不整合がファイルに含まれる場合、XBRL 検証の失敗を引き起こします。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。

▼ **utr**

--utr = true|false
true の場合、XBRL Unit Registry 1.0 拡張子を有効化します。デフォルト:false。

▼ **validate-dts-only**

--validate-dts-only = true|false
DTS は、XBRL インスタンスドキュメントから検出されます。参照されたすべてのタクソノミスキーマとリンクベースは検出され、検証されます。残りの XBRL インスタンスドキュメントは無視されます。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。

▼ **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。

▼ インライン XBRL

▼ **document-set**

--document-set = true|false
true に設定されている場合、送信される全てのファイル (インライン XBRL ドキュメント) は、インライン XBRL ドキュメントセットとして扱われます。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。

▼ **ixbrl-version**

--ixbrl-version = 1.0|1.1|detect
検証のために使用されるインライン XBRL 仕様のバージョンを指定します。デフォルトの値は、detect です。

▼ **transformation-registry**

```
--transformation-registry =
  #all |
  http://www.xbrl.org/2008/inlineXBRL/transformation |
  http://www.xbrl.org/inlineXBRL/transformation/2010-04-20 |
  http://www.xbrl.org/inlineXBRL/transformation/2011-07-31 |
```

```
http://www.xbrl.org/inlineXBRL/transformation/2015-02-26
```

インラインXBRL 内で、日付と数値型は、多種の構文フォーマットで表示されますが、XBRL 内では、これらの型は特殊なフォーマットを有しています。例えば、インラインXBRL 内の日付は、`01 January 2017`, というフォーマットを与えられるかもしれませんが、XBRL 内での日付のフォーマットは`2017-01-01` です。インライン XBRL 変換レジストリは、日付と数値の値がインラインXBRL からXBRL に変換された時の変換ルールを指定します。`transformation-registry` オプションは、指定することのできるレジストリを制限します。オプションを複数回追加して、1つ以上の変換レジストリを指定します。デフォルトの値は、使用することのできる全ての変換レジストリを選択する`#all` です。 (*RaptorXML+XBRL Server のバージョン内で使用することのできる変換レジストリのリストに関しては、CLI 上のコマンドの説明を参照してください*)。

uri-transformation

--uri-transformation = `none|make-absolute|make-relative|keep-relative`
URI がどのように生成されたXBRL ドキュメントに書き込まれるかを指定します。

- `none`: URI verbatim をターゲットドキュメントにコピーする。
- `make-absolute`: 入力ドキュメント内の対応する要素でのインスコープベースURI に対して解決し、相対的なURI を絶対的にする。例 :入力ファイルが`c:\test\inlinexbrl.xhtml` の場合、そして、スキーマ`schemas\schema.xsd` に対して相対的なレファレンスを含む場合、相対的なレファレンスは次に対して解決されます :`c:\test\schemas\schema.xsd`。 入力ドキュメント内の`xml:base` 属性は、ベースURI を変更することができることに注意してください。
- `make-relative`: 可能であれば、絶対的および相対的を出力ドキュメントに対して相対的なURI にします (それ以外の場合、解決された絶対 URI を書きます)。
- `keep-relative`: 可能であれば、相対的なURI のみを出力ドキュメントに対して相対的にします (そして絶対 URI をコピーします)。

validate-xbrl

--validate-xbrl = `true|false`
`true` に設定されている場合、インラインXBRL ドキュメントにより生成されたXBRL ドキュメントの検証を有効化します。false に設定されている場合は、XBRL を生成する最初のステップの後に検証は中断されます。デフォルトの値は、`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。

xbrl-output

--xbrl-output = *FILE*
生成されたXBRL 出力をオプションで指定されたファイルの場所に書き込みます。複数のインラインXBRL 入力ファイルが処理された場合でも、XBRL ドキュメント1つのみが生成されます。

XBRL フォーミュラとアサーション

assertion-severity

--assertion-severity = `true|false`
Assertion Severity 1.0 の拡張子を有効化します。デフォルト:`true`.
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。

assertions-output

--assertions-output = *FILE*
指定された*FILE* に対してのアサーション評価の出力を書き込みます。設定されると、自動的に`--formula-execution=true` を指定します。

assertions-output-format

--assertions-output-format = json|xml
アサーション評価の出力フォーマットを指定します。デフォルトはjson です。

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = true|false
false の場合、フォーミュラアサーションテーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト:true。

formula

--formula = true|false
XBRL Formula 1.0 拡張子を有効化します。デフォルトは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。

formula-assertion-set [[DEPRECATED]]

--formula-assertion-set = *VALUE*
フォーミュラの実行を与えられたアサーションのセットに制限します。1つ以上のアサーションセットを指定するため、オプションを複数回追加します。短い形式は--as です。 ***VALUE*** は、@id 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。特別な値 ##none と##all も使用することができます。

formula-execution

--formula-execution = true|false
XBRL フォーミュラの評価を有効化します。デフォルトはtrue です。 true の場合自動的に--formula=true を指定します。
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。

formula-output

--formula-output = *FILE*
指定された*FILE* に対するフォーミュラ評価の出力を書き込みます。設定されると、自動的に--formula-execution=true を指定します。

formula-parameters

--formula-parameters = *JSON-ARRAY*
XBRL フォーミュラ評価のパラメーターをJSON マップの配列として直接 CLI 上に指定します。 詳細に関してはフォーミュラパラメーターのセクションを参照してください。

formula-parameters-file

--formula-parameters-file = *FILE*
フォーミュラ評価のパラメーターを含む*FILE* を指定します。ファイルはXML ファイルまたはJSON ファイルであることができます。フォーミュラパラメーターのセクションを参照してください。

ignore-assertion

--ignore-assertion = *VALUE*
実行から与えられたアサーションを除外します。 1つ以上のアサーションを指定するためには、このオプションを複数回追加してください。

ignore-assertions-file

--ignore-assertions-file = *FILE*
処理から除外されるアサーションのID/XPointer のリストを含むファイルへのパスを指定します。ファイル内では、

一行につき1つのアサーションを入力してください。

ignore-formula

--ignore-formula = *VALUE*
実行から与えられたフォーミュラを除外します。 1つ以上のフォーミュラを指定するためには、このオプションを複数回追加してください。

ignore-formulas-file

--ignore-formulas-file = *FILE*
処理から除外されるフォーミュラID/XPointer のリストを含むファイルへのパスを指定します。 ファイル内では、一行につき1つのフォーミュラを入力してください。

message-lang

--message-lang = *VALUE*
検証メッセージを表示する際に使用される言語を指定します。デフォルトの言語 :`en`. 使用することのできる他の言語 `de`, `es`, `ja`, それぞれ、ドイツ語、スペイン語、日本語を指します。

message-role

--message-role = *VALUE*
検証メッセージを表示するために使用される優先メッセージを指定します。デフォルト :`http://www.xbrl.org/2010/role/message`.

preload-formula-schemas

--preload-formula-schemas = `true|false`
XBRL Formula 1.0 仕様のスキーマをプリロードします。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。

process-assertion [a]

--a | --process-assertion = *VALUE*
フォーミュラの実行を与えられたアサーションに制限します。 1つ以上のアサーションを指定するため、オプションを複数回追加します。短い形式は `--a` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

process-assertion-set [as]

--as | --process-assertion-set = *VALUE*
フォーミュラの実行を与えられたアサーションのセットに制限します。1つ以上のアサーションセットを指定するため、オプションを複数回追加します。 短い形式は `--as` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

process-assertions-file

--process-assertions-file = *FILE*
実行するアサーションのID/XPointer のリストを含むファイルへのパスを指定します。 ファイル内では、一行につき1つのアサーションを入力してください。

process-formula [f]

--f | --process-formula = *VALUE*
フォーミュラの実行を与えられたフォーミュラに制限します。1つ以上のフォーミュラを指定するため、オプションを複数回追加します。短い形式は `--f` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグ

メント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

- **process-formulas-file**

  --process-formulas-file = *FILE*
  実行するフォーミュラID/XPointer のリストを含むファイルへのパスを指定します。ファイル内では、一行につき1つのフォーミュラを入力してください。

- **report-unsatisfied-assertion-evaluations**

  --report-unsatisfied-assertion-evaluations = `true|false`
  アサーション重要度レベルに従い、条件を満たさないアサーションの評価をエラーまたは警告としてレポートします。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。

- **variableset-execution-timeout**

  --variableset-execution-timeout = *VALUE*
  (**--formula-execution=true**) フォーミュラを実行する際に適用されます。単一の変数のセットの実行に許可される最長時間を指定します (フォーミュラ 値、存在、整合性アサーション)。時間は分数で指定されており、正の数である必要があります。 デフォルトは 30分です。特定の変数セットが実行をタイムアウト前に完了しない場合は中断されます。エラーメッセージが表示されます (詳細ログに入力されます)。しかし、タイムアウトのチェックは各変数セットの評価の後で実行され、各 XPath 式の実行中には実行されないことに注意してください。ですから、単一のXPath 式の実行に長い時間が要される場合、タイムアウトの制限を過ぎる可能性があります。変数セットの実行は、完全な変数セットの評価が実行されてから、中断されます。

- XBRL テーブル

  - **concept-label-linkrole**

    --concept-label-linkrole = *VALUE*
    コンセプトラベルをレンダする際、優先する拡張リンクロールを指定します。

  - **concept-label-role**

    --cconcept-label-role = *VALUE*
    コンセプトラベルをレンダする際、優先するラベルロールを指定します。デフォルト:`http://www.xbrl.org/2003/role/label`。

  - **generic-label-linkrole**

    --generic-label-linkrole = *VALUE*
    ジェネリックラベルをレンダする際、使用される優先する拡張されたリンクロールを指定します。

  - **generic-label-role**

    --generic-label-role = *VALUE*
    ジェネリックラベルをレンダする際、優先するラベルロールを指定します。デフォルト:http://www.xbrl.org/2003/role/label。

  - **label-lang**

    --label-lang = *VALUE*
    ラベルをレンダする際使用する優先ラベル言語を指定します。デフォルト:`en`。

  - **preload-table-schemas**

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

XBRL Table 1.0 仕様のスキーマをプリロードします。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。

process-table [t]

--t | --process-table = *VALUE*
フォーミュラの実行を与えられたテーブルに制限します。1つ以上のテーブルを指定するため、オプションを複数回追加します。短い形式は --t です。***VALUE*** は、@id 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。##none や ##all などの特別な値も使用することができます。

table

--table = true|false
XBRL Table 1.0 拡張子を有効化します。デフォルト値は true です。true の場合自動的に --formula=true および --dimensions=true を指定します。
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。

table-elimination

--table-elimination = true|false
HTML 出力内の空のテーブル行 列の削除を有効化します。デフォルトは 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。

table-execution

--table-execution = true|false
XBRL テーブルの評価を有効化します。デフォルトは false です。--table-output が指定されている場合、true に設定されます。true の場合自動的に --table=true を指定します。
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。

table-linkbase-namespace

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

前のドラフト仕様を使用して作成されたテーブルリンクベースのロードを有効化します。テーブルリンクベース検証、解像度、レイアウトはしかしながら、常に 2014年 3月 18日版 Table Linkbase 1.0 勧告に従い実行されます。##detect を使用して自動検出を有効化します。

table-output

--table-output = *FILE*
指定された *FILE* にテーブル出力を書き込みます。設定されると、自動的に --table-execution=true を指定します。

table-output-format

--table-output-format = xml|html
テーブル出力のフォーマットを指定します。デフォルトは xml 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`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+XBRL 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 = *VALUE*
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。

3.3.3 valxbrltaxonomy (dts)

valxbrltaxonomy | dts コマンドは、1つまたは複数のXBRL タクソノミ (スキーマ)をXBRL 2.1、Dimensions 1.0 およびFormula 1.0 仕様に従い検証します。

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

InputFile 引数が検証するXBRL タクソノミです。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 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 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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows 上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ XBRL 検証と処理

▾ **assertion-severity**

--assertion-severity = true|false
Assertion Severity 1.0 の拡張子を有効化します。デフォルト : true.
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。

▾ **dimensions**

--dimensions = true|false
XBRL Dimension 1.0 拡張子を有効化します。デフォルトは 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。

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = true|false
false の場合、フォーミュラアサーションテーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト：true。

extensible-enumerations

--extensible-enumerations = true|false
true の場合 XBRL Extensible Enumerations 1.0 拡張子を有効化します。デフォルト：true。

inconsistencies-limit

--inconsistencies-limit = *VALUE*
値が 1-65535 の範囲である、XBRL 不整合性の制限を指定します。処理は、制限に到達しても継続されますが、更なる不整合性は報告されません。デフォルトの値：100.

formula

--formula = true|false
XBRL Formula 1.0 拡張子を有効化します。デフォルトは 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。

formula-parameters

--formula-parameters = *JSON-ARRAY*
XBRL フォーミュラ評価のパラメーターを JSON マップの配列として直接 CLI 上に指定します。詳細に関してはフォーミュラパラメーターのセクションを参照してください。

formula-parameters-file

--formula-parameters-file = *FILE*
フォーミュラ評価のパラメーターを含む *FILE* を指定します。ファイルは XML ファイルまたは JSON ファイルであることができます。フォーミュラパラメーターのセクションを参照してください。

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。

preload-formula-schemas

--preload-formula-schemas = true|false
XBRL Formula 1.0 仕様のスキーマをプリロードします。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。

preload-xbrl-schemas

--preload-xbrl-schemas = true|false
XBRL 2.1 仕様のスキーマをプリロードします。デフォルトは 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。

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。* 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 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 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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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 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。

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。

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 または XBRL 内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所のURL 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がある場合、--schema-mapping オプションの値が使用されます。(XBRL オプションおよび XML/XSD オプション) 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。

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
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 **2.3** 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。

treat-inconsistencies-as-errors

--treat-inconsistencies-as-errors = true|false
XBRL 2.1 仕様の定義に対する不整合がファイルに含まれる場合、XBRL 検証の失敗を引き起こします。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。

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。

XBRL テーブル

concept-label-linkrole

--concept-label-linkrole = *VALUE*
コンセプトラベルをレンダする際、優先する拡張リンクロールを指定します。

concept-label-role

--cconcept-label-role = *VALUE*
コンセプトラベルをレンダする際、優先するラベルロールを指定します。デフォルト:`http://www.xbrl.org/2003/role/label`。

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = `true|false`
`false` の場合、フォーミュラアサーション/テーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト:`true`。

generic-label-linkrole

--generic-label-linkrole = *VALUE*
ジェネリックラベルをレンダする際、使用される優先する拡張されたリンクロールを指定します。

generic-label-role

--generic-label-role = *VALUE*
ジェネリックラベルをレンダする際、優先するラベルロールを指定します。デフォルト:http://www.xbrl.org/2003/role/label。

label-lang

--label-lang = *VALUE*
ラベルをレンダする際使用する優先ラベル言語を指定します。デフォルト:`en`。

preload-table-schemas

--preload-table-schemas = `true|false`
XBRL Table 1.0 仕様のスキーマをプリロードします。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。

process-table [t]

--t | --process-table = *VALUE*
フォーミュラの実行を与えられたテーブルに制限します。1つ以上のテーブルを指定するため、オプションを複数回追加します。短い形式は `--t` です。***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。##none や ##all などの特別な値も使用することができます。

table

--table = `true|false`
XBRL Table 1.0 拡張子を有効化します。デフォルト値は `true` です。`true` の場合自動的に `--formula=true` および `--dimensions=true` を指定します。
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。

table-execution

--table-execution = `true|false`
XBRL テーブルの評価を有効化します。デフォルトは `false` です。`--table-output` が指定されている場合、`true` に設定されます。`true` の場合自動的に `--table=true` を指定します。
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。

table-linkbase-namespace

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

前のドラフト仕様を使用して作成されたテーブルリンクベースのロードを有効化します。テーブルリンクベース検証、解像度、レイアウトはしかしながら、常に 2014年 3月 18日版 Table Linkbase 1.0 勧告に従い実行されます。`##detect` を使用して自動検出を有効化します。

table-output

--table-output = *FILE*
指定された *FILE* にテーブル出力を書き込みます。設定されると、自動的に--table-execution=true を指定します。

table-output-format

--table-output-format = `xml|html`
テーブル出力のフォーマットを指定します。デフォルトは `xml` 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、 `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+XBRL 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 = *VALUE*
`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。

3.3.4 valtaxonomypackage (taxpkg)

valtaxonomypackage | taxpkg コマンドは、1つまたは複数のXBRL タクソノミパッケージをタクソノミ1.0パッケージ仕様に従い検証します。

Windows	**RaptorXMLXBRL valtaxonomypackage | taxpkg [options]** ***TaxonomyPackage***
Linux	**raptorxmlxbrl valtaxonomypackage | taxpkg [options]** ***TaxonomyPackage***
Mac	**raptorxmlxbrl valtaxonomypackage | taxpkg [options]** ***TaxonomyPackage***

TaxonomyPackage 引数は、検証されるタクソノミパッケージです。タクソノミパッケージは通常 ZIP ファイルです。複数のタクソノミパッケージを検証するには、以下を行います : (i) CLI 上で検証されるパッケージをリストします。または、 (ii) テキストファイル (.txt ファイル)内で検証されるパッケージをリストし、テキストファイルを *TaxonomyPackage* 引数として、 --listfile オプションセットを true に設定して提供します (*下のオプションリストを参照してください*)。

サンプル

- **raptorxmlxbrl** valxtaxonomypackage c:\Test.zip
- **raptorxmlxbrl** taxpkg --listfile=true c:\TaxonomyPackageList.txt
- **raptorxmlxbrl** taxpkg --listfile c:\TaxonomyPackageList.txt

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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 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 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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション およびXML/XSD オプション) 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 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。

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。

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 またはXBRL内の

xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所のURL 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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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。

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

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 **2.3** 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false

グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

- **globalresourcefile [gr]**

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

- 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*
    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、`valany --h`。(または、`help` コマンドはお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+XBRL 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 = *VALUE*

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。

3.4 XSLT コマンド

XSLT コマンド:

- xslt: 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: XSLT ドキュメントの検証のためのコマンド

各コマンドの引数とオプションはサブセクション xslt および valxslt にリストされています。

3.4.1 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として指定されます。

Windows	**RaptorXMLXBRL xslt [options] *XSLT-File***
Linux	**raptorxmlxbrl xslt [options] *XSLT-File***
Mac	**raptorxmlxbrl 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`) 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`) が必要とされます。`--output` 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 以下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

- **raptorxmlxbrl** xslt --input=c:\Test.xml --output=c:\Output.xml c:\Test.xslt
- **raptorxmlxbrl** xslt --template-entry-point=StartTemplate --output=c:\Output.xml c:\Test.xslt
- **raptorxmlxbrl** xslt --input=c:\Test.xml --output=c:\Output.xml --param=date://node[1]/@att1 --p=title:'stringwithoutspace' --param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2 つのグループに分けられています。値は、2 つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。

▾ **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 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 式内またはお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：*

```
raptorxmlxbrl xslt --input=c:\Test.xml --output=c:\Output.xml --
param=date://node[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

streaming

--streaming = true|false

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。

template-entry-point

--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。

template-mode

--template-mode = *VALUE*

テンプレートモードが変換に使用されるように指定します。

xslt-version

--xslt-version = 1|1.0|2|2.0|3|3.0
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 スキーマおよび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:false。

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 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 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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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 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 またはXBRL内の xsi:schemaLocation およびxsi:noNamespaceSchemaLocation 属性 のスキーマの場所のURL 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, がある場合、--schema-mapping オプ

ションの値が使用されます。(XBRL オプション および XML/XSD オプション) 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
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。

▼ **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-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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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 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*
    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、 valany --h。(または、 help コマンドお引数と共に使用することができます。例 :help valany。)

  - **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+XBRL 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。

3.4.2 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。

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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ XSLT 処理

▾ **template-entry-point**

--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。

▾ **template-mode**

--template-mode = *VALUE*
テンプレートモードが変換に使用されるように指定します。

▾ **xslt-version**

--xslt-version = 1|1.0|2|2.0|3|3.0
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：false。

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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプションおよびXML/XSD オプション）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またはXBRL内の xsi:schemaLocation およびxsi:noNamespaceSchemaLocation 属性 のスキーマの場所のURL 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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプションおよびXML/XSD オプション）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
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。

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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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 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*
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、`valany --h`。(または、`help` コマンドはお引数と共に使用することができます。例 :`help valany`。)

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+XBRL 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。

3.5 XQuery コマンド

XQuery コマンド:

- xquery: 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: 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: XQuery ドキュメントの検証のため
- valxqueryupdate: XQuery (アップデート) ドキュメントの検証のため

各コマンドの引数とオプションはサブセクション xquery および valxquery でリストされています。

3.5.1 xquery

xquery コマンドはXQuery ファイルを単一引数として取り、この引数を入力任意の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として指定されます。

Windows **RaptorXMLXBRL xquery [options]** ***XQuery-File***

Linux **raptorxmlxbrl xquery [options]** ***XQuery-File***

Mac **raptorxmlxbrl 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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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：*

```
raptorxmlxbrl xslt --input=c:\Test.xml --output=c:\Output.xml --
```

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

xquery-version

--xquery-version = 1|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:false。

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
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。

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-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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

▼ **enable-globalresources**

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

▼ **globalresourcefile [gr]**

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

▼ 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 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*
    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、`valany --h`。(または、`help` コマンドはお引数と共に使用することができます。例 :`help valany`。)

  - **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+XBRL 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。

3.5.2 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。

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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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：*

```
raptorxmlxbrl xslt --input=c:\Test.xml --output=c:\Output.xml --
param=date://node[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

xquery-update-version

--xquery-update-version = 1|3
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 スキーマおよび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:false。

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
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。

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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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 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*
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、`valany --h`。(または、`help` コマンドはお引数と共に使用することができます。例 :`help valany`。)

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+XBRL 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。

3.5.3 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。

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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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：false。

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
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。

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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

--gr | --globalresourcefile = *FILE*

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

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*
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、`valany --h`。(または、`help` コマンドお引数と共に使用することができます。例 :`help valany`。)

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+XBRL 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。

3.5.4 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。

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

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

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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|3
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：false。

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
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。

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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

▼ 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*
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、`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`。)

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+XBRL 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。

3.6 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	Avro バイナリファイルからAvro スキーマを抽出します。
valavro	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	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	Avro スキーマ仕様に対して、Avro スキーマを検証します。
valjsonschema	JSON スキーマドキュメントの有効性をチェックします。
valjson	JSON ドキュメントの有効性をチェックします。
wfjson	JSON ドキュメントの整形式をチェックします。

3.6.1 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。

Windows **RaptorXMLXBRL avroextractschema [options] --output=*AvroSchemaFile* *AvroBinaryFile***

Linux **raptorxmlxbrl avroextractschema [options] --output=*AvroSchemaFile* *AvroBinaryFile***

Mac **raptorxmlxbrl avroextractschema [options] --output=*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

- **raptorxmlxbrl** avroextractschema --output=c:\MyAvroSchema.avsc c:\MyAvroBinary.avro

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2 つのグループに分けられています。値は、2 つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`valany --h`。(または、`help` コマンドお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+XBRL 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 = *VALUE*
`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。

3.6.2 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。

Windows	**RaptorXMLXBRL avrotojson [options] --output=*JSONFile* *AvroBinaryFile***
Linux	**raptorxmlxbrl avrotojson [options] --output=*JSONFile* *AvroBinaryFile***
Mac	**raptorxmlxbrl 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

- **raptorxmlxbrl** avrotojson --output=c:\MyAvroData.json c:\MyAvroBinary.avro

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2 つのグループに分けられています。値は、2 つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。`*` 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`valany --h`。(または、`help` コマンドお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+XBRL 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 = *VALUE*
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。

3.6.3 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。

Windows	**RaptorXMLXBRL jsontoavro [options] --output=*AvroBinary* --schema=*AvroSchema* *JSONFiles***
Linux	**raptorxmlxbrl jsontoavro [options] --output=*AvroBinary* --schema=*AvroSchema* *JSONFiles***
Mac	**raptorxmlxbrl 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：() 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。(i) 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` 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

- **raptorxmlxbrl** `jsontoavro --output=c:\MyAvroBinary.avro --schema=c:\MyAvroSchema.avsc c:\MyAvroData.json`
- **raptorxmlxbrl** `jsontoavro --output=c:\MyAvroBinary.avro --schema=c:\MyAvroSchema.avsc c:\MyAvroData-01.json  c:\MyAvroData-02.json`
- **raptorxmlxbrl** `jsontoavro --output=c:\MyAvroBinary.avro --schema=c:\MyAvroSchema.avsc --listfile=true c:\MyAvroData.txt`

▾ コマンドライン上の文字種とスラッシュ

Windows上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は（`raptorxmlxbrl`）全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、大文字と小文字（`RaptorXMLXBRL`）は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、 2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。`*` 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

  - **enable-globalresources**

    --enable-globalresources = `true|false`
    グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

  - **globalresourcefile [gr]**

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

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*
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、 `valany --h`。(または、 `help` コマンドお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+XBRL 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 = *VALUE*
`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。

3.6.4 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。

Windows	**RaptorXMLXBRL valavro	avro [options] *AvroBinaryFile***
Linux	**raptorxmlxbrl valavro	avro [options] *AvroBinaryFile***
Mac	**raptorxmlxbrl 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 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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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est.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。* 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`valany --h`。(または、`help` コマンドはお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+XBRL 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 = *VALUE*
`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。

3.6.5 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。

Windows	**RaptorXMLXBRL valavrojson | avrojson [options] --schema=*AvroSchema JSONFile***
Linux	**raptorxmlxbrl valavrojson | avrojson [options] --schema=*AvroSchema JSONFile***
Mac	**raptorxmlxbrl valavrojson | avrojson [options] --schema=*AvroSchema JSONFile***

JSONFile 引数は、検証するJSON ドキュメントを指定します。**--schema** 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。(i) I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 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

- **raptorxmlxbrl** valavrojson --schema=c:\MyAvroSchema.avsc c:\MyJSONDataFile.json
- **raptorxmlxbrl** avrojson --schema=c:\MyAvroSchema.avsc c:\MyJSONDataFile.json

▾ コマンドライン上の文字種とスラッシュ

*Windows上での***RaptorXMLXBRL**
*Unix (Linux、Mac) 上での***raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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 \RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = true|false
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`valany --h`。(または、`help` コマンドはお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+XBRL 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 = *VALUE*
`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。

3.6.6 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。

Windows	**RaptorXMLXBRL valavroschema | avroschema [options] *AvroSchema***
Linux	**raptorxmlxbrl valavroschema | avroschema [options] *AvroSchema***
Mac	**raptorxmlxbrl 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。(i) 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 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

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は（raptorxmlxbrl）全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、大文字と小文字（RaptorXMLXBRL）は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます：(i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。* 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 \RaptorXMLXBRLServer2017\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 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 カタログを参照してください。
  - **enable-globalresources**
    --enable-globalresources = true|false
    グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。
  - **globalresourcefile [gr]**
    --gr | --globalresourcefile = *FILE*
    グローバルリソース ファイル を指定します。(グローバルリソースを有効化します)。

- 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*
    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、`valany --h`。(または、`help` コマンドお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+XBRL 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 = *VALUE*
`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。

3.6.7 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 スキーマドラフト 4仕様に従い検証します。

Windows	**RaptorXMLXBRL valjsonschema | jsonschema [options]** ***InputFile***
Linux	**raptorxmlxbrl valjsonschema | jsonschema [options]** ***InputFile***
Mac	**raptorxmlxbrl 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 (i) 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 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

- **raptorxmlxbrl** valjsonschema c:\MyJSONSchema.json
- **raptorxmlxbrl** jsonschema c:\MyJSONSchema-01.json c:\MyJSONSchema-02.json
- **raptorxmlxbrl** jsonschema --listfile=true c:\FileList.txt

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。`*` 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、`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+XBRL 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 = *VALUE*
`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。

3.6.8 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。

Windows	**RaptorXMLXBRL valjson | json [options] *InputFile***
Linux	**raptorxmlxbrl valjson | json [options] *InputFile***
Mac	**raptorxmlxbrl valjson | json [options] *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 (i) 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` 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

- **raptorxmlxbrl** valjson --schema=c:\MyJSONSchema.json c:\MyJSONInstance.json
- **raptorxmlxbrl** json --schema=c:\MyJSONSchema.json c:\MyJSONInstance-01.json c:\MyJSONInstance-02.json
- **raptorxmlxbrl** json --schema=c:\MyJSONSchema.json --listfile=true c:\FileList.txt

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows および Mac のみでしか使用できません。
* 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*を使用してください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。

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。`*` 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。

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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

`--enable-globalresources =` `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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*
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、 `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+XBRL 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 = *VALUE*
`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。

3.6.9 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。

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

- **raptorxmlxbrl** wfjson c:\MyJSONFile.json
- **raptorxmlxbrl** wfjson c:\MyJSONFile-01.json c:\MyJSONFile-02.json
- **raptorxmlxbrl** wfjson --listfile=true c:\FileList.txt

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

コマンドのオプションは以下にリストされ、2つのグループに分けられています。値は、2つのケースを除いて、引用なしで指定することができます: (i) 値文字列がスペースを含む場合、または (i) オプションの詳細で明確に引用が必要と指定されている場合。

▾ 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。

▾ **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。`*` 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

- **enable-globalresources**

  `--enable-globalresources =` `true|false`
  グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

- **globalresourcefile [gr]**

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

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*
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、`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+XBRL 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 = *VALUE*
`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。

3.7 Valany コマンド

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、対応するライブラリ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。

Windows **RaptorXMLXBRL valany [options]** ***InputFile***

Linux **raptorxmlxbrl valany [options]** ***InputFile***

Mac **raptorxmlxbrl 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)
- valxsd (xsd)
- valxml-withdtd (xml)
- valxml-withxsd (xsi)
- valxslt
- valxquery
- valxbrl (xbrl)
- valinlinexbrl (ixbrl)
- valxbrltaxonomy (dts)
- valavrojson (avrojson)

サンプル

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

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。

3.8 スクリプトコマンド

script コマンドは、RaptorXML Python API を使用するPython 3 スクリプトを実行します。

Windows	**RaptorXMLXBRL script [options] *File***
Linux	**raptorxmlxbrl script [options] *File***
Mac	**raptorxmlxbrl script [options] *File***

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:

Windows	**RaptorXMLXBRL script [-h | --help]**
Linux	**raptorxmlxbrl script [-h | --help]**
Mac	**raptorxmlxbrl script [-h | --help]**

サンプル

- **raptorxmlxbrl** script c:\MyPythonScript.py
- **raptorxmlxbrl** script -h
- **raptorxmlxbrl** 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。*
- **raptorxmlxbrl** 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文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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*を使用してください。

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/using/cmdline.html を参照してください。

3.9 ヘルプとライセンスコマンド

このセクションはRaptorXML+XBRL Server の 2つの重要な機能について説明しています：

- ヘルプコマンド: 使用することのできるコマンドについての情報、またはコマンドの引数とオプションについて表示します。
- ライセンス: RaptorXML へのライセンスの与え方について説明します。

3.9.1 help

help コマンドは単一引数を取ります：ヘルプのためのコマンドの名前が必要とされます。コマンドの構文およびコマンドの正しい実行に関連した情報を表示します。

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

メモ： 引数が与えられない場合、help コマンドを実行すると、使用可能なすべてのコマンドが短い説明と共に表示されます。

サンプル
ヘルプコマンドのサンプル：

```
raptorxmlxbrl help valany
```

上のコマンドは 1 つの引数を含みます：コマンド valany にはヘルプが必要です。このコマンドが実行されると、valany コマンドのヘルプ情報が表示されます。

- コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

--help オプション
--help オプションを使用することにより、コマンドのヘルプ情報を確認することができます。例えば、valany コマンド付きの --help オプションを次のように使用することができます：

```
raptorxmlxbrl valany --help
```

により、valany の引数を持つ help コマンドを使用することと同じ結果を得ることができます：

```
raptorxmlxbrl help valany
```

両方の場合とも、valany コマンドのヘルプ情報が表示されます。

- コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows および Mac のみでしか使用できません。
* 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てください。

3.9.2 licenseserver

`licenseserver` コマンドは、RaptorXML+XBRL Server をAltova LicenseServer に登録します。このコマンドは、引数として、名前またはLicenseServer を作動中のサーバーのIP アドレスを取ります。

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

LicenseServer へのRaptorXML+XBRL Server の登録に成功すると、LicenseServer Web インターフェイスの URL が返されます。ブラウザーウィンドウにURL を入力して、Web インターフェイスにアクセスし、LicenseServer ドキュメントで説明されているとおり、ライセンスを与えるプロセスを行ってください。

サンプル

`licenseserver` コマンドのサンプル:

```
raptorxmlxbrl licenseserver DOC.altova.com
```

コマンドは、`DOC.altova.com` という名のマシンは、Altova LicenseServer を作動しているマシンと指定します。

- コマンドライン上の文字種とスラッシュ

  Windows 上での **RaptorXMLXBRL**
  Unix (Linux、Mac) 上での **raptorxmlxbrl**

  * 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows およびMac のみでしか使用できません。
  * 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てください。

オプション

以下のオプションを使用することが可能です:

--j|json=true|false
登録の試みの結果を、マシンにより解析することのできるJSON オブジェクトとして印刷します。

--h|help
コマンドのヘルプテキストを表示します。

--version
RaptorXML+XBRL Server のバージョン番号を表示します。オプションはコマンドの前に置かれる必要があります。ですから、以下のようになります:`raptorxmlxbrl --version licenseserver`。

3.9.3 assignlicense

`assignlicense` コマンドは、Windows にのみ使用することができます。このコマンドは、Altova LicenseServer が登録されている、RaptorXML+XBRL Server にライセンスファイルをアップロードし、RaptorXML+XBRL Server にライセンスを割り当てます (`licenseserver` コマンドを参照)。ライセンスファイルの URL を取ります。このコマンドを使用することにより、ライセンスの有効性をテストすることもできます。

Windows	**RaptorXMLXBRL assignlicense [options]** ***LICENSE-FILE***
Linux	***not applicable***
Mac	***not applicable***

サンプル

- **raptorxmlxbrl** `assignlicense C:\licensepool\mylicensekey.lic`
- **raptorxmlxbrl** `assignlicense --test-only=true C:\licensepool\mylicensekey.lic`

▼ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (`raptorxmlxbrl`) 全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、大文字と小文字 (`RaptorXMLXBRL`) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

以下のオプションを使用することが可能です :

--t|test-only=true|false
値が `true` の場合、LicenseServer にライセンスがロードされ、検証されますが、割り当ては行われません。

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。

--h|help
コマンドのヘルプテキストを表示します。

--version
RaptorXML+XBRL Server のバージョン番号を表示します。オプションはコマンドの前に置かれる必要があります。ですから、以下のようになります : `raptorxmlxbrl --version licenseserver`.

3.9.4 verifylicense

verifylicense コマンドは、Windows にのみ使用することができます。このコマンドは、RaptorXML+XBRL Server がライセンスされているかどうかをチェックし、オプションで与えられたライセンスキーがすでに RaptorXML+XBRL Server に割り当てられているかを確認します。このコマンドはライセンスキーをオプションとして取ります。このコマンドはライセンスの状態または与えられたライセンスキーの有効性に関するステートメントを返します。

Windows	**RaptorXMLXBRL verifylicense [options]**
Linux	***not applicable***
Mac	***not applicable***

サンプル

- **raptorxmlxbrl** verifylicense
- **raptorxmlxbrl** verifylicense --license-key=a-39-character-long-license-code-(7x5, plus 4 hyphens)
- **raptorxmlxbrl** verifylicense --l=a-39-character-long-license-code-(7x5, plus 4 hyphens)

返されるステートメントは、以下に類似します：

- The product has a valid license
- The product does not have a valid license
- The license key AAAAAAA-BBBBBBB-CCCCCCC-DDDDDDD-EEEEEEE is assigned to the product
- The license key AAAAAAA-BBBBBBB-CCCCCCC-DDDDDDD-EEEEEEE is not assigned to the product

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

オプション

以下のオプションを使用することが可能です：

--l|license-key=*VALUE*
VALUE は、39 文字から構成されるライセンスキーで、区切りはありません。キーはハイフンで区切られた 5 つのブロックから構成されます。

--h|help
コマンドのヘルプテキストを表示します。

--version

RaptorXML+XBRL Server のバージョン番号を表示します。オプションはコマンドの前に置かれる必要があります。ですから、以下のようになります：`raptorxmlxbrl --version licenseserver`.

3.10 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 4 つの言語 (英語、ドイツ語、スペイン語、および日本語) は、*<ProgramFilesFolder>*\Altova\RaptorXMLXBRLServer2017\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 コマンドを使用して行います。生成されたファイル内のリソース文字列は、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)、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リ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\RaptorXMLXBRLServer2017\bin\ フォルダー内に保存します。DLL ファイルは、RaptorXMLXBRLServer_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、in RaptorXMLXBRLServer_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 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 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+XBRL 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、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 4 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 コマンドを使用してください。

3.10.1 exportresourcestrings

exportresourcestrings コマンドは、RaptorXML リソース文字列を含むXML ファイルを出力します。このコマンドは以下の 2 つの引数を取ります : (i) 出力 XML ファイル内のリソース文字列の言語 および、(i) 出力 XML ファイルのパスと名前。許可されるエクスポート言語は、以下のとおりです (各言語の言語コードがかっこ内に記載されています) :英語 (en)、ドイツ語 (de)、スペイン語 (es)、および日本語 (ja)。

Windows **RaptorXMLXBRL exportresourcestrings *LanguageCode XMLOutputFile***

Linux **raptorxmlxbrl exportresourcestrings *LanguageCode XMLOutputFile***

Mac **raptorxmlxbrl exportresourcestrings *LanguageCode XMLOutputFile***

引数

exportresourcestrings コマンドは次の引数を必要とします :

LanguageCode	エクスポートされたXML ファイル内のソース文字列の言語である、エクスポートのターゲットの言語を指定します。サポートされる言語は以下の通りです : en、de、es、ja
XMLOutputFile	エクスポートされたXML ファイルの場所と名前を指定します。

サンプル

このコマンドは、Strings.xml というファイルを、ドイツ語に翻訳されたRaptorXML アプリケーションの全てのリソース文字列を含む c:\ に作成します。

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

▾ コマンドライン上の文字種とスラッシュ

Windows 上での **RaptorXMLXBRL**
Unix (Linux、Mac) 上での **raptorxmlxbrl**

* 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows および Mac のみでしか使用できません。
* 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てください。

3.10.2 setdeflang

setdeflang コマンド(略して sdl) は、RaptorXML のデフォルトの言語を設定します。このコマンドは、必須 LanguageCode 引数を取ります。

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

サンプル
このコマンドはアプリケーションのデフォルトの言語をドイツ語に設定します。

```
raptorxmlxbrl setdeflang de
```

- コマンドライン上の文字種とスラッシュ

  Windows上での **RaptorXMLXBRL**
  Unix (Linux、Mac) 上での **raptorxmlxbrl**

  * 小文字は (raptorxmlxbrl) 全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、大文字と小文字 (RaptorXMLXBRL) は、Windows およびMac のみでしか使用できません。
  * 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てください。

サポートされる言語
下のテーブルは、現在サポートされている言語を言語コードと共にリストしています。

en	英語
de	ドイツ語
es	スペイン語
ja	日本語

3.11 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
- メッセージ、エラー、ヘルプ
- 処理
- XBRL
- XML
- XSD
- XQuery
- XSLT
- JSON/Avro

3.11.1 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\RaptorXMLXBRLServer2017\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 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 カタログを参照してください。

enable-globalresources

--enable-globalresources = `true|false`
グローバルリソースを無効化します。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 を指定します (グローバルリソースを有効化します)。

globalresourcefile [gr]

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

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。`*` 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。

3.11.2 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*
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、 valany --h。(または、 help コマンドはお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+XBRL 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 = *VALUE*
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。

3.11.3 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
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 **2.3** 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。

3.11.4 XBRL

- XBRL 検証と処理オプション

  - **dimensions**

    --dimensions = true|false
    XBRL Dimension 1.0 拡張子を有効化します。デフォルトは 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。

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

    --evaluate-referenced-parameters-only = true|false
    false の場合、フォーミュラのアサーションテーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト：true。

  - **extensible-enumerations**

    --extensible-enumerations = true|false
    true の場合 XBRL Extensible Enumerations 1.0 拡張子を有効化します。デフォルト：true。

  - **inconsistencies-limit**

    --inconsistencies-limit = *VALUE*
    値が 1-65535 の範囲である、XBRL 不整合性の制限を指定します。処理は、制限に到達しても継続されますが、更なる不整合性は報告されません。デフォルトの値：100.

  - **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。

  - **preload-xbrl-schemas**

    --preload-xbrl-schemas = true|false
    XBRL 2.1 仕様のスキーマをプリロードします。デフォルトは 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。

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。* 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 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 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を介して使用されます。スキーマでは**デフォルトの値**です。
- load-by-namespace: カタログマッピングを介して、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, がある場合、--schema-mapping オプションの値が使用されます。 (XBRL オプション および XML/XSD オプション) 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 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。

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。

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 または XBRL 内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所のURL 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がある場合、--schema-mapping オプションの値が使用されます。(XBRL オプションおよび XML/XSD オプション) 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。

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

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 **2.3** 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。

treat-inconsistencies-as-errors

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

XBRL 2.1 仕様の定義に対する不整合がファイルに含まれる場合、XBRL 検証の失敗を引き起こします。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。

utr

--utr = true|false

true の場合、XBRL Unit Registry 1.0 拡張子を有効化します。デフォルト:false。

validate-dts-only

--validate-dts-only = true|false

DTS は、XBRL インスタンスドキュメントから検出されます。参照されたすべてのタクソノミスキーマとリンクベースは検出され、検証されます。残りの XBRL インスタンスドキュメントは無視されます。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。

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

す。

XBRL フォーミュラとアサーションオプション

assertion-severity

--assertion-severity = true|false
Assertion Severity 1.0 の拡張子を有効化します。デフォルト：true.
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。

assertions-output

--assertions-output = *FILE*
指定された*FILE* に対してのアサーション評価の出力を書き込みます。設定されると、自動的に--formula-execution=true を指定します。

assertions-output-format

--assertions-output-format = json|xml
アサーション評価の出力フォーマットを指定します。デフォルトはjson です。

formula

--formula = true|false
XBRL Formula 1.0 拡張子を有効化します。デフォルトは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。

formula-assertion-set [[DEPRECATED]]

--formula-assertion-set = *VALUE*
フォーミュラの実行を与えられたアサーションのセットに制限します。1つ以上のアサーションセットを指定するため、オプションを複数回追加します。短い形式は--as です。 ***VALUE*** は、@id 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。特別な値 ##none と##all も使用することができます。

formula-execution

--formula-execution = true|false
XBRL フォーミュラの評価を有効化します。デフォルトはtrue です。 true の場合自動的に--formula=true を指定します。
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。

formula-output

--formula-output = *FILE*
指定された*FILE* に対するフォーミュラ評価の出力を書き込みます。設定されると、自動的に--formula-execution=true を指定します。

formula-parameters

--formula-parameters = *JSON-ARRAY*
XBRL フォーミュラ評価のパラメーターをJSON マップの配列として直接 CLI 上に指定します。 詳細に関してはフォーミュラパラメーターのセクションを参照してください。

formula-parameters-file

--formula-parameters-file = *FILE*
フォーミュラ評価のパラメーターを含む *FILE* を指定します。ファイルはXML ファイルまたはJSON ファイルであることができます。フォーミュラパラメーターのセクションを参照してください。

▼ **message-lang**

--message-lang = *VALUE*
検証メッセージを表示する際に使用される言語を指定します。デフォルトの言語 :`en`. 使用することのできる他の言語 `de`, `es`, `ja`, それぞれ、ドイツ語、スペイン語、日本語を指します。

▼ **message-role**

--message-role = *VALUE*
検証メッセージを表示するために使用される優先メッセージを指定します。デフォルト :`http://www.xbrl.org/2010/role/message`.

▼ **preload-formula-schemas**

--preload-formula-schemas = `true|false`
XBRL Formula 1.0 仕様のスキーマをプリロードします。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。

▼ **process-assertion [a]**

--a | --process-assertion = *VALUE*
フォーミュラの実行を与えられたアサーションに制限します。1つ以上のアサーションを指定するため、オプションを複数回追加します。短い形式は `--a` です。***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

▼ **process-assertion-set [as]**

--as | --process-assertion-set = *VALUE*
フォーミュラの実行を与えられたアサーションのセットに制限します。1つ以上のアサーションセットを指定するため、オプションを複数回追加します。 短い形式は `--as` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

▼ **process-formula [f]**

--f | --process-formula = *VALUE*
フォーミュラの実行を与えられたフォーミュラに制限します。1つ以上のフォーミュラを指定するため、オプションを複数回追加します。短い形式は `--f` です。 ***VALUE*** は、`@id` 属性または、リソースを認識するXPointer フラグメント付きのURI の値です。 ##none や ##all などの特別な値も使用することができます。

▼ **report-unsatisfied-assertion-evaluations**

--report-unsatisfied-assertion-evaluations = `true|false`
アサーション重要度レベルに従い、条件を満たさないアサーションの評価をエラーまたは警告としてレポートします。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。

▼ **variableset-execution-timeout**

--variableset-execution-timeout = *VALUE*
(**--formula-execution=true**) フォーミュラを実行する際に適用されます。単一の変数のセットの実行に許可される最長時間を指定します (フォーミュラ 値、存在、整合性アサーション)。時間は分数で指定されており 正の数である必要があります。 デフォルトは 30分です。特定の変数セットが実行をタイムアウト前に完了し

ない場合は中断されます。エラーメッセージが表示されます（詳細ログに入力されます）。しかし、タイムアウトのチェックは各変数セットの評価の後で実行され、各 XPath 式の実行中には実行されないことに注意してください。ですから、単一の XPath 式の実行に長い時間が要される場合、タイムアウトの制限を過ぎる可能性があります。変数セットの実行は、完全な変数セットの評価が実行されてから、中断されます。

XBRL テーブルオプション

concept-label-linkrole

--concept-label-linkrole = *VALUE*
コンセプトラベルをレンダする際、優先する拡張リンクロールを指定します。

concept-label-role

--cconcept-label-role = *VALUE*
コンセプトラベルをレンダする際、優先するラベルロールを指定します。デフォルト：`http://www.xbrl.org/2003/role/label`。

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = `true|false`
`false` の場合、フォーミュラアサーションテーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト：`true`。

generic-label-linkrole

--generic-label-linkrole = *VALUE*
ジェネリックラベルをレンダする際、使用される優先する拡張されたリンクロールを指定します。

generic-label-role

--generic-label-role = *VALUE*
ジェネリックラベルをレンダする際、優先するラベルロールを指定します。デフォルト：http://www.xbrl.org/2003/role/label。

label-lang

--label-lang = *VALUE*
ラベルをレンダする際使用する優先ラベル言語を指定します。デフォルト：`en`。

preload-table-schemas

--preload-table-schemas = `true|false`
XBRL Table 1.0 仕様のスキーマをプリロードします。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。

process-table [t]

--t | --process-table = *VALUE*
フォーミュラの実行を与えられたテーブルに制限します。1つ以上のテーブルを指定するため、オプションを複数回追加します。短い形式は `--t` です。***VALUE*** は、`@id` 属性または、リソースを認識する XPointer フラグメント付きの URI の値です。##none や ##all などの特別な値も使用することができます。

table

--table = `true|false`
XBRL Table 1.0 拡張子を有効化します。デフォルト値は `true` です。`true` の場合自動的に--

formula=true および--dimensions=true を指定します。
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。

table-elimination

--table-elimination = true|false
HTML 出力内の空のテーブル行 列の削除を有効化します。デフォルトは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。

table-execution

--table-execution = true|false
XBRL テーブルの評価を有効化します。デフォルトはfalse です。 --table-output が指定されている場合、true に設定されます。true の場合自動的に--table=true を指定します。
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。

table-linkbase-namespace

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

前のドラフト仕様を使用して作成されたテーブルリンクベースのロードを有効化します。テーブルリンクベース検証、解像度、レイアウトはしかしながら、常に 2014年 3月 18日版 Table Linkbase 1.0 勧告に従い実行されます。##detect を使用して自動検出を有効化します。

table-output

--table-output = *FILE*
指定された*FILE* にテーブル出力を書き込みます。設定されると、自動的に--table-execution=true を指定します。

table-output-format

--table-output-format = xml|html
テーブル出力のフォーマットを指定します。デフォルトはxml です。

3.11.5 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：false。

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
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。

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。

3.11.6 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。

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。

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 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 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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプション および XML/XSD オプション）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 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 または XBRL 内の xsi:schemaLocation および xsi:noNamespaceSchemaLocation 属性 の スキーマの場所の URL 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, がある場合、--schema-mapping オプションの値が使用されます。（XBRL オプション および XML/XSD オプション）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。

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。

3.11.7 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：*

```
raptorxmlxbrl xslt --input=c:\Test.xml --output=c:\Output.xml --
param=date://node[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

updated-xml

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

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`。

xquery-update-version

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

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。

3.11.8 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：*

```
raptorxmlxbrl xslt --input=c:\Test.xml --output=c:\Output.xml --
param=date://node[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

streaming

--streaming = `true|false`

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。

template-entry-point

--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。

template-mode

--template-mode = *VALUE*

テンプレートモードが変換に使用されるように指定します。

xslt-version

--xslt-version = `1|1.0|2|2.0|3|3.0`

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。

3.11.9 JSON/ Avro

▼ **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。

▼ **codec**

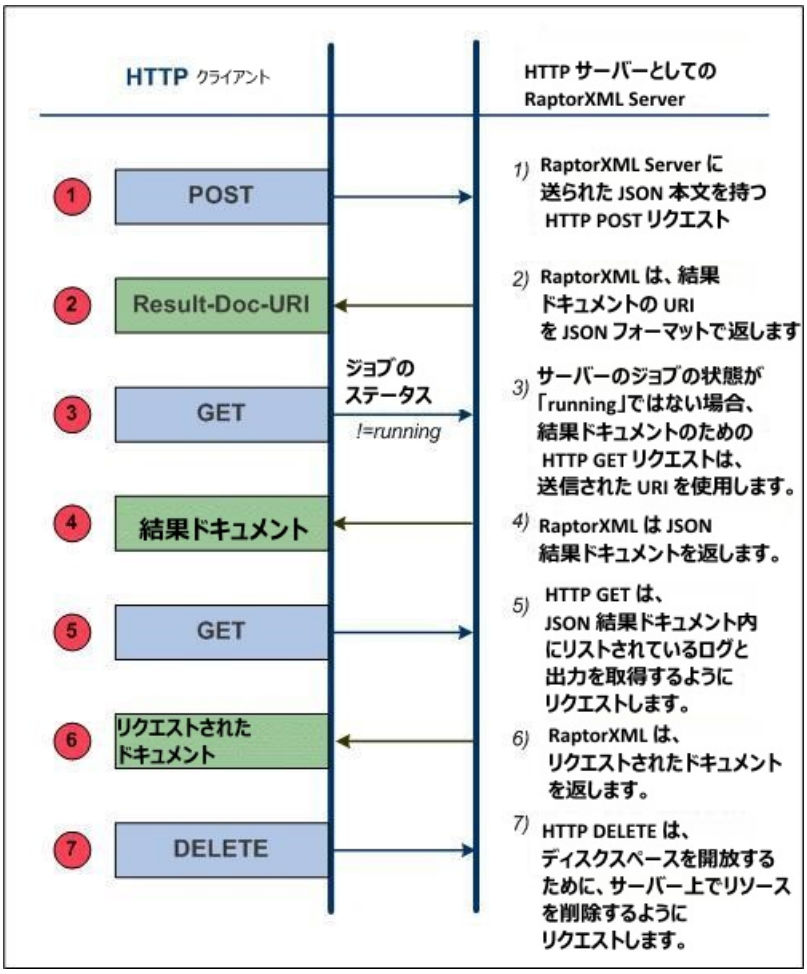
--codec = `null|deflate`
使用するAvro 圧縮コーデックを指定します。デフォルトは、`null` です。

チャプター 4

HTTP インターフェイス

4 HTTP インターフェイス

RaptorXML+XBRL Server は、HTTP (または HTTPS) 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 インターフェイスに関連するセキュリティの問題

HTTP 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+XBRL 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` 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+XBRL 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のセクションで説明されています。

4.1 サーバーセットアップ

RaptorXML+XBRL Server を正しくセットアップするには、以下を行います。RaptorXML+XBRL Server が既に正確にインストールおよびライセンス供与されていると仮定します。

1. RaptorXML+XBRL 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、Linux、Mac OS X。
2. サーバーへの接続をテストするために初期サーバー構成を使用します。（初期サーバー構成 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 `server_config.xml` を確認してください。ファイル内の設定を変更する場合は、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を編集するには、RaptorXML+XBRL 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、システムログを参照してください。

HTTPS に関する詳細は、HTTPS 設定 のセクションを参照してください。

4.1.1 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

RaptorXML+XBRL 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\RaptorXMLXBRLServer2017\bin\RaptorXMLXBRL.exe
```

RaptorXML+XBRL 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+XBRL Server をサービスとして Windows 上に登録します。ですが、RaptorXML+XBRL 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+XBRL Server"`
- コマンドプロンプトウィンドウ内の RaptorXML+XBRL Server 実行可能ファイルを使用して開始 : `RaptorXMLXBRLServer.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の `http.log-screen` 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+XBRL Server をサービスとして開始します:

[< Debian 8]	`sudo /etc/init.d/ raptorxmlxbrlserver start`
[≥ Debian 8]	`sudo systemctl start raptorxmlxbrlserver`
[< CentOS 7]	`sudo initctl start raptorxmlxbrlserver`
[≥ CentOS 7]	`sudo systemctl start raptorxmlxbrlserver`

[< Ubuntu 15]	sudo initctl start raptorxmlxbrlserver
[≥ Ubuntu 15]	sudo systemctl start raptorxmlxbrlserver
[RedHat]	sudo initctl start raptorxmlxbrlserver

RaptorXML+XBRL Server を停止する場合は、以下を使用します :

[< Debian 8]	sudo /etc/init.d/ raptorxmlxbrlserver stop
[≥ Debian 8]	sudo systemctl stop raptorxmlxbrlserver
[< CentOS 7]	sudo initctl stop raptorxmlxbrlserver
[≥ CentOS 7]	sudo systemctl stop raptorxmlxbrlserver
[< Ubuntu 15]	sudo initctl stop raptorxmlxbrlserver
[≥ Ubuntu 15]	sudo systemctl stop raptorxmlxbrlserver
[RedHat]	sudo initctl stop raptorxmlxbrlserver

Mac OS X 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+XBRL Server をサービスとして開始します:

```
sudo launchctl load /Library/LaunchDaemons/
com.altova.RaptorXMLXBRLServer2017.plist
```

RaptorXML+XBRL Server を停止する場合は、以下を使用します :

```
sudo launchctl unload /Library/LaunchDaemons/
com.altova.RaptorXMLXBRLServer2017.plist
```

4.1.2 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

RaptorXML+XBRL 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+XBRL 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 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+XBRL 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 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。

4.1.3 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

RaptorXML+XBRL 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\RaptorXMLXBRLServer2017\etc\server_config.xml
```

RaptorXML+XBRL 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\RaptorXMLXBRLServer2017\Output\`。

他のデフォルトの設定は `server_config.xml` の内のリスト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) を変更して、RaptorXML+XBRL 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:

```
RaptorXMLXBRL.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\RaptorXMLXBRLServer2017\output\</server.output-
root-dir>
  <server.script-root-dir>C:\Program Files\Altova\RaptorXMLXBRLServer2017\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\RaptorXMLXBRLServer2017\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\RaptorXMLXBRLServer2017\Log\access.log</http.access-
file>
  <http.error-file>C:\ProgramData\Altova\RaptorXMLXBRLServer2017\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\RaptorXMLXBRLServer2017\etc\cert\key.pem</
https.private-key>
  <https.certificate>C:\Program Files\Altova\RaptorXMLXBRLServer2017\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|ja (英語、ドイツ語、スペイン語、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]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+XBRL 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]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]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* 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*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\RaptorXMLXBRLServer2017\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+XBRL 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+XBRL 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+XBRL 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+XBRL Server がコマンド `RaptorXMLXBRLServer.exe debug` で開始された場合、 (*サーバーの開始* 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+XBRL 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+XBRL 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+XBRL 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 設定とSSL 暗号化のセットアップ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 とSSL 暗号化のセットアップを参照してください。

RaptorXML+XBRL 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、`100.60.300.6`、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+XBRL 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://100.60.300.6:8087/
```

メモ: `server_config.xml` が変更された後、RaptorXML+XBRL 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+XBRL 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+XBRL 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）。

4.1.4 HTTPS 設定

RaptorXML+XBRL 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 `server_config.xml` 内の**https.enable** 設定を**true** に変更します。構成ファイルの多種の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+XBRL Server に搭載されています：

```
python3.exe examples\ServerAPI\python\RunRaptorXML.py --host localhost -p
443 -s
```

4.1.5 SSL 暗号化のセットアップ

RaptorXML+XBRL 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+XBRL 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Mac OS X 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+XBRL Server にインストールされた**秘密キー**を必要とします。RaptorXML+XBRL 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+XBRL 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+XBRL 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+XBRL 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ー 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、関連機関と強力してください。

   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+XBRL 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 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、(i) セカンダリ中間証明書、(ii) 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。

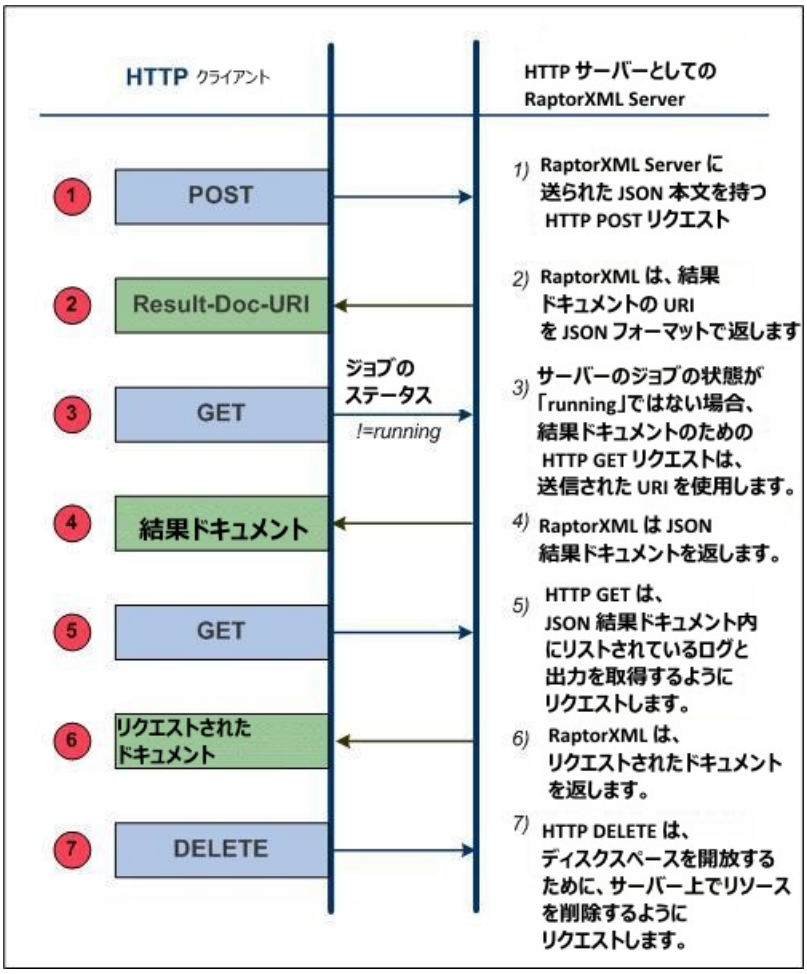
4.2 クライアントリクエスト

RaptorXML+XBRL Server が[サービスとして開始]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Web クライアントがあります。使用しやすく安定した、Web 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に使用されます。リクエストは RaptorXML+XBRL 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+XBRL Server (サーバーの初期設定アドレス) のアドレスと仮定されます。リクエストは、**RaptorXML+XBRL Server ジョブ** と称されます。

2. RaptorXML+XBRL 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。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。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+XBRL Server は、結果ドキュメントを JSON フォーマットで返します。結果ドキュメントは、元のリクエストの RaptorXML+XBRL 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。各リクエストは個別の `GET` メソッドで送信されます。

6. RaptorXML+XBRL 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。

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。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。

4.2.1 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

HTTP `POST` メソッドによりRaptorXML+XBRL 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{}、二重引用符、:、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。
- ファイルパスよりも、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/Testxslt"
           ],
  "options":{
         "input":"file:///C:/Work/Testxml",
         "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":"xsd",
  "args":[
        "file:///C:/Work/Testxml"
```

```
        ],
  "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 およびサンプル-2：カタログを使用したスキーマ検索を参照してください。

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](#)を参照してください。

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:

```
raptorxmlxbrl 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 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。I 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。

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--
```

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。

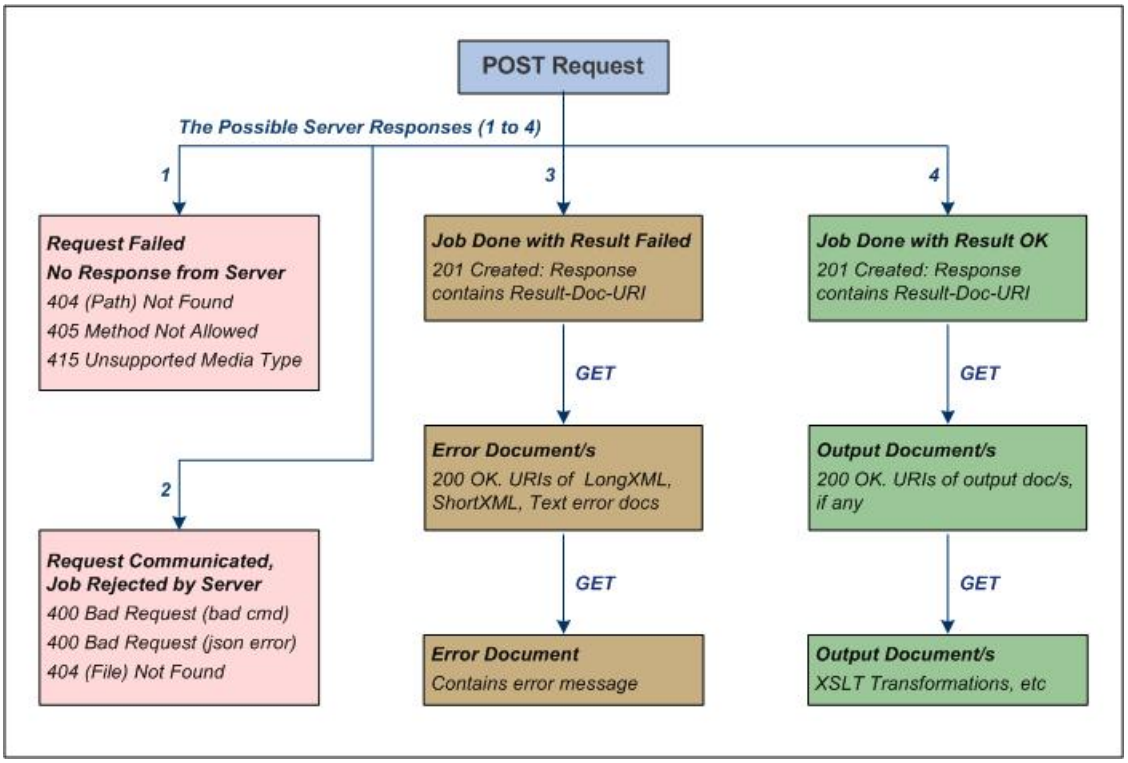
```
-----PartBoundary
Content-Disposition: form-data; name="msg"
Content-Type: application/json

{"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--
```

4.2.2 POST リクエストに対するサーバーの反応

このセクション:

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 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 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ト内に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 および エラー出力ドキュメントの取得 を参照してください。

4.2.3 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

(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)。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*)。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、また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*)。

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*)。
- `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で説明されています。

4.2.4 エラー /メッセージ /出力 ドキュメントの取得

結果ドキュメントは、ファイルURI または エラードキュメントの相対 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 もあります。)異なる種類のURI は、下に説明されています。

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
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

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 (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。

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` 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れます (*サンプルリストを参照*)。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)
```

結果ドキュメント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 および結果ドキュメントの取得 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+XBRL Server は、リクエストされたドキュメントを返します。

4.2.5 処理後のサーバーリソースの解放

RaptorXML+XBRL 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 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*] + [jobid]

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。

チャプター 5

Python と .NET API

5 Python と .NET API

RaptorXML+XBRL Server には、以下のAPI が搭載されています：

- Python API
- .NET Framework API

Altova LicenseServer にクライアントマシンを登録する

API パッケージをクライアントマシン上で作動するには、そのマシンは、RaptorXML+XBRL Server クライアントとしてライセンスが与えられます。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ます：

1. Altova LicenseServer にマシンをRaptorXML+XBRL Server クライアントとして登録します。
2. そのマシンへLicenseServer からRaptorXML+XBRL Server ライセンスを割り当てます。

与えられたマシンからAPI パッケージをプランする場合、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+XBRL 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+XBRL 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+XBRL 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+XBRL 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 +XBRL 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+XBRL Server クライアント として登録し、RaptorXML+XBRL 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+XBRL Server がインストールされていない)マシンをRaptorXML+XBRL 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\RaptorXMLXBRLServer2017\bin`
Linux	`/opt/Altova/RaptorXMLXBRLServer2017/bin`
Mac	`/usr/local/Altova/RaptorXMLXBRLServer2017/bin`

コマンドラインは以下のコマンドを実行します：

<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、

```
registerlicense <LicenseServer>
```

このコマンドは、マシンをRaptorXML+XBRL Server クライアントとしてAltova LicenseServer に登録します。RaptorXML+XBRL 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 ドキュメント。

Linux にデプロイする

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/RaptorXMLServer2017/lib

- libcrypto.so.1.0.0
- libssl.so.1.0.0
- libstdc++.so.6
- libtbb.so.2

5.1 Python API

RaptorXML+XBRL Server のPython インターフェイスは、Python API を介して、XML、XSD およびXBRL のためのXML ドキュメント、XML スキーマ ドキュメント、XBRL インスタンスドキュメント、およびXBRL タクソノミ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+XBRL 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、XSD およびXBRLのための)Python API は、XML、XSD、およびXBRL インスタンスとタクソノミ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 またはXBRL インスタンス 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+XBRL 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

Python のための RaptorXML+XBRL Server パッケージ

RaptorXML+XBRL 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。RaptorXMLXBRL 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、RaptorXMLXBRL 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+XBRL 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+XBRL Server 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)`
- `valxsd (xsd)`
- `valxbrltaxonomy (dts)`
- `valxbrl (xbrl)`

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) およびHTTP インターフェイスを介して使用することができます。RaptorXML+XBRL 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+XBRL 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の `server.script-root-dir` 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` のサブディレクトリである) ジョブ出力ディレクトリ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。

5.1.1 Python API Versions

RaptorXML+XBRL 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+XBRL Server により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+XBRL 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+XBRL 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+XBRL 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+XBRL 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+XBRL Server v2014 で紹介されました。

コマンドラインフラグ :	**`--script-api-version=1`**
ドキュメント :	Python API バージョン 1 レファレンス

これは、オリジナルの RaptorXML+XBRL Server Python API です。以下の RaptorXML+XBRL 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+XBRL 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。

Python API バージョン 2

RaptorXML+XBRL Server v2015r3 で紹介されました。最新の API バージョンは `2.4` です。

コマンドライン フラグ:	`--script-api-version=2`
	`--script-api-version=2.1`
	`--script-api-version=2.2`
	`--script-api-version=2.3`
	`--script-api-version=2.4`
		v 2015r3
		v 2015r4
		v 2016
		v 2016r2
		v2017

ドキュメント:	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+XBRL 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+XBRL 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。

5.1.2 Python パッケージとしての RaptorXML+XBRL Server

RaptorXML+XBRL Server 2017 から、Python API がネイティブの Python ホイール パッケージ **Python 3.5** 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 3.5 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+XBRL 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+XBRL 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+XBRL 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 モジュールで、インストールされている Python バージョンに一致する必要があります。

RaptorXML+XBRL 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
- ホイール ファイルのロケーション
- pip を使用してホイールをインストールする
- ルート カタログ ファイル
- JSON 構成ファイル

RaptorXML+XBRL Server の Python API の使用方法に関しての詳細は、Python API リファレンスとサンプル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-2.4.0-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+XBRL 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\RaptorXMLXBRLServer2017\bin`
Linux	`/opt/Altova/RaptorXMLXBRLServer2017/bin`
Mac	`/usr/local/Altova/RaptorXMLXBRLServer2017/bin`

---

pip を使用してホイールをインストールする

RaptorXML+XBRL 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 3.5 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 RaptorXMLXBRL モジュールは、RaptorXML+XBRL 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、RaptorXMLXBRL 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。RaptorXMLXBRL モジュールは、RaptorXML+XBRL 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+XBRL 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 RaptorXMLXBRL モジュールの JSON 構成ファイル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。 RaptorXMLXBRL 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\RaptorXMLXBRLServer\Installation_v2017_x64\Setup\CatalogPath`	レジストリキーは、RaptorXML+XBRL 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/RaptorXMLXBRLServer2017/etc/RootCatalog.xml`	ハードコードされています。*Linux のみ*
4	ロケーション:`/usr/local/Altova/RaptorXMLXBRLServer2017/etc/RootCatalog.xml`	ハードコードされています。*Mac のみ*
5	環境変数 `ALTOVA_RAPTORXML_PYTHON_CONFIG`	JSON 構成ファイル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
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
8	ロケーション:`/etc/altova/altova/`**`raptorxml-python.config`**	JSON 構成ファイル。*Linux と Mac のみ*

JSON 構成ファイル

RaptorXMLXBRLServer 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]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]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"
}
```

5.2 .NET Framework API

RaptorXML+XBRL Server の**.NET Framework API** により、RaptorXML+XBRL Server をC# と他の .NET 言語で書かれアプリケーションに統合することができます。

これは、.NET アセンブリとして実装され、RaptorXML+XBRL Server を直接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置きます。API は、ドキュメントを検証するためのアクセスであり、RaptorXML+XBRL 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。このメカニズムはのRaptorXML+XBRL Server のPython API ために使用されるメカニズムに類似しています。

レファレンスとリソース

- *API ドキュメント*:最新のRaptorXML+XBRL 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http://manual.altova.com/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。

チャプター 6

Java インターフェイス

6 Java インターフェイス

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+XBRL 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。

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` 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` 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。

メモ: `getFactory` メソッドは、インストールされたRaptorXML のエディションに従い、対応するファクトリオブジェクトを返します。

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

詳細に関しては、`RaptorXMLFactory` 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 プロジェクトも参照することができます。

6.1 Java プロジェクトの例

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。
- XML ファイルの検証
- XSLT 変換を実行し、結果を文字列として返す
- XQuery ドキュメントの処理後、結果を文字列として返す
- プロジェクトの実行

基本的な機能は、RaptorXML+XBRL 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.raptorxmlRaptorXMLException
    {
       System.out.println("RaptorXML Java - XQuery execution");
       com.altova.raptorxmlXQuery 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.raptorxmlRaptorXMLfactory();
          rxml.setErrorLimit(3);

          ValidateXML();
          RunXSLT();
          RunXQuery();
       }

       catch(com.altova.raptorxmlRaptorXMLException e)
       {
          e.printStackTrace();
       }

    }

}
```

6.2 Java のための RaptorXML インターフェイス

RaptorXML API の Java のインターフェイスのまとめが下に説明されています。詳細はそれぞれのセクションで説明されています。

- RaptorXMLFactory
  ネイティブ呼び出しを使用して新しい RaptorXML COM オブジェクトインスタンスを作成し、RaptorXML エンジンにアクセスします。ます。
- XMLValidator
  XMLValidator エンジンのためのインターフェイス。
- XSLT
  XSLT エンジンのためのインターフェイス。
- XQuery
  XQuery エンジンのためのインターフェイス。
- XBRL
  XBRL エンジンのためのインターフェイス。
- RaptorXMLException
  RaptorXMLException メソッドのためのインターフェイス。

6.2.1 RaptorXMLFactory

public interface RaptorXMLFactory

RaptorXMLFactory() を使用して新しいRaptorXML COM オブジェクトインスタンスを作成することができます。これによりRaptorXML エンジンにアクセスすることができます。RaptorXMLFactory とRaptorXML COM オブジェクトの関係は、一対一です。これは、続くget<*ENGINENAME*>()関数への呼び出しが同じエンジンインスタンスのインターフェイスを返すことを意味します。インターフェイスの メソッド が、機能別に説明されています。 **RaptorXMLFactory** インターフェイスの 列挙 は別のセクションで説明されています。

エンジン
対応するエンジンの呼び出しメソッド。

▼ **getXBRL**

public XBRL getXBRL
XBRL エンジンを取得します。RaptorXMLFactory の新しいXBRL インスタンスを返します。

▼ **getXMLValidator**

public XMLValidator getXMLValidator
XMLValidator エンジンを取得します。このRaptorXMLFactory の新しいXMLValidator インスタンスを返します。

▼ **getXQuery**

public XQuery getXQuery
XQuery エンジンを取得します。このRaptorXMLFactory の新しいXQuery インスタンスを返します。
RaptorXMLFactory.

▼ **getXSLT**

public XSLT getXSLT
XSLT エンジンを取得します。このRaptorXMLFactory の新しいXSLT インスタンスを返します。

エラーと警告
エラーと警告のためのパラメータを設定します。

▼ **setErrorFormat**

public void setErrorFormat(*ENUMErrorFormat* **format)**
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)。

▼ **setErrorLimit**

public void setErrorLimit(*int* **limit)**
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。

▼ **setReportOptionalWarnings**

```
public void setReportOptionalWarnings(boolean report)
```

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。

カタログとグローバルリソース

これらのメソッドは使用するカタログファイルの場所を提供します。

▼ **setGlobalCatalog**

```
public void setGlobalCatalog(String catalog)
```

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。

▼ **setUserCatalog**

```
public void setUserCatalog(String catalog)
```

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。

▼ **setGlobalResourceConfig**

```
public void setGlobalResourceConfig(String config)
```

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。

▼ **setGlobalResourceFile**

```
public void setGlobalResourceFile(String file)
```

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。

HTTP サーバー設定

これらのメソッドは HTTP サーバー名とポートを設定し、サーバーの構成ファイルを指定します。

▼ **setServerFile**

```
public void setServerFile(String file)
```

HTTP サーバーアドレスに相対する、HTTP サーバーの構成ファイルの場所を設定します。エラーが発生すると、**RaptorXMLException** が挙げられます (*Java インターフェイスの詳細*)。入力パラメーターはサーバーアドレスに相対した HTTP サーバー構成ファイルのアドレスを与える文字列。

▼ **setServerName**

```
public void setServerName(String name)
```

HTTP サーバーの名前を設定します。エラーが発生すると、**RaptorXMLException** が挙げられます (*Java インターフェイスの詳細*)。入力パラメーターは、HTTP サーバー名を与える文字列です。

▼ **setServerPort**

```
public void setServerPort(int port)
```

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** が挙げられます (*Java インターフェイスの詳細*)。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

製品の情報

これらのメソッドはインストールされた製品の情報を収集します。

▼ **getProductName**

```
public string getProductName()
```

製品の名前を文字列として返します。 *サンプル*: `Altova RaptorXML+XBRL Server 2017r2sp1(x64)` に対しては、`Altova RaptorXML+XBRL Server` が返されます。エラーが発生すると、**RaptorXMLException** が挙げられます (*Java インターフェイスの詳細*)。

▼ **getProductNameAndVersion**

```
public string getProductNameAndVersion()
```

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+XBRL Server 2017r2sp1(x64)` に対しては、`Altova RaptorXML+XBRL Server 2017r2sp1(x64)` が返されます。エラーが発生すると、**RaptorXMLException** が挙げられます (*Java インターフェイスの詳細*)。

▼ **getMajorVersion**

```
public int getMajorVersion()
```

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+XBRL 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** が挙げられます (*Java*)。

▼ **getMinorVersion**

```
public int getMinorVersion()
```

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+XBRL Server 2017r2sp1(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** が挙げられます (*Java インターフェイスの詳細*)。

▼ **getServicePackVersion**

```
public int getServicePackVersion()
```

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+XBRL Server 2017r2sp1(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`** が挙げられます (*Java インターフェイスの詳細*)。

is64Bit

```
public boolean 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+XBRL Server 2017r2sp1(x64)` に対しては、`true` を返します。エラーが発生すると、**`RaptorXMLException`** が挙げられます (*Java インターフェイスの詳細*)。

getAPIMajorVersion

```
public int getAPIMajorVersion()
```

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と異なる場合があります

getAPIMinorVersion

```
public int getAPIMinorVersion()
```

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と異なる場合があります。

getAPIServicePackVersion

```
public int getAPIServicePackVersion()
```

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と異なる場合があります。

6.2.2 XMLValidator

public interface XMLValidator

与えられた XML ドキュメント、スキーマドキュメント、および DTD ドキュメントを検証します。XML ドキュメントの検証は、内部および外部 DTD、または XML スキーマにより実行されることができます。また、XML、DTD、および XML スキーマ ドキュメントの整形式をチェックします。インターフェイスのメソッドは、下に説明されており、機能別にグループ化されています。

処理

検証のパラメータを指定し、検証の情報を収集するメソッド

▾ **extractAvroSchema**

public boolean extractAvroSchema(*String* outputPath)
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** が挙げられます (*Java インターフェイスに移動*)。`getLastErrorMessage` (***Java 内で***)、または、`LastErrorMessage` (***COM/.NET 内で***) を追加情報にアクセスするために使用します。

▾ **isValid(ENUM type)**

public boolean isValid(*ENUMValidationType* type)
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` リテラル (Java、COM/.NET) を値として取る、`type` パラメータ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、a **RaptorXMLException** が挙げられます (*Java インターフェイスに移動*)。`getLastErrorMessage` (***Java 内で***)、または、`LastErrorMessage` (***COM/.NET 内で***) を追加情報にアクセスするために使用します。

▾ **isValid**

public boolean isValid()
提供されたドキュメントの検証の結果を返します。結果は成功時には`true`、失敗時には`false` です。

▾ **isWellFormed(ENUM type)**

public boolean isWellFormed(*ENUMWellformedCheckType* type)
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` を取るリテラル (Java、COM/.NET)。`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、a **RaptorXMLException** が挙げられます (*Java インターフェイスに移動*)。`getLastErrorMessage` (***Java 内で***)、または、`LastErrorMessage` (***COM/.NET 内で***) を追加情報にアクセスするために使用します。

▾ **isWellFormed**

public boolean 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。 結果は成功時には`true`、失敗時には`false` です。

▼ **getLastErrorMessage**

```
public String getLastErrorMessage()
```

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。

▼ **setAssessmentMode**

```
public void setAssessmentMode(ENUMAssessmentMode mode)
```

ENUMAssessmentMode リテラル(Java、COM/.NET)を取るmode パラメータ内で定義される、XML 検証の評価モード (Strict/Lax) を設定します。

▼ **setPythonScriptFile**

```
public void setPythonScriptFile(String file)
```

Python スクリプトファイルのロケーションを設定します。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。

▼ **setStreaming**

```
public void setStreaming(boolean support)
```

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。

入力ファイル

検証コマンドの入力ファイルの検証コマンド (XML、XML スキーマ、および DTD) を指定するメソッド

▼ **setAvroSchemaFileName**

```
public void setAvroSchemaFileName(String filePath)
```

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。

▼ **setAvroSchemaFromText**

```
public void setAvroSchemaFromText(String schemaText)
```

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。

▼ **setInputFileName**

```
public void setInputFileName(String filePath)
```

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。

▼ **setInputFileCollection**

```
public void setInputFileCollection(Collection<?> fileCollection)
```

入力データとして使用されるXML 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

が入力 XML ファイルの絶対 URL である、文字列のコレクション。

▼ **setInputFromText**

public void setInputFromText(*String* inputXMLText)
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。

▼ **setInputTextCollection**

public void setInputTextCollection(*Collection<?>* stringCollection)
入力データとして使用される複数のXML ファイルを提供します。それぞれが入力 XML ファイルのコンテンツである、文字列のコレクション。

▼ **setInputXMLFileCollection (DEPRECATED. Use setInputFileCollection instead.)**

public void setInputXMLFileCollection(*Collection<?>* fileCollection)
入力データとして使用されるXML ファイルのコレクションが提要されます。ファイルはURL により識別されます。それぞれが入力 XML ファイルの絶対 URL である文字列のコレクションです。

▼ **setInputXMLFileName (DEPRECATED. Use setInputFileName instead.)**

public void setInputXMLFileName(*String* xmlFile)
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。

▼ **setInputXMLFromText (DEPRECATED. Use setInputFromText instead.)**

public void setInputXMLFromText(*String* inputXMLText)
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。

▼ **setInputXMLTextCollection(DEPRECATED. Use setInputTextCollection instead.)**

public void setInputXMLTextCollection(*Collection<?>* stringCollection)
入力データとして使用される複数のXML ファイルを提供します。それぞれがXML ファイルのコンテンツである文字列のコレクションです。

▼ **setSchemaFileCollection**

public void setSchemaFileCollection(*Collection<?>* fileCollection)
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。

▼ **setSchemaFileName**

public void setSchemaFileName(*String* filePath)
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。

▼ **setSchemaFromText**

public void setSchemaFromText(*String* schemaText)
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。

▼ **setSchemaTextCollection**

public void setSchemaTextCollection(*Collection<?>* stringCollection)
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。

▼ **setDTDFileName**

public void setDTDFileName(*String* filePath)
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。

▼ **setDTDFromText**

public void setDTDFromText(*String* dtdText)
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。

XML スキーマ

検証に使用されるXML スキーマのオプションを設定するメソッド

▼ **setSchemaImports**

public void setSchemaImports(*ENUMSchemaImports* opt)
`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 (Java、COM/.NET) により指定されます。

▼ **setSchemalocationHints**

public void setSchemalocationHints(*ENUMLoadSchemalocation* opt)
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 (Java、COM/.NET) により指定されます。

▼ **setSchemaMapping**

public void setSchemaMapping(*ENUMSchemaMapping* opt)
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 (Java、COM/.NET) により指定されます。 .

▼ **setXSDVersion**

public void setXSDVersion(*ENUMXSDVersion* version)
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` (Java、

COM/.NET) は、の列挙リテラルです。

XML

入力 XML データに関連するオプションを指定するメソッド

setEnableNamespaces

```
public void setEnableNamespaces(boolean enable)
```

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。

setXincludeSupport

```
public void setXincludeSupport(boolean support)
```

`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。

setXMLValidationMode

```
public void setXMLValidationMode(ENUMXMLValidationMode mode)
```

有効性または整形式をチェックするかを決定する、`ENUMXMLValidationMode` (Java、COM/.NET) の列挙リテラルである XML 検証モードを設定します。

6.2.3 XSLT

```
public interface XSLT
```

与えられた XSLT 1.0、2.0、または 3.0 ドキュメンを使用して 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テキスト文字列として与えられることができます。出力はファイルとして (名前の付けられた場所で) 返されます。XSLT パラメーターは与えられることができ、チャートなどの、Altova 拡張関数は、特別な処理のために有効化されることができます。XSLT ドキュメントも検証されることができます。文字列の入力が URL と解釈される箇所では、絶対パスが使用される必要があります。XSLT インターフェイスのメソッドが、最初に説明されています。

処理
XSLT 変換のパラメーターを指定するメソッド。

▼ **isValid**

public boolean isValid()
ENUMXSLTVersion で挙げられている仕様に従い、XSLT ドキュメントの検証の結果を返します。(setVersion メソッドを参照してください)。結果は成功時には true、失敗時には false です。エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **execute**

public boolean execute(*String* outputFile)
ENUMXSLTVersion 内で名前が挙げられているXSLT 仕様に従い、XSLT 変換を実行します。(setVersion メソッドを参照してください)。また、outputFile パラメーター内で名前を付けられた出力ファイルの結果を保存します。エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。
パラメーター:
outputFile: 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。

▼ **executeAndGetResultAsString**

public String executeAndGetResultAsString()
ENUMXSLTVersion 内で名前が挙げられている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。(setVersion メソッドを参照してください)。このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。追加出力ファイルが必要な場合は、execute メソッドを使用してください。エラーが発生した場合 RaptorXMLException が挙げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **executeAndGetResultAsStringWithBaseOutputURI**

public String executeAndGetResultAsStringWithBaseOutputURI()
ENUMXSLTVersion 内で名前が挙げられているXSLT 仕様に従い、XSLT 変換を実行します。(setVersion メソッドを参照してください)。)また、結果をベースURI により定義された場所で文字列として返します。このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。追加出力ファイルが必要な場合は、execute メソッドを使用してください。エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **getMainOutput**

public String getMainOutput()

最後に実行されたジョブのメイン出力を返します。

▼ **getAdditionalOutputs**

```
public String getAdditionalOutputs()
```

最後に実行されたジョブの追加出力を返します。

▼ **getLastErrorMessage**

```
public String getLastErrorMessage()
```

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。

▼ **setIndentCharacters**

```
public void setIndentCharacters(String chars)
```

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。

▼ **setStreamingSerialization**

```
public void setStreamingSerialization(boolean support)
```

ストリーミング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、ストリーミングのシリアル化を有効化します。`false` の値は無効化します。

XSLT

XSLT スタイルシートに関連したオプションを指定するメソッド

▼ **setVersion**

```
public void setVersion(ENUMXSLTVersion version)
```

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。
パラメーター:
version: EnumXSLTVersion は、以下のいずれかの列挙リテラルを保有します。eVersion10、eVersion20、または eVersion30。

▼ **setXSLFileName**

```
public void setXSLFileName(String xslFile)
```

変換に使用される XSLT ドキュメントの場所を URL として設定します。
パラメーター:
xslFile: 提供される文字列は、使用する XSLT 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。

▼ **setXSLFromText**

```
public void setXSLFromText(String xslText)
```

XSLT ドキュメントのコンテンツをテキストとして提供します。
パラメーター:

xslText: 与えられてた文字列は変換に使用されるXSLT ドキュメントです。

addExternalParameter

```
public void addExternalParameter(String name, String value)
```

新しい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、メソッドの個別の呼び出しで指定されます。パラメーターは XSLT ドキュメント内で宣言される必要があります。パラメーターの値は、XPath 式であり、文字列であるパラメーターの値は、一重引用符で囲まれていなければなりません。

パラメーター:

name: 文字列として QName であるパラメーターの名前を保持します。

value: 文字列としてのパラメーターの値を保有します。

clearExternalParameterList

```
public void clearExternalVariableList()
```

AddExternalParameter 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。

setInitialTemplateMode

```
public void setInitialTemplateMode(String mode)
```

初期テンプレートモードの名前を設定します。このモードの値を持つテンプレートと共に処理が開始されます。変換は XML と XSLT ドキュメントを割り当てた後に開始されなければなりません。

パラメーター:

mode: 文字列としての最初のテンプレートの名前。

setNamedTemplateEntryPoint

```
public void setNamedTemplateEntryPoint(boolean template)
```

開始する処理と共に、名前の付けられたテンプレートの名前を設定します。

パラメーター:

template: 文字列としての名前を付けられたテンプレートの名前。

XML スキーマ

検証に使用されるXML スキーマのオプションを設定するメソッド。

setSchemaImports

```
public void setSchemaImports(ENUMSchemaImports opt)
```

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 (Java、COM/.NET) により指定されます。

setSchemalocationHints

```
public void setSchemalocationHints(ENUMLoadSchemalocation opt)
```

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 (Java、COM/.NET) により指定されます。

▼ **setSchemaMapping**

```
public void setSchemaMapping(ENUMSchemaMapping opt)
```

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 (Java、COM/.NET) により指定されます。 .

▼ **setXSDVersion**

```
public void setXSDVersion(ENUMXSDVersion version)
```

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 (Java、COM/.NET) は、の列挙リテラルです。

XML

処理される XML データに関連するパラメータを指定するメソッド。

▼ **setInputXMLFileName**

```
public void setInputXMLFileName(String xmlFile)
```

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。

▼ **setInputXMLFromText**

```
public void setInputXMLFromText(String inputXMLText)
```

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。

▼ **setLoadXMLWithPSVI**

```
public void setLoadXMLWithPSVI(boolean psvi)
```

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。

▼ **setXincludeSupport**

```
public void setXincludeSupport(boolean support)
```

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。

▼ **setXMLValidationErrorAsWarning**

```
public void setXMLValidationErrorAsWarning(boolean enable)
```

有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode (Java、COM/.NET) の列挙リテラルである XML 検証モードを設定します。

▼ **setXMLValidationMode**

public void setXMLValidationMode(*ENUMXMLValidationMode* mode)
有効性または整形式をチェックするかを決定する ENUMXMLValidationMode (Java、COM/.NET) の列挙リテラルであるXML 検証モードを設定します。

拡張子

チャートなどの特別な処理のためにAltova 拡張関数 が有効化されるか指定するメソッド。

▼ **setChartExtensionsEnabled**

public void setChartExtensionsEnabled(*boolean* enable)
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。

▼ **setDotNetExtensionsEnabled**

public void setDotNetExtensionsEnabled(*boolean* enable)
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。

▼ **setJavaExtensionsEnabled**

public void setJaveExtensionsEnabled(*boolean* enable)
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。

▼ **setJavaBarcodeExtensionLocation**

public void setJavaBarcodeExtensionLocation(*String* path)
バーコード拡張ファイルAltovaBarcodeExtension.jar を含むフォルダーへのパスを指定します。詳細に関しては、Altova バーコード拡張関数 のセクションを参照してください。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\\"

パラメーター :
path: 提供される文字列は、使用する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。

6.2.4 XQuery

```
public interface XQuery
```

RaptorXML エンジンを使用して XQuery 1.0 と 3.0 ドキュメントを実行します。XQuery および XML ドキュメントは (URL を介して) ファイルとしてまたはテキスト文字列として与えられることができます。出力はテキスト文字列として (名前の付けられた場所で) 返されます。外部 XQuery 変数が提供されることができ、多数のシリアル化オプションを使用することができます。 XQuery ドキュメントも検証されることができます。文字列の入力が URL と解釈される箇所では、絶対パスが使用されるべきです。XQuery インターフェイスの メソッド では機能がグループ別に説明されています。

処理
XQuery 実行のパラメーターを指定するメソッド

▼ **isValid**

```
public boolean isValid()
```

ENUMXQueryVersion で挙げられているXQuery 仕様に従い、XQuery ドキュメントの検証の結果を返します。 (setVersion メソッドを参照してください)。結果は成功時にはtrue 、失敗時にはfalse です。エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **isValidUpdate**

```
public boolean isValidUpdate()
```

ENUMXQueryVersion で挙げられている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。 (setVersion メソッドを参照してください)。
結果は成功時にはtrue 、失敗時にはfalse です。エラーが発生した場合、RaptorXMLException が挙げられます。 追加情報にアクセスするには、 getLastErrorMessage メソッドを使用してください。

▼ **execute**

```
public boolean execute(String outputFile)
```

ENUMXQueryVersion で挙げられているXQuery 仕様に従い、XQuery ドキュメントの検証の結果を返します。 (setVersion メソッドを参照してください)。結果は成功時にはtrue 、失敗時にはfalse です。エラーが発生した場合、RaptorXMLException が挙げられます。 追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **executeAndGetResultAsString**

```
public String executeAndGetResultAsString()
```

ENUMXQueryVersion 内で名前が挙げられているXSLT 仕様に従い、XSLT 変換を実行し変換の結果を文字列として返します。(setVersion プロパティを参照してください)。このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。追加出力ファイルが必要な場合は、execute メソッドを使用してください。

▼ **executeUpdate**

```
public boolean executeUpdate(String outputFile)
```

ENUMXQueryVersion で挙げられているXQuery Update 仕様に従いXQuery アップデートを実行します。(setVersion メソッドを参照してください)。 そして、結果を outputFile パラメーターで名前の付けられた出力ファイルに保存します。 パラメーターは出力ファイルの場所 (パスとファイル名) を提供する文字列。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 が揚げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **executeUpdateAndGetResultAsString**

public String executeUpdateAndGetResultAsString()
ENUMXQueryVersion 内で名前が挙げられている XQuery Update 仕様に従い、XQuery Update 変換を実行し結果を文字列として返します。(setVersion メソッドを参照してください)。このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。メソッドを使用してください。execute メソッドを使用してください。

▼ **getLastErrorMessage**

public String getLastErrorMessage()
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。

▼ **setUpdatedXMLWriteMode**

public void setUpdatedXMLWriteMode(*ENUMXQueryUpdatedXML* mode)
更新に使用するモードを設定します。ENUMXQueryUpdatedXML 結果リテラルを取ります：eUpdatedDiscard, eUpdatedWriteback または eUpdatedAsMainResult。

XML
処理される XML データに関連したパラメーターを指定するメソッド。

▼ **setInputXMLFileName**

public void setInputXMLFileName(*String* xmlFile)
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。

▼ **setInputXMLFromText**

public void setInputXMLFromText(*String* inputXMLText)
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。

▼ **setLoadXMLWithPSVI**

public void setLoadXMLWithPSVI(*boolean* psvi)
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。

▼ **setXincludeSupport**

public void setXincludeSupport(*boolean* support)

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。

▼ **setXMLValidationErrorAsWarning**

public void **setXMLValidationErrorAsWarning**(boolean enable)
有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode (Java、COM/.NET) の列挙リテラルである XML 検証モードを設定します。

▼ **setXMLValidationMode**

public void setXMLValidationMode(*ENUMXMLValidationMode* **mode**)
有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode (Java、COM/.NET) の列挙リテラルである XML 検証モードを設定します。

▼ **setXSDVersion**

public void setXSDVersion(*ENUMXSDVersion* **version**)
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 (Java、COM/.NET) は、の列挙リテラルです。

XQuery

XQuery ドキュメントに関連下オプションを指定するメソッド。

▼ **setVersion**

public void setVersion(*ENUMXQueryVersion* **version**)
処理に使用される XQuery バージョンを設定します。(検証または XQuery 実行)。ENUMXQueryVersion 列挙リテラルを取ります :eVersion10 または eVersion30。デフォルトは、eVersion30ml です。

▼ **setXQueryFileName**

public void setXQueryFileName(*String* **queryFile**)
実行される XQuery 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 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。

▼ **setXQueryFromText**

public void setXQueryFromText(*String* **queryText**)
テキストとしての XQuery 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される XQuery ドキュメントです。

▼ **addExternalVariable**

public void addExternalVariable(*String* **name**, *String* **value**)
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、メソッドの個別の呼び出し内で指定されなければなりません。変数は (任意の型の宣言と共に)XQuery ドキュメント内で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 である変数の名前を文字列として保有します。value パラメーターは変数の値を文字列として保有します。

▼ **clearExternalVariableList**

public void clearExternalVariableList()
AddExternalVariable メソッドにより作成された外部変数リストをクリアします。

シリアル化のオプション

処理の出力のプロパティを指定するメソッド。

▼ **setIndentCharacters**

public void setIndentCharacters(*String* chars)
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。

▼ **setKeepFormatting**

public void setKeepFormatting(*boolean* keep)
executeUpdate によりアップデートされるファイルの元の書式を保有するオプションを有効化または無効化します。keep パラメーターはブール値の true または false を取ります。

▼ **setOutputEncoding**

public void setOutputEncoding(*String* encoding)
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。

▼ **setOutputIndent**

public void setOutputIndent(*boolean* indent)
出力ドキュメント内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 の値は無効化します。

▼ **setOutputMethod**

public void setOutputMethod(*String* outputMethod)
出力ファイルのシリアル化を指定します。効なファイル:xml | xhtml | html | text は文字列として与えられます。デフォルトの値は xml です。

▼ **setOutputOmitXMLDeclaration**

public void setOutputOmitXMLDeclaration(*boolean* omit)
結果ドキュメント内のXML 宣言を有効化、または、無効化します。true の値は、宣言を無視します。false の値は宣言を含みます。デフォルトの値 :false.

拡張子

チャートなど、Altova 拡張関数が特別な処理のために有効化されるか指定するメソッド。

▼ **setChartExtensionsEnabled**

```
public void setChartExtensionsEnabled(boolean enable)
```

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。

▼ **setDotNetExtensionsEnabled**

```
public void setDotNetExtensionsEnabled(boolean enable)
```

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。

▼ **setJavaExtensionsEnabled**

```
public void setJaveExtensionsEnabled(boolean enable)
```

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。

6.2.5 XBRL

public interface XBRL

提供された XBRL インスタンスドキュメントまたは XBRL タクソノミドキュメントを検証します。インターフェイスの メソッドが 最初に説明されており、続いて 列挙 について説明されています。

ユーティリティクラス

FormulaParam のためのユーティリティクラスが定義されます。2 つのメンバーと 1 つのコンストラクタを有します。

▼ **ParamValuePair**

```
public class ParamValuePair
public class ParamValuePair
  {
    public String paramType;
    public String paramValue;
    public ParamValuePair(String type,String value )
    {
            paramType = type;
            paramValue = value;
    }
  };
```

処理

検証に関する情報を取得するための検証のパラメーターを指定するメソッド。

▼ **isValid(ENUM type)**

```
public boolean isValid(ENUMValidationType type)
```

XBRL インスタンスドキュメントまたは XBRL タクソノミ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 リテラルを値として取る、type パラメータ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。エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、getLastErrorMessage メソッドを使用してください。

▼ **isValid**

```
public boolean isValid()
```

提供された XBRL ドキュメントの検証の結果を返します。結果は成功時には true、失敗時には false です。

▼ **getLastErrorMessage**

```
public String getLastErrorMessage()
```

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。

▼ **setEvaluateReferencedParametersOnly**

```
public void setEvaluateReferencedParametersOnly(boolean enable)
```

フォーミュラアサーションテーブルに参照されていない場合でも false の場合は、すべてのパラメーターの検証を強制します。デフォルトは true です。

▼ **setParallelAssessment**

```
public void setParallelAssessment(boolean support)
```

パラレル評価の使用を有効化または無効化します。 の値は、を有効化します。`true` はパラレル評価を有効化します。`false` の値は無効化します。デフォルト値は `false` です。

▼ **setPythonScriptFile**

```
public void setPythonScriptFile(String file)
```

Python スクリプトファイルの URL としてのロケーションを設定します。提供される文字列は、使用する 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。

入力ファイル

検証コマンドの入力を指定するメソッド。

▼ **setInputFileName**

```
public void setInputFileName(String filePath)
```

検証される XBRL 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 XBRL 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。

▼ **setInputFileCollection**

```
public void setInputFileCollection(Collection<?> fileCollection)
```

入力データとして使用される XBRL 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。`fileCollection` パラメーターは入力 XBRL ファイルの絶対 URL である文字列のコレクションです。

▼ **setInputFromText**

```
public void setInputFromText(String schemaText)
```

XBRL ドキュメントのコンテンツをテキストとして提供します。与えられる文字列は、検証する XBRL ドキュメントのコンテンツです。

▼ **setInputTextCollection**

```
public void setInputTextCollection(Collection<?> stringCollection)
```

入力データとして使用される複数の XBRL ファイルのコンテンツを提供します。`stringCollection` パラメーターはそれぞれが入力 XBRL 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です。

フォーミュラとアサーション

XBRL フォーミュラとアサーションのオプションを指定するメソッド

▼ **addAssertionForProcessing**

```
public void addAssertionForProcessing(String assertion)
```

与えられているアサーションに、アサーションの実行を制限します。1つ以上のアサーションを指定するために複数回呼び出します。 与えられた文字列はアサーションの名前を有します。アサーション無しの処理する場合 ##none を使用し、 ##all をすべてのアサーションの処理をする場合使用します。

addAssertionSetForProcessing

```
public void addAssertionSetForProcessing(String assertionSet)
```

与えられているアサーションセットに、アサーションセットの実行を制限します。 1つ以上のアサーションセットを指定するために複数回呼び出します。アサーションセット無しの処理する場合 ##none を使用し、 ##all をすべてのアサーションセットの処理をする場合使用します。

addFormulaArrayParameter

```
public void addFormulaArrayParameter(String type, String name, Object[]
values)
```

フォーミュラ評価プロセス内で使用されている列挙パラメーターを追加します。
パラメーター:
type: 配列の値内のペアでない値のデフォルトデータ型を与える文字列。 デフォルト xs:string。
name: パラメーターの名前を与える文字列。
values: 値の配列とデータ型の値のペア
コードサンプルの詳細に関しては、XBRL フォーミュラパラメーターのセクションを参照してください。

addFormulaForProcessing

```
public void addFormulaForProcessing(String formula)
```

フォーミュラの実行を与えられたフォーミュラに制限します。1つ以上のフォーミュラを指定するために複数回呼び出します。与えられた文字列はフォーミュラの名前を有します。フォーミュラ無しの処理する場合 ##none を使用し、 ##all をすべてのフォーミュラの処理をする場合使用します。

addFormulaParameter (with NS)

```
public void addFormulaParameter(String type, String name, String value,
String namespace)
```

フォーミュラ評価プロセス内で使用されるパラメーターを追加します。
パラメーター:
type: パラメーターのデータ型を与える文字列。
name: パラメーターの名前を与える文字列。
value: パラメーターの値を与える文字列。
namespace: パラメーターの名前空間を与える文字列。
メモ:このメソッドが使用される場合、名前空間は addFormulaParameterNamespace にパスされます。

addFormulaParameter

```
public void addFormulaParameter(String type, String name, String value)
```

フォーミュラ評価プロセス内で使用されるパラメーターを追加します。
パラメーター:
type: パラメーターのデータ型を与える文字列。
name: パラメーターの名前を与える文字列。
value: パラメーターの値を与える文字列。

addFormulaParameterNamespace

```
public void addFormulaParameterNamespace(String prefix, String URI)
```
パラメーター名、型、または値の QNames 内で使用される名前空間を定義します。
パラメーター:
prefix addFormulaArrayParameter にパスされる名前空間プレフィックスの値です。
URI: 名前空間 URI。

clearFormulaParameterList

```
public void clearFormulaParameterList()
```
addFormulaParameter メソッドを使用して作成されたフォーミュラパラメーターのリストをクリアします。

readFormulaAssertions

```
public String readFormulaAssertions()
```
指定されたファイルからフォーミュラアサーションを取得します。フォーミュラアサーションを含む文字列を返します。

readFormulaOutput

```
public String readFormulaOutput()
```
指定されたファイル内のフォーミュラアサーションを評価して、結果を返します フォーミュラアサーションの評価である文字列を返します。

setFormulaAssertionsAsXML

```
public void setFormulaAssertionsAsXML(boolean enable)
```
RaptorXML+XBRL がアサーションが有効化されて作動している場合、アサーションファイルの XML フォーマットを有効化します。true の値は、XML 出力を有効化します。false の値は、JSON 出力を生成します。デフォルトは false です。
bEnable: ブール値の true または false を必要とします。

setFormulaAssertionsOutput

```
public void setFormulaAssertionsOutput(String outputFile)
```
フォーミュラアサーションの出力を含むファイルの場所を設定します。
パラメーター:
outputFile: 与えられた文字列には、出力ファイルのフルパスがあります。

setFormulaOutput

```
public void setFormulaOutput(String outputFile)
```
フォーミュラ評価の出力を含むファイルの場所を設定します。与えられた文字列には、出力ファイルのフルパスがあります。

evaluateFormula

```
public boolean evaluateFormula()
```
XBRL インスタンスファイル内の XBRL フォーミュラの評価結果を返します。結果は成功時には true 、失敗時には false です。エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには getLastErrorMessage メソッドを使用してください。
成功時にはブール値の true または失敗時には false を必要とします。

▼ **setFormulaExtensionEnabled**

`public void setFormulaExtensionEnabled(boolean enable)`
検証のための XBRL フォーミュラ拡張子を有効化します。`true` の値は、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`true` はです。

▼ **setFormulaPreloadSchemas**

`public void setFormulaPreloadSchemas(boolean enable)`
XBRL フォーミュラスキーマがプリロードされるかどうかを定義します。の値は、を有効化します。`true` は、スキーマをプリロードします。`false` はプリロードしません。デフォルト値は`false` です。

テーブル

XBRL テーブルのためのオプションを指定するメソッド。

▼ **addTableForProcessing**

`public void addTableForProcessing(String table)`
テーブルの生成を与えられたテーブルのみに制限します。 1つ以上のテーブルを指定するために複数回呼び出します。与えられた文字列はテーブル名を有します。テーブル無しの処理する場合 `##none` を使用し、`##all` テーブルの処理をする場合使用します。

▼ **generateTables**

`public boolean generateTables()`
インスタンスファイル内のXBRL テーブルを評価します。結果は成功時には`true` 、失敗時には`false` です。エラーが発生した場合、`RaptorXMLException` が挙げられます。追加情報にアクセスするには、`getLastErrorMessage` メソッドを使用してください。

▼ **setTableEliminateEmptyRows**

`public void setTableEliminateEmptyRows(boolean enable)`
HTML 出力内で、空のテーブル行列の削除を有効化します。`true` の値は、サポートを有効化します。`false` の値は無効化します。

▼ **setTableExtensionEnabled**

`public void setTableExtensionEnabled(boolean enable)`
検証のために、XBRL Table 1.0 拡張子を有効化します。`true` の値はサポートを有効化します。`false` の値は無効化します。

▼ **setTableLinkbaseNamespace**

`public void setTableLinkbaseNamespace(String namespace)`
前のドラフト仕様により書き込まれたテーブルリンクベースのロードを有効化します。与えられた`namespace` パラメーターは、テーブルリンクベースを指定します。テーブルリンクベースの検証、解決、およびレイアウトは、しかしながら、常に 2014 年 3月 18日版のTable Linkbase 1.0 勧告に従い実行されます。`##detect` を使用して自動検出を有効化します。
パラメーター:

namespace: 以下の値が認識されています :
##detect
http://xbrl.org/PWD/2013-05-17/table
http://xbrl.org/PWD/2013-08-28/table
http://xbrl.org/CR/2013-11-13/table
http://xbrl.org/PR/2013-12-18/table
http://xbrl.org/2014/table

setTableOutput

```
public void setTableOutput(String outputFile)
```

テーブル生成の出力を含むファイルの場所を設定します。与えられた文字列には、出力ファイルのフルパスがあります。

setTableOutputFormat

```
public void setTableOutputAsXML(ENUMTableOutputFormat format)
```

テーブル出力ファイルのフォーマットを設定します。フォーマットは ENUMTableOutputFormat の値になります。

setTablePreloadSchemas

```
public void setTablePreloadSchemas(boolean enable)
```

XBRL Table 1.0 仕様のスキーマのプリロードを有効化します。true の値はサポートを有効化します。false の値は無効化します。デフォルトは false です。

XML とXML スキーマ

XInclude サポートレベルとXML スキーマのためのオプションを設定するためのメソッド

setXincludeSupport

```
public void setXincludeSupport(boolean support)
```

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。

setSchemaImports

```
public void setSchemaImports(ENUMSchemaImports opt)
```

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 (Java、COM/.NET) により指定されます。

setSchemalocationHints

```
public void setSchemalocationHints(ENUMLoadSchemalocation opt)
```

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 (Java、COM/.NET) により指定されます。

setSchemaMapping

public void setSchemaMapping(*ENUMSchemaMapping* opt)
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 (Java、COM/.NET) により指定されます。 .

一般的な XBRL
ラベルとその他のXBRL パラメーターのためのオプションのメソッド

addIXBRLTransformationRegistryLimit

`public void` **addIXBRLTransformationRegistryLimit(`String limit`)**
指定されているバージョンに対して使用することのできるXBRL 変換レジストリを制限します。`limit` パラメーターは、インラインXBRL レジストリに制限されているバージョンです。

clearIXBRLTransformationRegistryLimit

`public void` **clearIXBRLTransformationRegistryLimit()**
インラインXBRL 変換レジストリのために指定されている制限を削除します。

enableIXBRLValidateTarget

`public void` **enableIXBRLValidateTarget(`boolean enable`)**
コンセプトラベルをレンダする際、優先する拡張リンクロールを指定します。ブール値のtrue または false を必要とします。デフォルトは `true` です。

setConceptLabelLinkrole

`public void` **setConceptLabelLinkrole(`String labelLinkrole`)**
コンセプトラベルをレンダする際、優先するリンクロールを指定します。与えられた文字列は優先するリンクロールを保有します

setConceptLabelRole

`public void` **setConceptLabelRole(`String labelRole`)**
コンセプトラベルをレンダする際、優先するコンセプトラベルロールを指定します。与えられた文字列は優先するラベルロールを保有します。デフォルト:`http://www.xbrl.org/2008/role/label`。

setDimensionsExtensionEnabled

`public void` **setDimensionExtensionEnabled(`boolean bEnable`)**
XBRL Dimension 拡張子の検証を有効化します。 ブール値のtrue または false を必要とします。 true の値は、サポートを有効化します。false の値は無効化します。デフォルトは `true` です。

setGenericLabelRole

`public void` **setGenericLabelRole(`String labelRole`)**
コンセプトラベルをレンダする際、優先するラベルロールを指定します。与えられた文字列は優先するラベルロールを保有します。デフォルト:`http://www.xbrl.org/2008/role/label`.

setIXBRLOutput

```
public void setIXBRLOutput(String outputFile)
```

指定されると、生成されたXBRL 出力はこのパスに書き込まれます。与えられた文字列は、出力の場所の絶対 URL を保持しています。

setIXBRLTreatAsDocumentSet

```
public void setIXBRLTreatAsDocumentSet(boolean enable)
```

すべての入力を単一のインラインXBRL ドキュメントセットとして扱います。ブール値のtrue または false を必要とします。デフォルトは `false` です。

setIXBRLUriTransformationStrategy

```
public void setIXBRLUriTransformationStrategy(ENUMIXBRLUriStrategy strategy)
```

生成されたXBRL ドキュメント内で、インラインXBRL URI がどのように変換されるかを指定します。選択された列挙は後のストラテジーを指定します。デフォルトの列挙は `eStrategyKeepRelative` です。

setIXBRLVersion

```
public void setIXBRLVersion(ENUMIXBRLVersion version)
```

使用するインラインXBRL バージョンを指定します。選択された列挙は使用されるバージョンを指定します。デフォルトの列挙は `eVersionDetect` です。

setLabelLang

```
public void setLabelLang(String labelLang)
```

レンダリングラベルを使用する時に優先する言語を指定します。与えられた文字列は優先されるラベル言語を保有します。デフォルト：`en`。

setPreloadSchemas

```
public void setPreloadSchemas(boolean preload)
```

XBRL 2.1 スキーマがプリロードされるかを定義します。ブール値のtrue または false を必要とします。`true` の値は、プリロードを有効化します。 `false` の値は無効化します。デフォルトは `true` です。

setTreatXBRLInconsistenciesAsErrors

```
public void setTreatXBRLInconsistenciesAsErrors(boolean treat)
```

XBRL 2.1 仕様の定義に対する不整合がファイルに含まれる場合、`true` の値は、XBRL 検証の失敗を引き起こします。デフォルトは `false` です。 `false` の場合、XBRL 2.1 仕様に従い生じるXBRL 矛盾はエラーとして扱われません。

6.2.6 RaptorXMLException

```
public interface RaptorXMLException
```

例外を生成する単一メソッドを有します。

RaptorXMLException
```
public void RaptorXMLException(String message)
```
処理中に発生するエラーについての情報を含む例外を生成します。
パラメーター:
message: エラーに関する情報を提供する文字列。

6.3 RaptorXML Enumerations for Java

次の列挙が定義されています。これらは機能別にグループ化されており、サブセクションで説明されています。

RaptorXMLFactory
XML 検証
XSLT とXQuery
XBRL

6.3.1 RaptorXMLFactory

RaptorXMLFactory インターフェイスは次の列挙を定義します。

ENUMErrorFormat

```
public enum ENUMErrorFormat {
        eFormatText
        eFormatShortXML
        eFormatLongXML }
```

ENUMErrorFormat は、以下のうちの1つの列挙リテラルをとります：eFormatText, eFormatShortXML, eFormatLongXML。これらは、エラーメッセージのフォーマットを設定します。eLongXML は最も詳細な説明を提供します。デフォルト：eFormatText.

使用（インターフェイス：メソッド）：
RaptorXMLFactory setErrorFormat

6.3.2 XML 検証

XMLValidator インターフェイスは次の列挙を定義します。

▼ **ENUMAssessmentMode**

```
public enum ENUMAssessmentMode {
        eAssessmentModeLax
        eAssessmentModeStrict    }
```

ENUMAssessmentMode は、以下のいずれかの列挙リテラルをとります：eAssessmentModeLax, eAssessmentModeStrict。これらは、検証がlax または strict になるかを設定します。

使用（インターフェイス：メソッド）：
XMLValidator setAssessmentMode

▼ **ENUMLoadSchemaLocation**

```
public enum ENUMLoadSchemalocation {
        eLoadBySchemalocation
        eLoadByNamespace
        eLoadCombiningBoth
        eLoadIgnore    }
```

ENUMLoadSchemalocation どの様にスキーマの場所が決定されるかどうかを示す列挙リテラルを含みます。XML または XBRL 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により選択が決定されます。属性は以下であることができます：xsi:schemaLocation または xsi:noNamespaceSchemaLocation。

- eLoadBySchemalocation XML または XBRL インスタンスドキュメント内のスキーマロケーション属性の URL を使用します。この列挙リテラルは**デフォルトの値**です。
- eLoadByNamespace xsi:noNamespaceSchemaLocation がカタログマッピングを使用してスキーマをロケートする場合、xsi:schemaLocation の名前空間の部分とからの文字列を使用します。
- eLoadCombiningBoth：名前空間 URL または schemaLocation URL の1つがカタログ 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。 両方がカタログ マッピングを有する場合は、ENUMSchemaMapping の値がどちらのマッピングを使用されるかを決定します。namespace および schemaLocation URL の双方がカタログ マッピング、を有さない場合、schemaLocation URL が使用されます
- eLoadCombiningBoth：xsi:schemaLocation と xsi:noNamespaceSchemaLocation 属性は両方とも無視されます。

使用（インターフェイス：メソッド）：

XMLValidator	setSchemalocationHints
XSLT	setSchemalocationHints
XBRL	setSchemalocationHints

▼ **ENUMSchemaImports**

```
public enum ENUMSchemaImports {
        eSILoadBySchemalocation
        eSILoadPreferringSchemalocation
        eSILoadByNamespace
        eSILoadCombiningBoth
        eSILicenseNamespaceOnly    }
```

ENUMSchemaImports は、それぞれが任意の namespace 属性と任意の schemaLocation 属性を持つ、スキーマの 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。

- eSILoadBySchemalocation は、カタログ マッピングを考慮し、schemaLocation 属性の値を使用して、スキーマをロケートします。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: 属性が存在する場合、カタログ マッピングを考慮して、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 マッピングを介して使用されます。この列挙リテラルは、デフォルトの値です。
- eSILoadByNamespace は、カタログ マッピングを介して、スキーマを検索するために namespace 属性の値を使用します。
- eSILoadCombiningBoth: namespace URL または schemaLocation URL の1つがカタログ 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。両方がカタログ マッピングを有する場合は、ENUMSchemaMapping の値がどちらのマッピングを使用されるかを決定します。namespace および schemaLocation URL の双方がカタログ マッピング、を有さない場合、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。

使用 (インターフェイス:メソッド):

XMLValidator	setSchemaImports
XSLT	setSchemaImports
XBRL	setSchemaImports

ENUMSchemaMapping

```
public enum ENUMSchemaMapping {
       eSMPreferSchemalocation
       eSMPreferNamespace    }
```

ENUMSchemaMapping は、名前空間またはスキーマロケーションが選択されるかを指定する列挙リテラルを含みます。 .

- eSMPreferNamespace: 名前空間を選択します。
- eSMPreferSchemalocation: は、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。

使用 (インターフェイス:メソッド):

XMLValidator	setSchemaMapping
XSLT	setSchemaMapping
XBRL	setSchemaMapping

ENUMXMLValidationMode

```
public enum ENUMXMLValidationMode {
       eProcessingModeValid
       eProcessingModeWF   }
```

ENUMXMLValidationMode は、実行するXML 検証の型 (検証または整形式チェック)を指定する列挙リテラルを含みます)。

- eProcessingModeValid: XML 処理モードを validation に設定します。
- 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。

使用 (インターフェイス:メソッド):
XMLValidator setXMLValidationMode

XSLT setXMLValidationMode
XQuery setXMLValidationMode

ENUMXMLValidationType

```
public enum ENUMValidationType {
       eValidateAny
       eValidateXMLWithDTD
       eValidateXMLWithXSD
       eValidateDTD
       eValidateXSD
       eValidateJSON
       eValidateJSONSchema
       eValidateAvro
       eValidateAvroSchema
       eValidateAvroJSON }
```

ENUMValidationType は、実行する検証を指定する、また、XML ドキュメントの場合、DTD または XSD のどちらに対して検証を行うか指定する列挙リテラルを含みます。

- eValidateAny: ドキュメントの型は自動的に検出されます。
- eValidateXMLWithDTD: XML ドキュメントを DTD に対して検証します。
- eValidateXMLWithXSD: XSD に対して XML ドキュメントを検証します (XML スキーマ)。
- eValidateDTD: DTD ドキュメントを検証します。
- eValidateXSD: XSD ドキュメントを検証します。
- eValidateJSON: JSON インスタンスドキュメントを検証します。
- eValidateJSONSchema: JSON スキーマドキュメントを検証します。
- eValidateAvro: Avro バイナリファイルを検証します。バイナリファイル内の Avro データをバイナリファイル内の Avro スキーマに対して検証します。
- eValidateAvroSchema: Avro スキーマ仕様に対して Avro スキーマを検証します。
- eValidateAvroJSON: JSON-シリアル化された Avro データファイルを Avro スキーマに対して検証します。

使用 (インターフェイス:メソッド):
XMLValidator isValid

ENUMWellFormedCheckType

```
public enum ENUMWellformedCheckType {
       eWellformedAny
       eWellformedXML
       eWellformedDTD
       eWellformedJSON    }
```

ENUMWellformedCheckType は、全ての型に対しての整形式のチェックを設定します。XML または DTD の 2 つのうちどちらの型かを自動的に検出した後、ドキュメントの整形式をチェックします。

- eWellformedAny: ドキュメントの型は自動的に検出されます。
- eWellformedXML: XML の型に対しての整形式のチェックを設定します。
- eWellformedDTD: DTD の型に対しての整形式のチェックを設定します。

- eWellformedJSON: JSON の型に対しての整形式のチェックを設定します。

使用 (インターフェイス:メソッド):
XMLValidator isWellformed

ENUMXSDVersion

```
public enum ENUMXSDVersion {
       eXSDVersionAuto
       eXSDVersion10
       eXSDVersion11    }
```

ENUMXSDVersion は、検証に使用するXML スキーマバージョンを示す以下の列挙リテラルを含みます。

- 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。

使用 (インターフェイス:メソッド):
XMLValidator setXSDVersion
XSLT setXSDVersion
XQuery setXSDVersion

6.3.3 XSLT と XQuery

XSLT インターフェイスは次の列挙を定義します。

▼ **ENUMXSLTVersion**

```
public enum ENUMXSLTVersion {
        eVersion10
        eVersion20
        eVersion30 }
```

ENUMXSLTVersion は、以下のいずれかの列挙リテラルを保有します。:eVersion10, eVersion20, eVersion30 。これらは、処理のために使用されるXSLT バージョンを設定します (検証、または、XSLT 変換)。

使用 (インターフェイス:メソッド):
XSLT setVersion

XQuery インターフェイスは次の列挙を定義します。

▼ **ENUMXQueryVersion**

```
public enum ENUMXQueryVersion {
        eVersion10
        eVersion30
        eVersion31 }
```

ENUMXQueryVersion は、以下のいずれかの列挙リテラルをとります :eVersion10, eVersion30, eVersion31。これらは、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)。

使用 (インターフェイス:メソッド):
XQuery setVersion

▼ **ENUMXQueryUpdatedXML**

```
public enum ENUMXQueryUpdatedXML {
        eUpdatedDiscard
        eUpdatedWriteback
        eeUpdatedAsMainResult }
```

ENUMXQueryVersion は、以下のいずれかの列挙リテラルをとります :

- 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、setInputXMLFileName と共に指定されます。
- eUpdatedAsMainResult: アップデートは、ExecuteUpdate 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。

使用 (インターフェイス:メソッド):
XQuery setUpdatedXMLWriteMode

6.3.4 XBRL

XBRL インターフェイスは次の列挙を定義します。

▼ **ENUMValidationType**

```
public enum ENUMValidationType {
        eValidateAny
        eValidateInstance
        eValidateTaxonomy
        eValidateInline
        eValidateTaxonomyPackage }
```

ENUMValidationType は、実行する検証を指定する列挙リテラルを含みます。また、XML ドキュメントの場合、DTD または XSD のどちらに対して検証を行うか指定する列挙リテラルを含みます。

- eValidateAny: ドキュメントの型は自動的に検出されます。
- eValidateInstance: XBRL インスタンスドキュメントを検証します (.xbrl ファイル拡張子)。
- eValidateTaxonomy: VXBRL タクソノミを検証します (.xsd ファイル拡張子)。
- eValidateInline: インライン XBRL (iXBRL) ドキュメントをインライン XBRL 1.0 または、インライン XBRL 1.1 仕様に対して検証します。 .
- eValidateTaxonomyPackage: XBRL タクソノミパッケージをタクソノミ 1.0 パッケージ仕様に従い検証します。

使用 (インターフェイス:メソッド):

XBRL	isValid

▼ **ENUMTableOutputFormat**

```
public enum ENUMTableOutputFormat {
        eFormatXML
        eFormatHTML }
```

ENUMTableOutputFormat 生成されたテーブルを含むドキュメントの出力フォーマットを指定するエミュレーション列挙リテラルを含みます。

- eFormatXML: 生成されたテーブルを持つ出力ドキュメントは XML フォーマットです。
- eFormatHTML: 生成されたテーブルを持つ出力ドキュメントは HTML フォーマットです。

使用 (インターフェイス:メソッド):

XBRL	setTableOutputFormat

▼ **ENUMIXBRLVersion**

```
public enum ENUMIXBRLVersion {
        eVersion10
        eVersion11
        eVersionDetect }
```

ENUMIXBRLVersion 使用するインライン XBRL 使用のバージョンを指定する列挙リテラルを含んでいます:インライン XBRL 1.0 または、インライン XBRL 1.1. eVersionDetect は自動検出を有効化します。

使用 (インターフェイス:メソッド):

XBRL	isValid

ENUMIXBRLUriStrategy

```
public enum ENUMIXBRLUriStrategy {
       eStrategyNone
       eStrategyMakeAbsolute
       eStrategyMakeRelative
       eStrategyKeepRelative }
```

ENUMIXBRLUriStrategy は、生成されたXBRL ドキュメント内で、インラインXBRL URI がどのように変換されるかを指定します。

- eStrategyNone: URI verbatim をターゲットドキュメントにコピーします。
- eStrategyMakeAbsolute: 入力ドキュメント内の対応する要素でインスコープのベースURI に対して解決することにより相対的なURI を絶対的 URI にします。
- eStrategyMakeRelative: 可能な場合、絶対、および 相対 URI を出力ドキュメントに対して相対的にします (それ以外の場合、解決された絶対 URI を書き込みます)。
- eStrategyKeepRelative: 可能な場合、相対的なURI のみを出力ドキュメントに対して相対的にします (そして絶対的 URI をコピーします)。

使用 (インターフェイス:メソッド):

XBRL	setIXBRLUriTransformationStrategy

チャプター 7

COM と .NET インターフェイス

7 COM と .NET インターフェイス

2 つのインターフェイスと 1 つのAPI

RaptorXML+XBRL Server のCOM および .NET インターフェイスは単一のAPI を使用します :RaptorXML +XBRL Server のCOM/.NET API。 .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構築されています。

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

このセクションの構成

このセクションは以下のように整理されています :

- COM インターフェイスについて は、COM インターフェイスの作動について、およびCOM インターフェイスとの作業に必要なステップについて説明します。
- .NET インターフェイスについて は、 .NET インターフェイスと作業するための環境の設定方法を説明します。
- プログラム言語 は、RaptorXML 機能の呼び出し方法を表示する、よく使用されているプログラム言語でのコードのリストを提供します
- API レファレンス は、API のオブジェクトモデル、オブジェクト、プロパティをドキュメントする、します。

7.1 COM インターフェイスについて

RaptorXML+XBRL Server がインストールされると、RaptorXML+XBRL 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+XBRL 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+XBRL 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

COM インターフェイスを介して、どのようにRaptorXML API を使用するかを表示するVBScript サンプルは、プログラム言語 のセクションにリストされています。これら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` フォルダーで使用することができます。

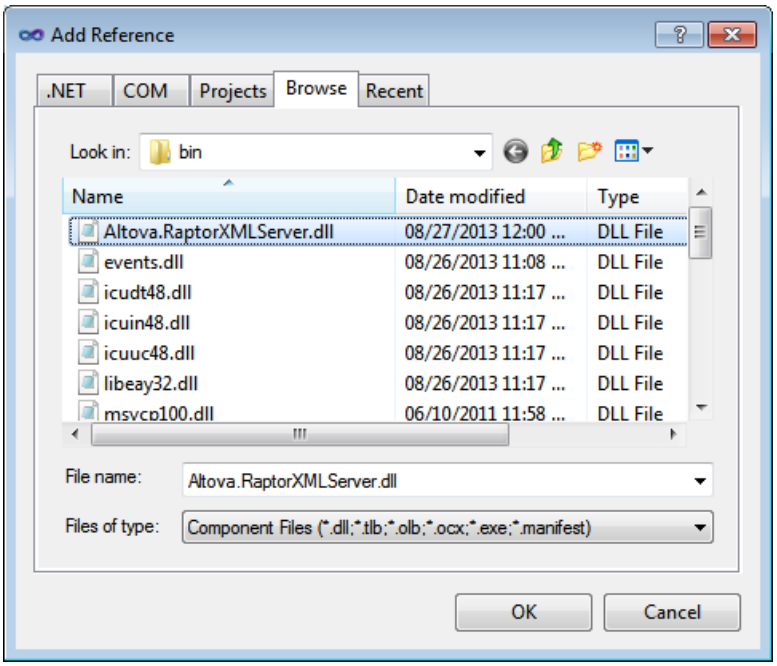
7.2 .NET インターフェイスについて

.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れます。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+XBRL 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

.NET インターフェイスを介して使用することのできる RaptorXML API の C# サンプル および Visual Basic .NET サンプル は、プログラム言語のセクションにリストされています。これらのリスト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/API` フォルダーで使用することができます。

7.3 プログラム言語

プログラム言語は、COM および .NET へアクセスをサポートする方法とは異なります。最もよく使用される言語（下のリンク参照）のいくつかのサンプルは使用開始の手助けとなるでしょう。このセクションのコードのリストはアクセスすることのできる基本的な機能を表示しています。基本的な機能は、RaptorXML+XBRL 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。

VBScript

VBScript は、RaptorXML+XBRL Server の COM API にアクセスするために使用することができます。VBScript リストは次の基本的な機能の使用例を示します：

- RaptorXML+XBRL Server COM API に接続する
- XML ファイルの検証
- XSL 変換を実行する
- XQuery を実行する

C#

C# は、RaptorXML+XBRL Server の .NET API にアクセスするために使用することができます。C# コードリストは、以下の基本機能のためのへAPI のアクセス方法を表示しています：

- RaptorXML+XBRL Server .NET API に接続する
- XML ファイルの検証
- XSL 変換を実行する
- XQuery を実行する

Visual Basic .NET

Visual Basic.NET は、C# と比較して構文のみが異なります。.NET API は同様の作動を行います。Visual Basic コードリストは以下のオペレーションについて説明しています：

- RaptorXML+XBRL Server .NET API に接続する
- XML ファイルの検証
- XSL 変換を実行する
- XQuery を実行する

このセクションには以下の コードサンプルが含まれています：

COM インターフェイス

- VBScript での例

.NET インターフェイス

- C# での例
- Visual Basic での例

7.3.1 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
- XML ファイルの検証
- XSLT 変換を実行し、結果を文字列として返す
- XQuery ドキュメントの処理し、結果をファイルに保存する
- 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

```
'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)
	objRaptorErrorLimit = 1
	objRaptorReportOptionalWarnings = true
	objRaptorServerName = "::1"
	objRaptorServerPort = 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 シリアル化 (optional - for fine-tuning the result's formatting)
        objXQ.OutputEncoding = "UTF8"
        objXQ.OutputIndent = true
        objXQ.OutputMethod = "xml"
        objXQ.OutputOmitXMLDeclaration = false

        'Run the query; the result will be serialized to the given path
        call objXQ.Execute("MyQueryResult.xml")
end sub
```

```
'Perform all sample functions
sub main
        Init
        ValidateXML
        RunXSLT
        RunXQuery
end sub

'Script entry point; run the main function
main
```

7.3.2 .NET サンプル :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
- XML ファイルの検証
- XSLT 変換を実行し、結果を文字列として返す
- XQuery ドキュメントの処理し、結果をファイルに保存する
- 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

```
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());
      }
    }
  }
}
```

7.3.3 .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
- XML ファイルの検証
- XSLT 変換を実行し、結果を文字列として返す
- XQuery ドキュメントの処理し、結果をファイルに保存する
- 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

```
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
```

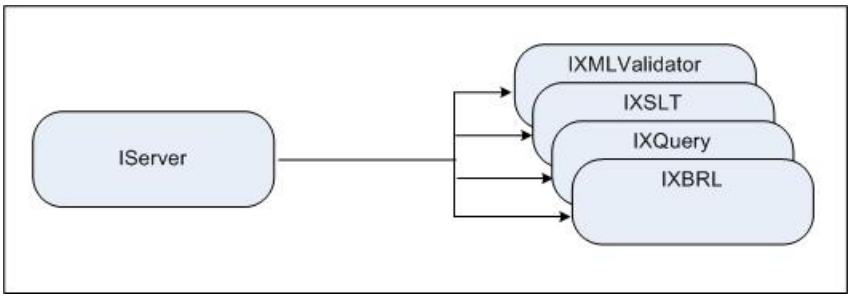
```
  'Perform a transformation; return the result as a string
   Sub RunXSLT()

    '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
```

7.4 API レファレンス

このセクションではAPI 仕様について説明されます :オブジェクトモデルと、インターフェイスと列挙の詳細。

RaptorXML の機能を使用する最初の点は、 `IServer` インターフェイスです。このオブジェクトは、RaptorXML の以下の機能を与えるオブジェクトを含みます :XML 検証、XBRL 検証、XSLT 変換およびXQuery ドキュメントの処理。RaptorXML のオブジェクトモデルのAPI は、次の図で示されています。



オブジェクトモデルの階層構造は下に表示されており、対応するセクションでインターフェイスについての詳細が説明されています。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
  |-- IXMLValidator
  |-- IXSLT
  |-- IXQuery
  |-- IXBRL
```

7.4.1 インターフェイス

次のインターフェイスが定義されています。詳細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。

IServer
IXMLValidator
IXSLT
IXQuery
IXBRL

IServer

IServer インターフェイスは、対応するRaptorXML エンジンのインターフェイスを返すメソッドを提供します :XML バリデーター、XBRL、XSLT とXQuery プロパティは、インターフェイスのパラメーターを定義します。

メソッド

IServer インターフェイスのメソッドは、対応するRaptorXML エンジンのインターフェイスを返します :XML バリデーター、XBRL、XSLT とXQuery。

▼ **GetXMLValidator**

IXMLValidator **GetXMLValidator()**
XML 検証エンジンのインスタンスを返します。

▼ **GetXBRL**

IXBRL **GetXBRL()**
XBRL エンジンのインスタンスを返します。

▼ **GetXSLT**

IXSLT **GetXSLT()**
XSLT エンジンのインスタンスを返します。

▼ **GetXQuery**

IXQuery **GetXQuery()**
XQuery エンジンのインスタンスを返します。

プロパティ

IServer インターフェイスのプロパティは、下にアルファベット順に説明されています。テーブルはプロパティを簡単参照のためグループ化して表示しています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

▼ **APIMajorVersion**

`int` **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と異なる場合があります

▼ **APIMinorVersion**

`int` **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と異なる場合があります。

▼ **APIServicePackVersion**

`int` **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と異なる場合があります。

▼ **ErrorFormat**

ENUMErrorFormat **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)。

▼ **ErrorLimit**

`int` **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。

▼ **GlobalCatalog**

`string` **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。

▼ **GlobalResourceConfig**

`string` **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。

▼ **GlobalResourceFile**

`string` **GlobalResourceFile**
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。

▼ **MajorVersion**

`int` **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+XBRL 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** が挙げられます (*Java*)。

▼ **MinorVersion**

`int` **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+XBRL Server 2017r2sp1 (x64)` に対しては、 (マイナーバージョンの `r2` から) が返されます。エラーが発生すると、

RaptorXMLException が挙げられます (*Java インターフェイスの詳細*)。

▼ **ProductName**

`string` **ProductName**
製品の名前を文字列として返します。 *サンプル*: `Altova RaptorXML+XBRL Server 2017r2sp1(x64)` に対しては、`Altova RaptorXML+XBRL Server` が返されます。エラーが発生すると、**RaptorXMLException** が挙げられます (*Java インターフェイスの詳細*)。

▼ **ProductNameAndVersion**

`string` **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+XBRL Server 2017r2sp1(x64)` に対しては、`Altova RaptorXML+XBRL Server 2017r2sp1(x64)` が返されます。エラーが発生すると、**RaptorXMLException** が挙げられます (*Java インターフェイスの詳細*)。

▼ **ReportOptionalWarnings**

`bool` **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。

▼ **ServerName**

`string` **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** が挙げられます (*Java インターフェイスの詳細*)。入力パラメーターは、HTTP サーバー名を与える文字列です。

▼ **ServerPath**

`string` **ServerPath**
URL のフォームで、HTTP サーバーへのパスを指定します。 エラーが生じた場合は、`RaptorXMLException` が挙げられます。

▼ **ServerPort**

`int` **ServerPort**
HTTP サーバーのサーバーポートを指定します。型は `ushort` です。エラーが生じた場合は、`RaptorXMLException` が挙げられます。

▼ **ServicePackVersion**

`int` **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+XBRL Server 2017r2sp1(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** が挙げられます (*Java インターフェイスの詳細*)。

▼ **UserCatalog**

`string` **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。

IXMLValidator

メソッド

IXMLValidator インターフェイスには次の *メソッド* が存在します。操作の成功または失敗に従い、`True` または `False` を返します。

ExtractAvroSchema

bool ExtractAvroSchema(string **outputPath**)
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** が挙げられます (*Java インターフェイスに移動*)。getLastErrorMessage (***Java 内で***)、または、LastErrorMessage (***COM/.NET 内で***) を追加情報にアクセスするために使用します。

IsValid

bool IsValid(ENUMValidationType **type**)
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 リテラル (Java、COM/.NET) を値として取る、type パラメータ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、a **RaptorXMLException** が挙げられます (*Java インターフェイスに移動*)。getLastErrorMessage (***Java 内で***)、または、LastErrorMessage (***COM/.NET 内で***) を追加情報にアクセスするために使用します。

IsWellFormed

bool IsWellFormed(ENUMWellformedCheckType **type**)
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 を取るリテラル (Java、COM/.NET)。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、a **RaptorXMLException** が挙げられます (*Java インターフェイスに移動*)。getLastErrorMessage (***Java 内で***)、または、LastErrorMessage (***COM/.NET 内で***) を追加情報にアクセスするために使用します。

プロパティ

IXMLValidator インターフェイスのプロパティは、下にアルファベット順に説明されています。テーブルはプロパティを簡単参照のためグループ化して表示しています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

AssessmentMode

ENUMAssessmentMode **AssessmentMode**
ENUMAssessmentMode リテラル (Java、COM/.NET) を取る mode パラメータ内で定義される、XML 検証の評価モード (Strict/Lax) を設定します。

AvroSchemaFileName

string 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。

AvroSchemaFromText

```
string 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。

DTDFileName

```
string 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。

DTDFromText

```
string 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。

EnableNamespaces

```
bool 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。

InputFileArray

```
object 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。

InputFileCollection

```
string InputFileCollection
```

入力データとして使用されるXML 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。それぞれが入力 XML ファイルの絶対 URL である、文字列のコレクション。

InputFileName

```
string 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。

InputFromText

```
string 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。

InputTextArray

object 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。

▼ **InputTextCollection**

string InputTextCollection
入力データとして使用される複数の XML ファイルを提供します。それぞれが入力 XML ファイルのコンテンツである、文字列のコレクション。

▼ **InputXMLFileCollection (DEPRECATED. Use InputFileCollection instead.)**

string InputXMLFileCollection
入力データとして使用される XML ファイルのコレクションが提要されます。ファイルは URL により識別されます。それぞれが入力 XML ファイルの絶対 URL である文字列のコレクションです。

▼ **InputXMLFileName (DEPRECATED. Use InputFileName instead.)**

string 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。

▼ **InputXMLFromText (DEPRECATED. Use InputFromText instead.)**

string 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。

▼ **InputXMLTextCollection(DEPRECATED. Use InputTextCollection instead.)**

string InputXMLTextCollection
入力データとして使用される複数の XML ファイルを提供します。それぞれが XML ファイルのコンテンツである文字列のコレクションです。

▼ **LastErrorMessage**

string 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。

▼ **ParallelAssessment**

bool ParallelAssessment
パラレルスキーマの有効性の評価を有効化/無効化します。

▼ **PythonScriptFile**

string PythonScriptFile
Python スクリプトファイルのロケーションを設定します。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。

▼ **SchemaFileArray**

object **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。

SchemaFileName

string SchemaFileName(*String* **filePath)**
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。

SchemaFromText

string 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。

SchemaImports

ENUMSchemaImports **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 (Java、COM/.NET) により指定されます。

SchemalocationHints

ENUMLoadSchemalocation **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 (Java、COM/.NET) により指定されます。

SchemaMapping

ENUMSchemaMapping **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 (Java、COM/.NET) により指定されます。 .

SchemaTextArray

object 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。

Streaming

bool 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。

XIncludeSupport

bool 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。

▼ **XMLValidationMode**

ENUMXMLValidationMode **XMLValidationMode**
有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode (Java、COM/.NET) の列挙リテラルであるXML 検証モードを設定します。

▼ **XSDVersion**

ENUMXSDVersion **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 (Java、COM/.NET) は、の列挙リテラルです。

IXSLT

IXSLT インターフェイスは、XSLT 1.0、XSLT 2.0、またはXSLT 3.0 変換を実行するためのメソッドとプロパティを提供します。結果はファイルに保存または文字列として返されます。インターフェイスは、XSLT パラメーターのXSLT スタイルシートへのパスを有効化します。XML のURL とXSLT ファイルは、URL のインターフェイスのプロパティを介して文字列として与えられることができます。または、XML とXSLT ドキュメントは、コード内でテキスト文字列として構築されることができます。

メモ: 文字列の入力がURL と解釈される箇所では、絶対パスが使用されるべきです。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

メモ: RaptorXML のXSLT 2.0 および 3.0 エンジンを、XSLT 1.0 スタイルシートを処理するために、下位互換性を保ち使用することができます。しかしながら、出力は、同じXSLT 1.0 スタイルシートをXSLT 1.0 エンジンで処理した結果と異なる場合があります。

メソッド

IXSLT インターフェイスのメソッドは下に説明されています。文字列入力がURL として解釈されるには、絶対パスが提供される必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

▼ **IsValid**

```
bool IsValid()
```

- ENUMXSLTVersion で挙げられる、XQuery 仕様に従いXQuery ドキュメントの検証の結果を返します。(EngineVersion プロパティを参照してください)。結果は、成功時にはtrue 失敗時にはfalse です。
- エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

▼ **Execute**

```
bool Execute(string outputPath)
```

- ENUMXSLTVersion で挙げられているXQuery 仕様に従いXQuery を実行します。(EngineVersion プロパティを参照してください)。出力ファイルに結果を保存します。
- 出力ファイルは出力ファイルのURL を提供する文字列である、outputPath により定義されています。
- 結果は、成功時にはtrue 失敗時にはfalse です。

- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには LastErrorMessage オペレーションを使用してください。.

ExecuteAndGetResultAsString

```
bool Execute()
```

- ENUMXSLTVersion 内で名前が挙げられているXSLT 仕様に従い、XSLT 変換を実行します。(EngineVersion プロパティを参照してください)。そして、変換結果を文字列としてベースURI (文字列 bstrBaseURI) により定義されている場所で返します。
- このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。追加出力ファイルが必要な場合は、Execute メソッドを使用してください。
- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには LastErrorMessage オペレーションを使用してください。

ExecuteAndGetResultAsStringWithBaseOutputURI

```
bool Execute(string baseURI)
```

- ENUMXSLTVersion 内で名前が挙げられているXSLT 仕様に従い、XSLT 変換を実行します。(EngineVersion プロパティを参照してください)。そして、baseURI により定義されている箇所の文字列を変換結果として返します。
- このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。追加出力ファイルが必要な場合は、Execute メソッドを使用してください。
- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには LastErrorMessage オペレーションを使用してください。

AddExternalParameter

```
void AddExternalParameter(string paramName, string paramValue)
```

- 外部パラメーターの名前と値の追加 :paramName とparamValue は文字列です。
- 各外部パラメーターとその値は、メソッドの個別の呼び出しで指定されます。変数は、オプションで型の宣言と共に、XSLT ドキュメント内で宣言されている必要があります。XSLT ドキュメント内の型宣言に関わらず、パラメーターが AddExternalParameter と与えられている場合、特別なデリミターは必要ありません。

ClearExternalParameterList

```
void ClearExternalParameterList()
```

- AddExternalParameter メソッドと共に作成された外部パラメーターリストをクリアします。

プロパティ

IXSLT インターフェイスのプロパティは、下にアルファベット順に説明されています。テーブルはプロパティを簡単参照のためグループ化して表示しています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

AdditionalOutputs

```
string AdditionalOutputs
```

最後に実行されたジョブの追加出力を返します。

ChartExtensionsEnabled

```
bool ChartsExtensionsEnabled
```

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。

DotNetExtensionsEnabled

```
bool 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。

EngineVersion

```
ENUMXSLTVersion EngineVersion
```

使用するXSLT バージョン(1.0、2.0、または 3.0)を指定します。プロパティの値は`ENUMXSLTVersion` リテラルです。

IndentCharacters

```
string 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。

InitialTemplateMode

```
string InitialTemplateMode
```

XSLT 処理の初期モードを設定します。モード値が与えられた文字列と等価のテンプレートを処理します。.

InputXMLFileName

```
string 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。

InputXMLFromText

```
string 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。

JavaBarcodeExtensionLocation

```
string 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。

JavaExtensionsEnabled

```
bool 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。

LastErrorMessage

```
string 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。

LoadXMLWithPSVI

```
bool 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。

MainOutput

```
string MainOutput
```

最後に実行されたジョブのメイン出力を返します。

NamedTemplateEntryPoint

```
string NamedTemplateEntryPoint
```

変換のエントリポイントとして使用される名前を付けられたテンプレートの名前を文字列として指定します。

SchemaImports

```
ENUMSchemaImports 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 (Java、COM/.NET) により指定されます。

SchemalocationHints

```
ENUMLoadSchemalocation 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 (Java、COM/.NET) により指定されます。

SchemaMapping

```
ENUMSchemaMapping 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 (Java、COM/.NET) により指定されます。 .

StreamingSerialization

```
bool StreamingSerialization
```

ストリーミング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、ストリーミングシリアル化を有効化します。`false` の値は無効化します。

XincludeSupport

```
bool 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。

▼ **XMLValidationErrorAsWarning**

bool XMLValidationErrorAsWarning
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。

▼ **XMLValidationMode**

***ENUMXMLValidationMode* XMLValidationMode**
有効性または整形式をチェックするかを決定する、ENUMXMLValidationMode (Java、COM/.NET) の列挙リテラルである XML 検証モードを設定します。

▼ **XSDVersion**

***ENUMXSDVersion* 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 (Java、COM/.NET) は、の列挙リテラルです。

▼ **XSLFileName**

string XSLFileName
変換に使用される XSLT ファイルを指定します。提供される文字列は、使用される XSLT の場所を与える絶対 URL である必要があります。

▼ **XSLFromText**

string XSLFromText
テキスト文字列として、変換で使用される XSLT ドキュメントのコンテンツを提供します。

IXQuery

IXQuery インターフェイスは、XQuery 1.0 または XQuery 3.0 ドキュメントを実行するためのメソッドとプロパティを提供します。結果はファイルに保存または文字列として返されます。インターフェイスは、外部 XQuery 変数の XQuery ドキュメントへのパスを有効化します。XQuery および XML ファイルの URL のインターフェイスのプロパティを介して文字列として与えられることができます。また XML および XQuery ドキュメントは、コード内でテキスト文字列として構築されることができます。

メモ: 文字列の入力が URL と解釈される箇所では、絶対パスが使用されるべきです。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

メソッド

IXQuery インターフェイスのメソッドは下に説明されています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

▼ **IsValid**

```
bool IsValid()
```

- ENUMXQueryVersion で挙げられる、XQuery 仕様に従いXQuery ドキュメントの検証の結果を返します。(EngineVersion プロパティを参照してください)。結果は、成功時にはtrue 、失敗時にはfalse です。
- エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

IsValidUpdate

```
bool IsValidUpdate()
```

- ENUMXQueryVersion で挙げられる、XQuery 仕様に従いXQuery Update ドキュメントの検証の結果を返します。(EngineVersion プロパティを参照してください)。結果は、成功時にはtrue 、失敗時にはfalse です。
- エラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

Execute

```
bool Execute(string outputFile)
```

- ENUMXQueryVersionで挙げられているXQuery 仕様に従いXQuery を実行します。(EngineVersion プロパティを参照してください)。 出力ファイルに結果を保存します。
- 出力ファイルは出力ファイルのURL を提供する文字列である、bstrOutputFile により定義されています。
- 成功時にはブール値のtrue が、失敗時にはfalse が返されます。
- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。 追加情報にアクセスするにはLastErrorMessage オペレーション、を使用してください。

ExecuteAndGetResultAsString

```
string ExecuteAndGetResultAsString()
```

- ENUMXQueryVersion で挙げられているXQuery Update 仕様に従いXQuery Update を実行します。(EngineVersion プロパティを参照してください)。 出力ファイルに結果を保存します。
- 出力ファイルのURL を与える文字列であるoutputFile により定義されます。
- 成功時にはtrue が、失敗時にはfalse が返されます。
- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

ExecuteUpdate

```
bool ExecuteUpdate(string outputFile)
```

- ENUMXQueryVersion で挙げられているXQuery Update 仕様に従いXQuery Update 変換を実行し、変換の結果を文字列として返します。(EngineVersion プロパティを参照してください)。変換の結果を文字列として返します。
- 出力ファイルのURL を与える文字列であるoutputFile により定義されます。
- 成功時にはtrue が、失敗時にはfalse が返されます。
- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessageオペレーションを使用してください。

ExecuteUpdateAndGetResultAsString

```
string ExecuteAndGetResultAsString()
```

- ENUMXQueryVersion で挙げられているXQuery Update 仕様に従いXQuery アップデートを実行し変換の結果を文字列として返します。(EngineVersion プロパティを参照してください)。
- このメソッドは、チャートまたはセカンダリの結果などの追加結果ファイルを作成しません。追加出力ファイルが必要な場合は、Execute メソッドを使用してください。
- 変換中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

AddExternalVariable

```
void AddExternalVariable(string varName, string varValue)
```

- 外部変数の名前と値を追加します :bstrName とbstrValue は文字列です。
- 各外部変数とその値は、メソッドの個別の呼び出し内で指定されなければなりません。変数は、オプションで型の宣言と共に、XQuery ドキュメント内で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んでください。

ClearExternalVariableList

```
void ClearExternalVariableList()
```

- AddExternalVariable メソッドを使用して作成された外部変数リストをクリアします。

プロパティ

IXQuery インターフェイスのプロパティは、下にアルファベット順に説明されています。テーブルはプロパティを簡単参照のためグループ化して表示しています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

ChartExtensionsEnabled

```
bool ChartsExtensionsEnabled
```

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。

DotNetExtensionsEnabled

```
bool 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。

EngineVersion

```
ENUMXQueryVersion EngineVersion
```

使用されるXQuery バージョンを指定します。(1.0 または 3.0)。プロパティの値は ENUMXQueryVersion リテラルです。

IndentCharacters

```
string 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。

InputXMLFileName

```
string 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。

InputXMLFromText

```
string 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。

JavaBarcodeExtensionLocation

```
string 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。

JavaExtensionsEnabled

```
bool 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。

KeepFormatting

```
bool KeepFormatting
```

オリジナルのドキュメントのフォーマットが(可能なかぎり)保管されるかどうかを指定します。true の値は、フォーマットの保管を有効化します。false はフォーマットを保管しません。デフォルト値はtrue です。

LastErrorMessage

```
string 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。

LoadXMLWithPSVI

```
bool 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。

OutputEncoding

```
string 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。

OutputIndent

bool 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 の値は無効化します。

OutputMethod

string OutputMethod
出力ドキュメントのシリアル化を指定します。有効な値は以下の通りです :xml | xhtml | html | text。デフォルト値は xml です。

OutputOmitXMLDeclaration

bool OutputOmitXMLDeclaration
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。デフォルト値は false です。

UpdatedXMLWriteMode

ENUMXQueryUpdatedXML UpdatedXMLWriteMode
XML ファイルに対してのアップデートがどのように扱われるか指定します。プロパティの値は ENUMXQueryUpdatedXML リテラルです。

XincludeSupport

bool 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。

XMLValidationErrorAsWarning

bool XMLValidationErrorAsWarning
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。

XMLValidationMode

***ENUMXMLValidationMode* XMLValidationMode**
有効性または整形式をチェックするかを決定する `ENUMXMLValidationMode` (Java、COM/.NET) の列挙リテラルである XML 検証モードを設定します。

XQueryFileName

string 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、使用される XSLT の場所を与える絶対 URL である必要があります。

XQueryFromText

string XQueryFromText
テキスト文字列として、使用する XQuery ドキュメントのコンテンツを提供します。

XSDVersion

***ENUMXSDVersion* 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 (Java、COM/.NET) は、の列挙リテラルです。

IXBRL

IXBRL インターフェイスは、 XBRL インスタンス、タクソノミドキュメント、およびフォーミュラを検証するためのメソッドを提供します。結果はブール値のtrue かまたはfalse です。インターフェイスは、フォーミュラパラメーターもフォーミュラ評価に適合することができます。フォーミュラアサーションと出力も読み込むことができ、文字列として返されます。プロパティは、インターフェイスのパラメーターを定義します。

メモ: 文字列の入力がURL と解釈される箇所では、絶対パスが使用されるべきです。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

構造
以下の構造が定義されます。

```
public struct XBRLParamValuePair
{
   String ParamType;
   String ParamValue;
};
```

メソッド
IXBRL インターフェイスのメソッドは下に説明されています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。

IsValid

```
bool IsValid(ENUMXBRLValidationType type)
```

- XBRL インスタンスドキュメントまたはXBRL タクソノミドキュメントの検証の結果を返します。
- nType はENUMXBRLValidationType の値です。検証の型はXBRL インスタンスドキュメントまたはXBRL タクソノミが検証されるかを指定します。デフォルトは、RaptorXML により自動的に決定されるドキュメントを表示するeValidateXBRLAny です。
- 実行中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

EvaluateFormula

```
bool EvaluateFormula()
```

- XBRL インスタンスドキュメント内のXBRL フォーミュラを評価します。フォーミュラが有効な場合は、true を無効な場合はfalse を返します。
- 実行中にエラーが発生した場合、RaptorXMLException が挙げられます。追加情報にアクセスするには、LastErrorMessage オペレーションを使用してください。

GenerateTables

```
bool GenerateTables()
```

- XBRL インスタンスドキュメント内の XBRL テーブルを評価します。成功時には `true`、失敗時には `false` を返します。
- 実行中にエラーが発生した場合、`RaptorXMLException` が挙げられます。追加情報にアクセスするには、`LastErrorMessage` オペレーションを使用してください。

AddFormulaArrayParameter

```
void AddFormulaArrayParameter(string defaultType, string name, object[] values)
```

- フォーミュラ評価プロセス内で使用されている列挙パラメーターを追加します。
- 全ての引数は文字列です：`sDefaultType` は、配列の値の内部のペアでない値のデフォルトのデータ型です。デフォルトは `xs:string` です。`sName` は、パラメータの名前です。`variantValues` は、値の配列とデータ型の値のペアです。
- コードサンプルの詳細に関しては、XBRL フォーミュラパラメーター のセクションを参照してください。

AddFormulaParameter [DEPRECATED]

```
void AddFormulaParameter(string type, string name, string value, string namespace)
```

- フォーミュラの評価にパラメーターを追加します。これは、非奨励です。
- 全ての引数は文字列です：`sType` は、パラメーターのデータ型です。`sName` は、パラメータの名前です。`sValue` は、パラメータの値です。`sNamespace` は、パラメーターの名前空間です。
- 各パラメーターはメソッドに対して個別の呼び出しにより指定される必要があります。

AddFormulaParameter

```
void AddFormulaParameter(string type, string name, string value, string namespace="")
```

- フォーミュラの評価にパラメーターを追加します。
- 全ての引数は文字列です：`sType` は、パラメーターのデータ型です。`sName` は、パラメータの名前です。`sValue` は、パラメータの値です。`sNamespace` は、パラメーターの名前空間で空の文字列です。
- 各パラメーターはメソッドに対して個別の呼び出しにより指定される必要があります。

AddFormulaParameterNamespace

```
void AddFormulaParameterNamespace(string prefix, string uri)
```

- パラメーター名、型、または値の QNames 内で使用される名前空間を定義します。
- 全ての引数は文字列です：`sPrefix` は `AddFormulaArrayParameter` にパスされる名前空間プレフィックスの値です。`sURI` は名前空間 URI です。
- 各パラメーターはメソッドに対して個別の呼び出しにより指定される必要があります。

AddIXBRLTransformationRegistryLimit

```
void AddIXBRLTransformationRegistryLimit(string limit)
```

- 指定されているバージョンに対して使用することのできる XBRL 変換レジストリを制限します。

- sLmit は、使用することのできるインラインXBRL レジストリが制限されているバージョンです。

.

ClearFormulaParameterList

```
void ClearFormulaParameterList()
```

- AddFormulaParameter メソッドを使用して作成されたフォーミュラパラメーターのリストをクリアします。

ClearIXBRLTransformationRegistryLimit

```
void ClearIXBRLTransformationRegistryLimit()
```

- 指定されているバージョンに対して使用することのできるXBRL 変換レジストリの制限を削除します。

ReadFormulaAssertions

```
void ReadFormulaAssertions()
```

- 評価されるファイルからフォーミュラアサーションを読み込みます。

ReadFormulaOutput

```
void ReadFormulaOutput()
```

- ファイルのフォーミュラアサーションの出力を読み込みます。

プロパティ

IXBRL インターフェイスのプロパティは、下にアルファベット順に説明されています。テーブルはプロパティを簡単参照のためグループ化して表示しています。URL として解釈される文字列の入力は絶対パスを提供する必要があることに注意してください。相対パスが使用される場合、相対パスを解決するメカニズムが呼び出し元モジュール内で定義される必要があります。.

AddAssertionForProcessing

```
string AddAssertionForProcessing
```

アサーションの評価を、与えられたアサーションのみに制限します。1つ以上のアサーションを指定するために複数回呼び出します。アサーション無しの場合は、##none を使用し、すべてのアサーションの場合は##all を使用します。

AddAssertionSetForProcessing

```
string AddAssertionSetForProcessing
```

アサーションセットの評価を、与えられたアサーションセットのみに制限します。1つ以上のアサーションセットを指定するために複数回呼び出します。アサーションセット無しの場合は、##none を使用し、すべてのアサーションセットの場合は##all を使用します。

AddTableForProcessing

```
string AddTableForProcessing
```

テーブルの生成を与えられたテーブルのみに制限します。1つ以上のテーブルを指定するために複数回呼び出します。テーブル無しの場合は、##none を使用し、すべてのテーブルの場合は##all を使用します。

ConceptLabelLinkrole

string ConceptLabelLinkrole
コンセプトラベルをレンダする際、優先する拡張リンクロールを指定します。

ConceptLabelRole

string ConceptLabelRole
コンセプトラベルをレンダする際、優先するラベルロールを指定します。デフォルト:http://www.xbrl.org/2008/role/label。
.

DimensionExtensionEnabled

bool DimensionExtensionEnabled
XBRL ディメンション拡張子 検証を有効化または無効化します。true の値は、ディメンション拡張子検証を有効化します。 false の値は無効化します。デフォルトは true です。 .

EnableIXBRLValidateTarget

bool EnableIXBRLValidateTarget
生成された XBRL ドキュメントの XBRL 検証を有効化します。デフォルト:true.

EvaluateReferencedParametersOnly

bool EvaluateReferencedParametersOnly
false の場合、フォーミュラ/アサーション/テーブルにより参照されていない場合でも、全てのパラメーターが強制的に評価されます。デフォルト:true。

FormulaAssertionsAsXML

bool FormulaAssertionsAsXML
フォーミュラアサーションファイルのXML 書式設定を有効化します。RaptorXML がアサーションが有効化されていて実行される場合、true の値は、XML フォーマットを有効化します。false の値は、JSON 出力を生成します。デフォルトは false です。

FormulaAssertionsOutput

string FormulaAssertionsOutput
フォーミュラアサーション出力ファイルの場所を指定します。フルパスを指定する必要があります。

FormulaExtensionEnabled

bool FormulaExtensionEnabled
XBRL フォーミュラ拡張子検証を有効化または無効化します。 true の値は、フォーミュラ拡張子検証を有効化します。false の値は無効化します。デフォルトは true です。

FormulaOutput

string FormulaOutput
XBRL フォーミュラ評価ファイルの出力の場所を指定します。フルパスを指定する必要があります。

FormulaParameterFile

string FormulaParameterFile
フォーミュラパラメーターファイルの場所を指定します。フルパスを指定する必要があります。

FormulaPreloadSchemas

bool FormulaPreloadSchemas
フォーミュラがプリロードされるか定義します。`true` の値はスキーマをプリロードします。デフォルトは、スキーマがプリロードされないように指定する`false` です。

GenericLabelLinkrole

string GenericLabelLinkrole
ジェネリックラベルをレンダする際、使用される優先する拡張されたリンクロールを指定します。

GenericLabelRole

string GenericLabelRole
ジェネリックラベルをレンダする際、優先するラベルロールを指定します。デフォルト:`http://www.xbrl.org/2008/role/label`。.

InputFileArray

object InputFileArray
入力データインスタンスとして使用されるXBRL ファイルの配列を設定します。配列は各入力ファイルの絶対 URL の文字列を含むオブジェクトです。

InputFileName

string InputFileName
XBRL インスタンスファイルのファイル名と場所を指定します。提供される文字列は絶対 URL か、呼び出しモジュール内で定義されたメカニズムに従い、ベースロケーションに対して相対的に解決することのできる相対パスである必要があります。

InputFromText

string InputFromText
XBRL 入力ドキュメントのコンテンツをテキストとして提供します。

InputTextArray

object InputTextArray
入力データとして使用されるテキストファイルの配列を設定します。配列は各入力ファイルの絶対 URL の文字列を含むオブジェクトです。

IXBRLOutput

string IXBRLOutput
生成されたXBRL 出力ファイルの出力場所を設定します。与えられた文字列は、出力の場所の絶対 URL を保持しています。

IXBRLTreatAsDocumentSet

bool IXBRLTreatAsDocumentSet
単一のインラインXBRL ドキュメントセットとしてすべての入力が扱われるかを設定します。デフォルトは `false` です。

IXBRLUriStrategy

ENUMIXBRLUriStrategy IXBRLUriStrategy
生成されたXBRL ドキュメント内で、インラインXBRL URI がどのように変換されるかを指定します。選択された ENUMIXBRLUriStratgey リテラルを取ります。
.

IXBRLVersion

ENUMIXBRLVersion IXBRLVersion
使用するインラインXBRL バージョンを指定します。選択された ENUMIXBRLVersion リテラルを取ります。

LabelLang

string LabelLang
ラベルをレンダする際使用する優先ラベル言語を指定します。デフォルト：`en`。

LastErrorMessage

string 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。

ParallelAssessment

bool ParallelAssessment
パラレルスキーマの有効性評価を有効化/無効化します。

PreloadSchemas

bool PreloadSchemas
XBRL 2.1 スキーマがプリロードされるか定義します。`true` の値はスキーマをプリロードします。デフォルトは `true` です。

PythonScriptFile

string PythonScriptFile
検証のために提出されたXML または XSD ファイルの追加処理を提供する、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 スクリプト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。

SchemaImports

***ENUMSchemaImports* 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 (Java、COM/.NET) により指定されます。

SchemalocationHints

***ENUMLoadSchemalocation* 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 (Java、COM/.NET) により指定されます。

SchemaMapping

***ENUMSchemaMapping* 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 (Java、COM/.NET) により指定されます。 .

TableEliminateEmptyRows

bool TableEliminateEmptyRows
テーブル生成の HTML 出力内の空の行 列を削除することを有効化します。

TableExtensionEnabled

bool TableExtensionEnabled
XBRL Table 1.0 拡張子を有効化 無効化します。

TableLinkbaseNamespace

string TableLinkbaseNamespace
前のドラフト仕様により書き込まれたテーブルリンクベースのロードを有効化します。与えられた文字列の値は、テーブルリンクベースを指定します。テーブルリンクベースの検証、解決、およびレイアウトは、しかしながら、常に 2014年 3月 18 日版の Table Linkbase 1.0 勧告に従い実行されます。##detect を使用して自動検出を有効化します。以下の値が認識されています :

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

TableOutput

string TableOutput
テーブル生成の出力のファイル名と場所を指定します。提出される文字列は出力ファイルのフルパスである必要があります。

TableOutputFormat

ENUMTableOutputFormat TableOutputFormat
テーブル生成出力ファイルのフォーマットを指定します。

TablePreloadSchemas

```
bool TablePreloadSchemas
```

XBRL Table 1.0 仕様スキーマのプレロードを有効化/無効化します。

TreatXBRLInconsistenciesAsErrors

```
bool TreatXBRLInconsistenciesAsErrors
```

XBRL 2.1 仕様の定義に対する不整合がファイルに含まれる場合、`true` の値は、XBRL 検証の失敗を引き起こします。 デフォルトは `false` です。XBRL 2.1 仕様に従い生じるXBRL 矛盾はエラーとして扱われません。

XIncludeSupport

```
bool 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。

7.4.2 列挙

以下の列挙が定義されています。このセクションのサブセクションで詳細が説明されています。

ENUMAssessmentMode
ENUMErrorFormat
ENUMLoadSchemalocation
ENUMQueryVersion
ENUMSchemaImports
ENUMSchemaMapping
ENUMValidationType
ENUMWellformedCheckType
ENUMXBRLValidationType
ENUMXMLValidationMode
ENUMXQueryVersion
ENUMXSDVersion
ENUMXSLTVersion

ENUMAssessmentMode

説明

XML バリデーターの評価モードを定義する以下の列挙リテラルを含みます：`Strict` または `Lax`.

使用

インターフェイス	*オペレーション*
IXMLValidator	AssessmentMode

列挙リテラル

```
eAssessmentModeStrict = 0
eAssessmentModeLax = 1
```

eAssessmentModeStrict
に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
にスキーマの有効性評価モードを`Lax` に設定します。

ENUMErrorFormat

説明

エラー出力のフォーマットを指定する以下の列挙リテラルを含みます。

使用

インターフェイス	オペレーション
IServer	ErrorFormat

列挙リテラル

```
eFormatText     = 0
eFormatShortXML = 1
eFormatLongXML  = 2
```

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。

ENUMLoadSchemalocation

説明

どの様にスキーマの場所が決定されるかどうかを示す列挙リテラルを含みます。

使用

インターフェイス	オペレーション
IXBRL	SchemalocationHints
IXMLValidator	SchemalocationHints
IXSLT	SchemalocationHints

列挙リテラル

```
eSHLoadBySchemalocat  = 0
ion
eSHLoadByNamespace    = 1
eSHLoadCombiningBoth  = 2
eSHLoadIgnore         = 3
```

eSHLoadBySchemalocation
Schemalocation を次に設定します :`LoadBySchemalocation`。 XML または XBRL インスタンスドキュメント内の`xsi:schemaLocation` および`xsi:noNamespaceSchemaLocation` 属性内のスキーマの場所の URL を使用します。 これは**デフォルトの値**です。

eSHLoadByNamespace
Schemalocation を次に設定します :`LoadByNamespace`。 `xsi:schemaLocation` の一部である名前空間を使用します。 (`xsi:noNamespaceSchemaLocation` の場合は空の文字列) カタログマッピングを使用してスキーマを検索します。

eSHLoadCombiningBoth
Schemalocation を次に設定します :`CombiningBoth`。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` パラメーター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
Schemalocation を次に設定します :`LoadIgnore`。パラメーターの値が`eSHLoadIgnore` の場合、`xsi:schemaLocation` と`xsi:noNamespaceSchemaLocation` 属性は両方とも無視されます。

ENUMQueryVersion

説明
XQuery のバージョンに以下の使用を指定する列挙リテラルを含みます :XQuery 1.0 または 3.0。

列挙リテラル

```
eXQVersion10    = 1
eXQVersion30    = 3
```

eXQVersion10
XQuery のバージョンを XQuery 1.0 に設定します。

eXQVersion30
XQuery のバージョンを XQuery 3.0 に設定します。

ENUMSchemaImports

説明
`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。

使用

インターフェイス	オペレーション
IXBRL	SchemaImports
IXMLValidator	SchemaImports
IXSLT	SchemaImports

列挙リテラル

```
eSILoadBySchemalocation    = 0
eSILoadPreferringSchemalo  = 1
cation
eSILoadByNamespace         = 2
eSICombiningBoth           = 3
eSILicenseNamespaceOnly    = 4
```

eSILoadBySchemalocation
スキーマインポートを以下に設定します：LoadBySchemalocation。カタログマッピングを考慮して、schemaLocation 属性の値がスキーマを検索する際に使用されます。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
スキーマインポートを以下に設定します：LoadPreferringSchemalocation。schemaLocation 属性が存在する場合、カタログマッピングを考慮して、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 マッピングを介して使用されます。このリテラルは列挙の**デフォルトの値**です。

eSILoadByNamespace
スキーマインポートを以下に設定します：LoadByNamespace。カタログマッピングを介して namespace 属性の値がスキーマを検索する際に使用されます。

eSICombiningBoth
スキーマインポートを以下に設定します：CombiningBoth. namespace または schemaLocation 属性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 パラメーターの値がどのマッピングを使用するか決定します。カタログマッピングが存在しない場合、（URL である）schemaLocation 属性の値が使用されます。

eSILicenseNamespaceOnly
スキーマインポートを以下に設定します：LicenseNamespaceOnly。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。

ENUMSchemaMapping

説明
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 および ENUMSchemaImports のあいま

いさを除去するために役に立ちます。

使用

インターフェイス	オペレーション
IXBRL	SchemaMapping
IXMLValidator	SchemaMapping
IXSLT	SchemaMapping

列挙リテラル

```
eSMPreferSchemalocati = 0
on
eSMPreferNamespace    = 1
```

eSMPreferSchemalocation
スキーマロケーションURL を選択するためのスキーママッピングオプションを設定します。

eSMPreferNamespace
名前空間を選択するためのスキーママッピングオプションを設定します。

ENUMTableOutputFormat

説明
生成されたテーブルを含むドキュメントの出力フォーマットを指定するエミュレーションリテラルを含みます。

使用

インターフェイス	オペレーション
IXBRL	TableOutputFormat

列挙リテラル

```
eFormatXML            = 0
eFormatHTML           = 1
```

eSMPreferSchemalocation
スキーマロケーションURL を選択するためのスキーママッピングオプションを設定します。

eSMPreferNamespace
名前空間を選択するためのスキーママッピングオプションを設定します。

ENUMValidationType

説明

検証するドキュメントの型を定義する、列挙リテラルを含みます。

使用

インターフェイス	オペレーション
IXMLValidator	IsValid

列挙リテラル

```
eValidateAny      = 0
eValidateXMLWith  = 1
DTD
eValidateXMLWith  = 2
XSD
eValidateDTD      = 3
eValidateXSD      = 4
eValidateJSON     = 5
eValidateJSONSch  = 6
ema
eValidateAvro     = 7
eValidateAvroSch  = 8
ema
eValidateAvroJSO  = 9
N
```

eValidateAny
検証の型を以下に設定します：`Any`。自動的に型を検出した後、ドキュメントを検証します。

eValidateXMLWithDTD
検証の型を以下に設定します：`XMLWithDTD`。これは、XML ドキュメントの DTD に対しての検証を指定します。

eValidateXMLWithXSD
検証の型を以下に設定します：`XMLWithXSD`。これは XML ドキュメントの XML スキーマに対しての検証を指定します。

eValidateDTD
検証の型を以下に設定します：`ValidateDTD`。DTD ドキュメントの検証を指定します。

eValidateXSD
検証の型を以下に設定します：`ValidateXSD`。W3C XML スキーマ ドキュメントの検証を指定します。

eValidateJSON
検証の型を以下に設定します：ValidateJSON。JSON スキーマ v 4 に従い JSON インスタンスドキュメントの検証を指定します。

eValidateJSONSchema
検証の型を以下に設定します： ValidateXSD。JSON スキーマ v 4 に従い JSON インスタンスドキュメントの検証を指定します。

eValidateAvro
検証の型を以下に設定します：ValidateAvro。Avro バイナリファイルの検証を指定します。

eValidateAvroSchema
検証の型を以下に設定します：ValidateAvroSchema。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
検証の型を以下に設定します：ValidateAvroJSON。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。

ENUMWellformedCheckType

説明
チェックするドキュメントの型を定義する列挙リテラルを含みます：XML または DTD。

使用

インターフェイス	*オペレーション*
IXMLValidator	IsWellFormed

列挙リテラル

```
eWellFormedAny  = 0
eWellFormedXML  = 1
eWellFormedDTD  = 2
eWellFormedJSO  = 3
N
```

eWellformedAny
全ての型に対しての整形式のチェックを設定します。XML または DTD の 2 つのうちどちらの型かを自動的に検出した後、ドキュメントの整形式をチェックします。

eWellformedXML
XML の型に対しての整形式のチェックを設定します。これは、XML 1.0 または XML 1.1 仕様に従い、XML ドキュメントの整形式をチェックします。

eWellformedDTD
DTD の型に対しての整形式のチェックを設定します。このDTD ドキュメントの整形式をチェックします。

eWellformedJSON
JSON の型に対しての整形式のチェックを設定します。これは、JSON ドキュメントの整形式をECMA-404 仕様に従いチェックします。

ENUMXBRLValidationType

説明
検証する XBRL ドキュメントの型を定義する列挙リテラルを含みます XBRL インスタンスまたは、XBRL タクソノミ。

使用

インターフェイス	オペレーション
IXBRL	IsValid

列挙リテラル

```
eValidateXBRLAny          = 0
eValidateXBRLInstance     = 1
eValidateXBRLTaxonomy     = 2
eValidateXBRLInline       = 3
eValidateXBRLTaxonomyP    = 4
ackage
```

eValidateXBRLAny
検証の型を以下に設定します :Any。これは、自動的に型 (インスタンスまたはタクソノミ)を検出した後にXBRL ドキュメントを検証します。

eValidateXBRLInstance
検証の型を以下に設定します :Instance。これは、1つまたは複数のXBRL インスタンスドキュメントの検証を指定します。

eValidateXBRLTaxonomy
検証の型を以下に設定します :Taxonomy。これは、1つまたは複数のタクソノミドキュメントの検証を指定します。

eValidateXBRLInline
検証の型を以下に設定します : Inline。1つまたは複数のXBRL ドキュメントの検証を指定します。

eValidateXBRLTaxonomyPackage
検証の型を以下に設定します : TaxonomyPackage。1つまたは複数のXBRL タクソノミパッケージの検証を指定します。

ENUMIXBRLVersion

説明

使用するインライン XBRL 使用のバージョンを指定する列挙リテラルを含んでいます :インラインXBRL 1.0 または インラインXBRL 1.1. **eVersionDetect** は自動検出を有効化します。

使用

インターフェイス	*オペレーション*
IXBRL	IXBRLVersion

列挙リテラル

```
eVersion10              = 0
eVersion11              = 1
eVersionDetect          = 2
```

eVersion10
インラインXBRL 1.0 使用を使用するものとして設定します。

eVersion11
インラインXBRL 1.1 使用を使用するものとして設定します。

eVersionDetect
提出されたインラインXBRL ドキュメントからのバージョンの自動検出を有効化します。

ENUMIXBRLUriStrategy

説明

インラインXBRL 内のURI が生成されたXBRL ドキュメントにどのように書き込まれるかを指定する列挙リテラルを含みます。

使用

インターフェイス	*オペレーション*
IXBRL	IXBRLUriStrategy

列挙リテラル

```
eStrategyNone           = 0
eStrategyMakeAbsolute   = 1
eStrategyMakeRelative   = 2
eStrategyKeepRelative   = 3
```

eStrategyNone
URI verbatim をターゲットドキュメントにコピーします。

eStrategyMakeAbsolute
入力ドキュメント内の対応する要素でインスコープのベースURI に対して解決することにより相対的なURI を絶対的 URI にします。

eStrategyMakeRelative
可能な場合、絶対、および、相対 URI を出力ドキュメントに対して相対的にします (それ以外の場合、解決された絶対 URI を書き込みます)。

eStrategyKeepRelative
可能な場合、相対的なURI のみを出力ドキュメントに対して相対的にします (そして絶対的 URI をコピーします)。

ENUMXMLValidationMode

説明
使用する XML 処理モードを定義する以下の列挙リテラルを含みます。検証 または 整形式。

使用

インターフェイス	オペレーション
IXMLValidator	XMLValidationMode
IXQuery	XMLValidationMode
IXSLT	XMLValidationMode

列挙リテラル

```
eXMLValidationMode = 0
WF
eXMLValidationMode = 1
ID
eXMLValidationMode = 2
Valid
```

eXMLValidationModeWF
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。

eXMLValidationModeID
内部。

eXMLValidationModeValid
XML 処理モードを`Validation` に設定します。

ENUMXQueryVersion

説明

XQuery のバージョンに以下の使用を指定する列挙リテラルを含みます：XQuery 1.0 または 3.0。

使用

インターフェイス	オペレーション
IXQuery	EngineVersion

列挙リテラル

```
eXQVersion10    = 1
eXQVersion30    = 3
```

eXQVersion10
XQuery のバージョンを XQuery 1.0 に設定します。

eXQVersion30
XQuery のバージョンを XQuery 3.0 に設定します。これはデフォルトの値です。

ENUMXQueryUpdatedXML

説明

XQuery のアップデートが処理されるかどうか指定する列挙リテラルを含みます。

使用

インターフェイス	オペレーション
IXQuery	UpdatedXMLWriteMode

列挙リテラル

```
eUpdatedDiscard    = 1
eUpdatedWriteback  = 2
eUpdatedAsMainResu = 3
lt
```

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、InputXMLFileName と共に指定されます。

eUpdatedAsMainResult

アップデートは、ExecuteUpdate 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。

ENUMXSDVersion

説明

検証に使用する XML スキーマバージョン示す以下の列挙リテラルを含みます。:XSD 1.0 または 1.1。

使用

インターフェイス	*オペレーション*
IXMLValidator	XSDVersion
IXQuery	XSDVersion
IXSLT	XSDVersion

列挙リテラル

```
eXSDVersionAut  = 0
o
eXSDVersion10   = 1
eXSDVersion11   = 2
```

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。

ENUMXSLTVersion

説明

使用する XSLT バージョンを定義する以下の列挙リテラルを含みます。:XSLT 1.0、2.0、または 3.0。

使用

インターフェイス	*オペレーション*
IXSLT	EngineVersion

列挙リテラル

```
eVersion10 = 1
eVersion20 = 2
eVersion30 = 3
```

eVersion10
XSLT バージョンをに XSLT 1.0 設定します。

eVersion20
XSLT バージョンをに XSLT 2.0 設定します。

eVersion30
XSLT バージョンをに XSLT 3.0 設定します。

チャプター 8

追加情報

8 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：

- XBRL フォーミュラパラメーター

8.1 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+XBRL 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` 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。

8.2 XBRL フォーミュラ パラメーター

このセクションは以下のトピックを含みます：

- XBRL フォーミュラ パラメーターのXML とJSON フォーマットのサンプルを与える、XBRL フォーミュラ パラメーター フォーマット;
- フォーミュラ パラメーターの使用 は、フォーミュラ パラメーターがJava およびCOM/.NET API ライブラリからのオブジェクトを使用することができることを表示するJava、VB.NET、C#、VBScript、およびJScript リストを含みます。

8.2.1 フォーミュラ パラメーターフォーマット

フォーミュラパラメーターはXML フォーマットまたはJSON フォーマットであることができます。

XML フォーマット

XML フォーマット内のフォーミュラパラメーターを下のリストは表示しています。

```
<?xml version="1.0" encoding="utf-8"?>
<options:formula-parameters
  xmlns:options="http://www.altova.com/schemas/altova/raptorxml/options"
  xmlns:p="http://xbrl.org/formula/conformance/param-stuff"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.altova.com/schemas/altova/raptorxml/options http://www.altova.com/
schemas/altova/raptorxml/options.xsd">

  <options:parameter name="p1">
    <options:value type="xs:string">hello world from new xml (without namespace)</options:value>
  </options:parameter>
  <options:parameter name="pp1" type="xs:string" value="hello world from new xml"/>

</options:formula-parameters>
```

以下の点に注意してください:

- `@type` 属性は任意です。デフォルトは `xs:string` です。 .
- パラメーターにシーケンスを割り当てるために、複数の `<options:value>` 子要素を指定することができます。
- `@value` と `<options:value>` は、同時に使用することができません。

JSON フォーマット

JSON フォーマット内のフォーミュラパラメーターを下のリストは表示しています。

```
{
  "formula-parameters":[
  {
    "name":"p1",
    "values":[
      {
        "type":"xs:string",
        "value":"hello world from json new (without namespace)"
      }
    ]
  },{
      "name":"ns1:p1",
      "values":[
        {
          "type":"xs:string",
          "value":"hello world from json new"
        }
       ]
      }
  ],
  "namespaces":{
    "xs":"http://www.w3.org/2001/XMLSchema",
    "ns1":"http://xbrl.org/formula/conformance/param-stuff"
  }
}
```

以下の点に注意してください:

- `type` キーは任意です。デフォルトは `xs:string` です。
- `xs` キーは任意です。デフォルトは `http://www.w3.org/2001/XMLSchema` です。
- JSON 文字列で値が直接指定されている場合、パラメーターマップ内の型が使用されます。
- 現在、以下でもサポートされています:

```
  {
  "name":"p2",
  "type":"xs:string",
  "value":"hello world from json new (without namespace)"
},{
  "name":"p3",
  "type":"xs:int",
  "values":["1","2"]
},{
  "name":"p4",
  "type":"xs:int",
  "values":["1",{"type":"xs:string","value":"abc"},"2"]
}
```

8.2.2 フォーミュラ パラメーターの使用

下のサンプルは、XBRL フォーミュラパラメーターがどのように異なるプログラム言語で使用されるかを表示しています。Java に関しては、Java API XBRL クラスを参照してください。他の言語に関しては、COM/.NET API のXBRL インターフェイスを参照してください。

Java

```
RaptorXMLFactory rxml = RaptorXML.getFactory();
XBRL xbrl = rxml.getXBRL();

xbrl.addFormulaParameter("ns1:string","ns1:Param1","ns1:theqname");
xbrl.addFormulaParameterNamespace("ns1","www.www.www");

// The parameter is an array of dates
xbrl.addFormulaArrayParameter("","startDates",new Object[]{new FormulaParam("xs:date",
"2010-01-01"),new FormulaParam("xs:date","2012-01-01")});

// The parameter is an array of figs
xbrl.addFormulaArrayParameter("ns1:figs","startFigs",new Object[]{"fig1","fig2","fig3"});

// The parameter is an array of figs, dates and raisins (rather wild example)
xbrl.addFormulaArrayParameter("ns1:figs","startDryFruit",new Object[]{"fig1","fig2",new
FormulaParam("xs:date","2010-01-01"),new FormulaParam("ns1:raisin","dried grape"),"fig3"});
```

VB.NET

```
Dim objRaptor As New Server()
  Dim objXBRL As XBRL
  objXBRL = objRaptor.GetXBRL()

  objXBRL.AddFormulaParameter("ns1:string","ns1:Param1","ns1:theqname")
  objXBRL.AddFormulaParameterNamespace("ns1","www.www.www")

  'The parameter is an array of dates
  objXBRL.AddFormulaArrayParameter("","startDates",{New XBRLFormulaParam With {.ParamType =
"xs:date", .ParamValue = "2010-01-01"},New XBRLFormulaParam With {.ParamType =
"xs:date", .ParamValue = "2012-01-01"}})

  'The parameter is an array of figs
  objXBRL.AddFormulaArrayParameter("ns1:figs","startFigs",{"fig1","fig2","fig3"})

  'The parameter is an array of figs, dates and raisins (rather wild example)
  objXBRL.AddFormulaArrayParameter("ns1:figs","startDryFruit",{"fig1","fig2",New
XBRLFormulaParam With {.ParamType = "xs:date", .ParamValue = "2010-01-01"},New
XBRLFormulaParam With {.ParamType = "ns1:raisin", .ParamValue = "dried grape"},"fig3"})
```

C#

```
Server app = new Server();
  XBRL objXBRL = app.GetXBRL();

  objXBRL.AddFormulaParameter("ns1:string","ns1:Param1","ns1:theqname");
  objXBRL.AddFormulaParameterNamespace("ns1","www.www.www");

  //The parameter is an array of dates
  objXBRL.AddFormulaArrayParameter("","startDates",new object[] {new XBRLFormulaParam
{.ParamType = "xs:date",.ParamValue = "2010-01-01"},new XBRLFormulaParam {.ParamType =
"xs:date",.ParamValue = "2012-01-01"}});

  //The parameter is an array of figs
  objXBRL.AddFormulaArrayParameter("ns1:figs","startFigs",new object[] {"fig1","fig2","fig3"});
```

```
  //The parameter is an array of figs, dates and raisins (rather wild example)
  objXBRL.AddFormulaArrayParameter("ns1:figs","startDryFruit",new object[] {"fig1","fig2",new
XBRLFormulaParam {ParamType = "xs:date",ParamValue = "2010-01-01" },new XBRLFormulaParam {
ParamType = "ns1:raisin",ParamValue = "dried grape" },"fig3" });
```

VBScript

Raptor 型ライブラリは、スクリプトの言語でロードされることができないため、また、`XBRLFormulaParameters` 型が存在しないため、 VBScript ユーザーは、`XBRL.FormulaParam` オブジェクトを使用する代わりに、クラスをプログラムで宣言する必要があります。(`XBRL.FormulaParam` が持つように)。クラスには以下の 2 つのパブリックプロパティが必要です :`ParamName` および `ParamValue` クラスには、使用を簡素化するために型と値を取るコンストラクターが必要です。それ以外の場合、メンバーセットを持つ、オブジェクトが作成される必要があります。 COM/.NET API's XBRL インターフェイスを参照してください。

```
Class MyPair
  Public ParamType
  Public ParamValue
  Public Default Function Init(inType,inValue)
       ParamType = inType
       ParamValue = inValue
       set Init = Me
  End Function
End Class


Sub Main
  Dim objRaptor
  Set objRaptor = WScript.GetObject("","RaptorXML.Server")
  Dim objXBRL
  Set objXBRL = objRaptor.GetXBRL

  Call objXBRL.AddFormulaParameter("ns1:string","ns1:Param1","ns1:theqname")
  Call objXBRL.AddFormulaParameterNamespace("ns1","www.www.www")

  'The parameter is an array of dates
  Call objXBRL.AddFormulaArrayParameter("","startDates",Array((New MyPair)("xs:date","2010-
01-01"),(New MyPair)("xs:date","2012-01-01")))

  'The parameter is an array of figs
  Call objXBRL.AddFormulaArrayParameter("ns1:figs","startFigs",Array("fig1","fig2","fig3"))

  'The parameter is an array of figs, dates and raisins (rather wild example)
  Call objXBRL.AddFormulaArrayParameter("ns1:figs","startDryFruit",Array("fig1","fig2",(New
MyPair)("xs:date","2010-01-01"),(New MyPair)("ns1:raisin","dried grape"),"fig3"))
End Sub

Call Main
```

JScript

Raptor 型ライブラリは、スクリプトの言語でロードされることができないため、また、`XBRLFormulaParameters` 型が存在しないため、 JScript ユーザーは、`XBRL.FormulaParam` オブジェクトを使用する代わりに、型の値ペアを保有するプログラムの関数クラスを、宣言する必要があります。メンバーの名前は、ParamType および `ParamValue` である必要があります。COM/.NET API のXBRL インターフェイスを参照してください。

```
function FormulaParam(inType,inValue)
{
```

```
  this.ParamType = inType;
  this.ParamValue = inValue;
}

function main()
{
  var objRaptor = new ActiveXObject("RaptorXMLServer");
  var objXBRL = objRaptor.GetXBRL();

  objXBRL.addFormulaParameter("ns1:string","ns1:Param1","ns1:theqname");
  objXBRL.addFormulaParameter("xs:string","Param1","bla","www:www:www");

  // The parameter is an array of dates
  objXBRL.addFormulaArrayParameter("","startDates",[new FormulaParam("xs:date","2010-01-01"),
new FormulaParam("xs:date","2012-01-01")]);

  // The parameter is an array of figs
  objXBRL.addFormulaArrayParameter("ns1:figs","startFigs",["fig1","fig2","fig3"]);

  // The parameter is an array of figs, dates and raisins (rather wild example)
  objXBRL.addFormulaArrayParameter("ns1:figs","startDryFruit",["fig1","fig2",new
FormulaParam("xs:date","2010-01-01"),new FormulaParam("ns1:raisin","dried grape"),"fig3"]);
}

main()
```

チャプター 9

XSLT および XQuery エンジンに関する情報

9 XSLT および XQuery エンジンに関する情報

RaptorXML+XBRL Server の XSLT および XQuery エンジンは、W3C 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 よりも厳密です。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+XBRL 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
- XSLT 2.0
- XSLT 3.0
- XQuery 1.0
- XQuery 3.1

9.1 XSLT 1.0

RaptorXML+XBRL Server のXSLT 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、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

9.2 XSLT 2.0

このセクション:

適合性

RaptorXML+XBRL Server のXSLT 2.0 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`) 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。t

- XML スキーマ名前空間から型コンストラクタ型を使用する場合、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`.

9.3 XSLT 3.0

RaptorXML+XBRL Server のXSLT 3.0 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2017 年 2月 7日版 XSLT 3.0 勧告候補 および2017 年 1月 17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。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。

9.4 XQuery 1.0

このセクション:

適合性

RaptorXML+XBRL 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。下には、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

XML ドキュメントused in executing an XQuery ドキュメントwith XQuery 1.0 エンジンmust be 整形式 .しかし、they do not need to be valid according to an XML スキーマ.If the ファイルis not valid, the invalid ファイルis loaded without schema information. 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、then post-schema 検証情報 is generated for XML データand will be used for クエリ検証 .

静的および動的な型のチェック

静的分析フェーズchecks aspects of クエリsuch as syntax, whether 外部レファレンス (例 for モジュール) exist, whether invoked 関数と変数 are defined, and so on. If an error is detected in 静的分析フェーズ, it is reported and the execution is stopped.

動的な型チェックis carried out ランタイム中 ,when クエリis actually executed. If a type is incompatible with the requirement of an operation, an error is reported. 例えば、式`xs:string("1") + 1` は、エラーを返します。because the addition operation cannot be carried out on an operand of type `xs:string`.

ライブラリモジュール

ライブラリモジュールstore 関数と変数 so they can be reused. XQuery 1.0 エンジンsupports モジュールthat are stored in **a single external XQuery ファイル**. Such モジュール ファイルmust contain モジュール 宣言 in its prolog, which associates a target 名前空間 .以下はモジュールサンプルです:

```
module 名前空間 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is used by importing it into an XQuery ファイルwith the `import module` statement クエリプロローグ内の. `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。i.e. in those 式 using the `external` keyword, as in:

```
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 is the Unicode-コードポイント照合順序 ,which compares strings on the basis of their Unicode コードポイント.その他にサポートされる照合順序 are the ICU 照合順序 listed here. To use a specific 照合順序 ,supply its URI as given in the list of supported 照合順序. Any string comparisons, including for the `fn:max` and `fn:min` 関数 ,will be made according to the specified 照合順序 .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

- The `xs:integer` データ型 is arbitrary-精度 ,i.e. it can represent any number of 桁 .
- The `xs:decimal` データ型 has a limit of 20 桁 after the decimal point.
- The `xs:float` and `xs:double` データ型 have limited-精度 of 15 桁 .

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。

9.5 XQuery 3.1

of RaptorXML+XBRL 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 1月 17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 でも同様です。

チャプター 10

XSLT と XPath/ XQuery 関数

10 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。

10.1 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。

W3C 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 (i) 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。W3C 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`
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`

XSLT 関数

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
- 位置情報
- イメージに関連した
- 数値
- シーケンス
- 文字列
- その他

チャート関数 (Enterprise および Server Editions のみ)

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

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。

10.1.1 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

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ご注意ください。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
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

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。 (ii) 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 1 つ以上の部分を持つ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、呼び出し環境の名前空間、型、機能、しかし変数を以外を含みます。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]

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]

10.1.2 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

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ご注意ください。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
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

▼ 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
- xs:date に期間を追加して、xs:date を返す
- xs:time に期間を追加して、return xs:time を返す
- フォーマットと期間の取得
- 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
- 日付から整数を週の曜日として返す
- 日付から週数を整数として返す
- 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
- 文字列入力から日付、日付時刻 または 時刻 を構築する
- 年齢に関連した関数

▼ アルファベット順にグループ化

altova:add-days-to-date
altova:add-days-to-dateTime
altova:add-hours-to-dateTime
altova:add-hours-to-time
altova:add-minutes-to-dateTime
altova:add-minutes-to-time
altova:add-months-to-date
altova:add-months-to-dateTime
altova:add-seconds-to-dateTime
altova:add-seconds-to-time
altova:add-years-to-date
altova:add-years-to-dateTime
altova:age
altova:age-details
altova:build-date
altova:build-duration
altova:build-time
altova:current-dateTime-no-TZ
altova:current-date-no-TZ
altova:current-time-no-TZ
altova:format-duration
altova:parse-date
altova:parse-dateTime

[トップ]

xs:dateTime に期間を追加する XP3 XQ3

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*) as **xs:dateTime** とする XP3 XQ3

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) returns 2024-01-15T14:00:00
- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -4) returns 2010-01-15T14:00:00

add-months-to-dateTime [altova:]

altova:add-months-to-dateTime(DateTime *as xs:dateTime*, **Months** を *xs:integer*) as **xs:dateTime** とする XP3 XQ3

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 XQ3

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** **XQ3**

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** **XQ3**

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*) as **xs:dateTime** とする **XP3** **XQ3**

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]

xs:date に期間を追加する XP3 XQ3

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 XQ3

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 XQ3

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 XQ3

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]

フォーマットと期間の取得 XP3 XQ3

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。

format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) as **xs:string** とする **XP3 XQ3**
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 XQ3**
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。

[Top]

---

xs:time に期間を追加する XP3 XQ3

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 XQ3**
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 XQ3
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 XQ3
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。

[Top]

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 XP3 XQ3

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-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() をxs:dateTime とする XP3 XQ3
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-date-no-TZ [altova:]

altova:current-date-no-TZ() をxs:date とする XP3 XQ3
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-time-no-TZ [altova:]

altova:current-time-no-TZ() をxs:time とする XP3 XQ3

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。

[Top]

---

xs:dateTime または xs:date として曜日を返す XP3 XQ3

これらの関数は、xs:dateTime または xs:date から曜日を (整数として) 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 XQ3

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*) as **xs:integer** とする XP3 XQ3

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 XQ3

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 XQ3**

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 引数が整数で 0 以外の場合、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。

[Top]

xs:dateTime または xs:date から週数を返す XP2 XQ1 XP3 XQ3

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 XQ3**

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 XQ3**

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も一週間先です。

[トップ]

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 XQ3

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 XQ3
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 XQ3
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 XQ3
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 XQ3
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 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 XQ3
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]

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 XQ3

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 XQ3
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 XQ3**

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 XQ3**

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]

年齢に関連した関数 XP3 XQ3

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 (i) 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 XQ3

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年よりも大きい場合、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 XQ3

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の引数よりも後の日付の場合、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 XQ3

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も早いまたは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 XQ3**

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。

[Top]

10.1.3 XPath/ XQuery 関数 :位置情報

以下の位置情報 XPath/XQuery 拡張関数は、RaptorXML+XBRL 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

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ご注意ください。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
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

parse-geolocation [altova:]

altova:parse-geolocation(GeolocationInputString *as xs:string*) を **xs:decimal+** とする XP3 XQ3

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 関数と Exif メタデータの @Geolocation 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 (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: 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, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E

サンプル: `33°55'11.11"N 22°44'66.66"W`

- 度、分、10進の秒、プレフィックスサイン付き (+/-); (N/W) 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"`

- 度、分、10進の分、方角のサフィックス付き (N/S, W/E)
  `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/W) 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, W/E)
  `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/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°44'66.66"
33.33  22°44'66.66"W
33.33  22.c
```

- *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 XQ3**

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` 関数と Exif メタデータの `@Geolocation` 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 (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: 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, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'66.66"W**

- 度、分、10進の秒、プレフィックスサイン付き (+/-); (**N/W**) 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"**

- 度、分、10進の分、方角のサフィックス付き (N/S, W/E)
  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/W**) 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, W/E)
  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/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°44'66.66"
33.33  22°44'66.66"W
33.33  22.c
```

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 XQ3**
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 関数とExif メタデータの @Geolocation 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 (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: 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, W/E)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'66.66"W**

- 度、分、 10進の秒、プレフィックスサイン付き (+/-); (**N/W**) 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"**

- 度、分、 10進の分、方角のサフィックス付き (N/S, W/E)
  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/W**) 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, W/E)
  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/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°44'66.66"
33.33 22°44'66.66"W
33.33 22.c

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 XQ3**

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 関数と Exif メタデータの @Geolocation 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 (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: 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, W/E)
  `D°M'S.SS"N/S D°M'S.SS"W/E`
  サンプル: `33°55'11.11"N 22°44'66.66"W`

- 度、分、10進の秒、プレフィックスサイン付き (+/-); (N/W) 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"`

- 度、分、10進の分、方角のサフィックス付き (N/S, W/E)
  `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/W) 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, W/E)
  `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/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°44'66.66"
33.33  22°44'66.66"W
33.33  22.c
```

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` `XQ3`

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 関数とExif メタデータの @Geolocation 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 (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: 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, W/E)
    D°M'S.SS"N/S D°M'S.SS"W/E
    サンプル: **33°55'11.11"N 22°44'66.66"W**

  - 度、分、 10進の秒、プレフィックスサイン付き (+/-); (**N/W**) 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"**

  - 度、分、 10進の分、方角のサフィックス付き (N/S, W/E)
    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/W**) 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, W/E)
    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/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°44'66.66"
33.33  22°44'66.66"W
33.33  22.c
```

- *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

[Top]

10.1.4 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+XBRL 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

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ご注意ください。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**
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**

suggested-image-file-extension [altova:]

altova:suggested-image-file-extension(Base64String *as string*) を **string?** とする
XP3 **XQ3**
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 **XQ3**
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。

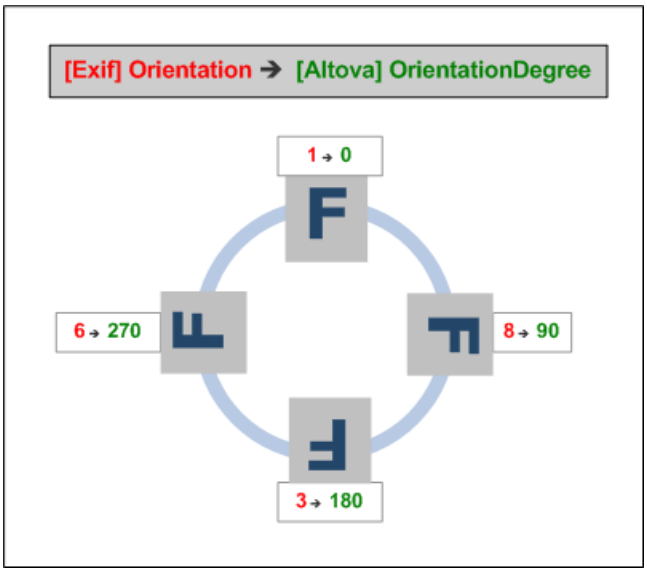
サンプル

- **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 XQ3**

Base64 エンコード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。The 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

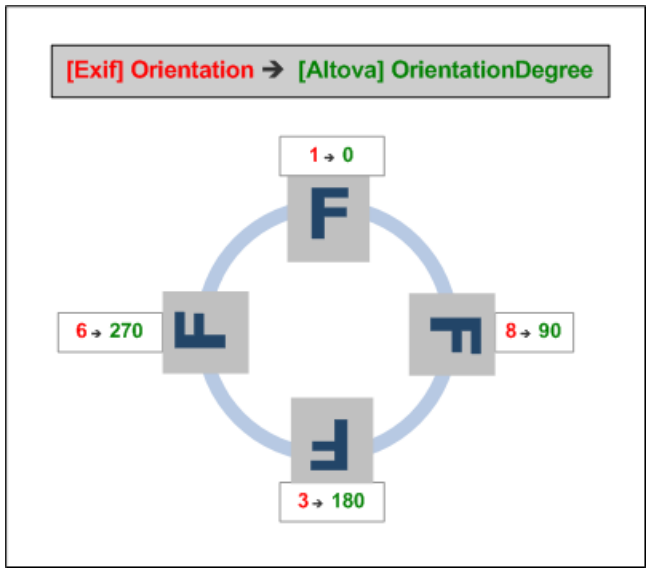
Altova XPath/XQuery エンジンはカスタム属性 **Geolocation** を標準 Exif メタデータタグから生成します。**Geolocation** は、4 つの Exif タグの連結です :単位の追加された (下のテーブル参照) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef。

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]

10.1.5 XPath/ XQuery 関数 :数値

Altova の数値拡張関数は 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

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ご注意ください。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
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

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 XQ3

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` 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。starting with `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` 関数の呼び出しにより `altova:reset-auto-number("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 XQ3

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`

▫ *サンプル*

- **altova:reset-auto-number**`("ChapterNumber")` は、`altova:generate-auto-number` 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` 関数の `StartsWith` 引数の

値に数値をリセットします。

[Top]

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 XQ3

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 XQ3

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]

数値をフォーマットする関数

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 XQ3

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の StartsWith 値にリセットされます。自動付番は altova:reset-auto-number 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。starting with 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 関数の呼び出しによりの altova:reset-auto-number("ChapterNumber") ようにリセットされます。

[トップ]

10.1.6 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

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ご注意ください。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**
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**

attributes [altova:]

`altova:attributes(AttributeName as xs:string)` を **`attribute()*`** とする **XP3 XQ3**
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)` as **`attribute()*`** **XP3 XQ3**
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グを含みます。使用することのできるフラ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。無効なフラグはエラーを生成します。1 つまたは複数のフラ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 XQ3
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*) as **element()*** XP3 XQ3
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グを含みます。使用することのできるフラ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。無効なフラグはエラーを生成します。1つまたは複数のフラ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 XQ3
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 XQ3**

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 つずつに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 XQ3

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 XQ3

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 XQ3**

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)) returns 1

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)) returns 2

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。

substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()***)** を **item()*** とする **XP3** **XQ3**

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。

10.1.7 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

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ご注意ください。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**
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**

camel-case [altova:]

altova:camel-case(InputString *as xs:string***)** を**xs:string** とする **XP3** **XQ3**
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** **XQ3**
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** **XQ3**

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 XQ3**

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。

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer*) を**xs:string** とする **XP3 XQ3**

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 XQ3**

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。

last-chars [altova:]

altova:last-chars(X-Number *as xs:integer*) を**xs:string** とする **XP3 XQ3**

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 XQ3**
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 XQ3**
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 内の文字数よりも多い場合の場合のみ発生します。もし、 StringToPad が StringLength の値よりも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 XQ3**
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 内の文字数よりも多い場合の場合のみ発生します。もし、 StringToPad が StringLength の値よりも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 XQ3**
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** **XQ3**

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** **XQ3**

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** **XQ3**

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') 3 を返します。
- **altova:substring-pos**('Altova', 'tov') 3 を返します。
- **altova:substring-pos**('Altova', 'tv') returns 0 を返します。

- `altova:substring-pos('AltovaAltova', 'to') 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 XQ3**
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) 3` を返します。
- `altova:substring-pos('Altova', 'to', 3) 3` を返します。
- `altova:substring-pos('Altova', 'to', 4) 0` を返します。
- `altova:substring-pos('Altova-Altova', 'to', 0) 3` を返します。
- `altova:substring-pos('Altova-Altova', 'to', 4) 10` を返します。

trim-string [altova:]

altova:trim-string(InputString *as xs:string*) を**xs:string** とする **XP3 XQ3**
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 XQ3**
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 XQ3**
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。

10.1.8 XPath/ XQuery 関数 :その他

以下の一般使用のためのXPath/XQuery 拡張関数は、RaptorXML+XBRL 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

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ご注意ください。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
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

URI 関数

▼ **get-temp-folder [altova:]**

altova:get-temp-folder() を `xs:string` とする XP2 XQ1 XP3 XQ3
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。

[Top]

10.1.9 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
- チャートの作成を行う関数

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`binarytobase64` または`binarytobase16` から選択することができます。

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`base64Binary` または`hexBinary` 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に注意してください。*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に注意してください。*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: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。

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。

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に注意してください。

```
<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
      PltBorderColor="#000000" Color
      PltBKColor="#ffffff" Color
      Title="" String
      ShowLegend="1" Bool
      OutsideMargin="3%" PercentOrPixel
      TitleToPltMargin="3%" PercentOrPixel
      LegendToPltMargin="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="10pt" 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>
```

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="32%"
                                    TitleToPlotMargin="3%"
                                    LegendToPlotMargin="6%"
                                    >
                                    <TitleFont
                                        Color="#023d7d"
```

```
                                    Name="Tahoma"
                                    Bold="1"
                                    Italic="0"
                                    UnderLine="0"
                                    MinFontHeight="10pt"
                                    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>
```

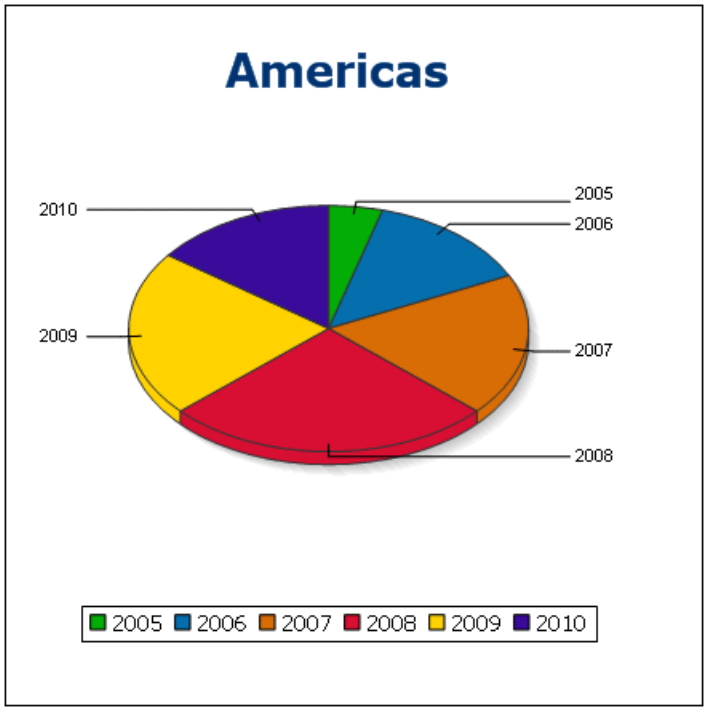
XML ドキュメント

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



10.1.10 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\Common2017\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\Common2017\jar フォルダーのサブフォルダー以下に収められてます：

- 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。

---

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('Code39','string('some value'),
        96,0,(altovaext-barcode-property:new('setModuleWidth',25.4 div 96 * 2)))"/>
      <xsl:value-of select="xs:base64Binary(xs:hexBinary(string(altovaext-
barcode:generateBarcodePngAsHexString($barcodeObject))))"/>
    </xsl:result-document>
  </xsl:template>
</xsl:stylesheet>
```

10.2 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()` という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 や .NET および MSXSL scripts for XSLT における拡張関数の使用がサポートされます。また、XSLT に対する MSXSL スクリプトもサポートします。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
- .NET 拡張関数
- XSLT に対する XBRL 関数
- XSLT に対する MSXSL スクリプト

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。

10.2.1 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
- Java :静的メソッドと静的フィールド
- Java : *インスタンス*メソッドと*インスタンス*フィールド
- データ型 : XPath/XQuery からJava へ
- データ型 : Java からXPath/XQuery へ

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ます。

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:Math="java:java.lang.Math"
              select="Math:cos(3.14)" />

<xsl:value-of xmlns:math="java"
              select="math: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ならびにユーザー定義 JAR ファイルを参照ください。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にご注意してください。

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)
- 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)
- 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)
- 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)

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 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

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>
```

クラスファイルがパッケージではなく、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

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。Car クラスファイルは `JavaProject/com/altova/extfunc` 以下に配置されています。XSLT ファイルも `JavaProject/com/altova/extfunc` 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: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で、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

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 xs xsl"/>

<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 または 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 xs xsl"/>

<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。

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になります。
`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 xs xsl"/>

<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。

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、暗示的に XPath/XQuery データ型へ変換できる場合、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*を参照ください）：
  ```
  <xsl:value-of select="date:toString(date:new ())" xmlns:date="java:java.util.Date" />
  ```

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>
```

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` を持っ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。

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。

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 というような) 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。

10.2.2 .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
- .NET 静的メソッドならびに静的フィールド
- .NET : インスタンスメソッドとインスタンスフィールド
- データ型 : XPath/XQuery から .NET へ
- データ型 : .NET から XPath/XQuery へ

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 桁数の数値)。
`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。

.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*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" />
```

.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>
```

.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。

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か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絞められます。
- 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10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。

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。

10.2.3 XSLT に対する XBRL 関数

XBRL 関数レジストリで定義されている関数は、XBRL インスタンス ドキュメントを変換するための XSLT コンテキスト内から呼び出すことができます。これらの関数は以下の 2 つのうちの 1 つの名前空間で定義されています :

`http://www.xbrl.org/2008/function/instance` (通常 `xfi:` プレフィックスと共に使用されています)
`http://www.xbrl.org/2010/function/formula` (通常 `xff:` プレフィックスと共に使用されています)

XBRL 関数 `xfi:context` は、たとえば、に展開することができます `http://www.xbrl.org/2008/function/instance:context` (`xfi:` プレフィックスにバインドされた名前空間を持つと仮定します。)

関数の完全なリストを見るには、以下に移動してください http://www.xbrl.org/functionregistry/functionregistry.xml。

10.2.4 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が <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。

例

<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 retailprice: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>TotalRetailPrice =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>TotalWholesale 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で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。

チャプター 11

Altova LicenseServer

11 Altova LicenseServer

Altova LicenseServer (今後は略して**LicenseServer** と称されます)は、Altova 製品のライセンスを集中して管理する場所です。ネットワークで作動するAltova アプリケーションはLicenseServer からライセンスを割り当てられます、ですから、管理者はライセンスを管理及び監視する柔軟性を有します。

現在のバージョン:2.3

Altova LicenseServer ライセンスのプロセス

LicenseServer を介して、Altova サーバー製品にライセンスを割り当てるには、以下の手順を踏みます:

1. LicenseServer の開始
2. LicenseServer のWeb UI であるLicenseServer 構成ページを開きます。 Windows、Linux、またはMac OS X
3. Altova to LicenseServer から受け取ったサーバー製品ライセンスをアップロードする。構成ページ内のライセンスプールタブで行います。
4. LicenseServe でAltova サーバー製品 (FlowForce Server、MapForce Server、StyleVision Server、RaptorXML(+XBRL) Server) の登録を行います。
5. 構成ページのクライアント管理 タブでAltova サーバーへのライセンスの割り当てを行います。

今後、ライセンスは便利にLicenseServer で集中して監視および管理することができます。使用可能な機能については構成ページ レファレンスを参照してください。

メモ: LicenseServer 構成ページはSSL をサポートしません。

LicenseServer のバージョンと他の Altova 製品との互換性

Altova サーバー製品の新しいバージョンは、サーバ製品のリリース時に最新のバージョンであるLicenseServer のバージョンによりのみライセンスを受けることができます。ですが、Altova サーバー製品の古いバージョンは新しいバージョンのLicenseServer と作動することができます。

ですから、新しいバージョンのAltova サーバー製品をインストールする場合、現在のLicenseServer のバージョンが最新でない場合、この古いLicenseServer バージョンをアンインストールし、Altova Web サイトで利用可能な最新バージョンをインストールしてください。古いバージョンのLicenseServer の全ての登録およびライセンス情報は、アンインストール時にサーバーマシンのデータベースに保存され、新しいバージョンに自動的にインポートされます。新しいバージョンのLicenseServer をインストールする際は、古いバージョンを新しいバージョンをインストールするまでにアンインストールします。

現在インストールされているLicenseServer のバージョンは、LicenseServer 構成ページ(全てのタブ)の下部に表示されます。

現在のバージョン:2.3

このドキュメントについて

このドキュメントは、以下のパートに整理されています:

- 以下についての基本情報:ネットワークの必要条件、Windows、Linux 、およびMac OS Xへのインストール方法、およびAltova ServiceController。
- ライセンスの割り当ての方法は、Altova LicenseServer を使用する順序を追ったライセンスの割り当ての方法を説明しています。

- 構成ページのレファレンス: LicenseServer での管理者のインターフェイスの説明。

最終更新日 : 2017年 04月 27日

11.1 ネットワーク情報

Altova LicenseServer は、ライセンスを必要とするAltova 製品が作動するすべてのクライアントからアクセスできるサーバーマシンにインストールされている必要があります。クライアントとサーバのファイアウォールは、LicenseServer が正しく作動するために必要なLicenseServer からへのネットワークトラフィックのフローを許可しなければなりません。

LicenseServer マシンでは**ポート 35355**がライセンス配布用に使用されます。ですので、クライアントマシンとネットワークトラフィックのために開かれている必要があります。

以下がLicenseServer のデフォルトのネットワークパラメータおよび必要条件です：

- *LicenseServer ライセンス配布用：*
  以下の一方または両方
  IPv4 TCP 接続　ポート `35355`
  IPv6 TCP 接続　ポート `35355`

管理タスクに関しては、LicenseServer はポート 8088 を使用するWeb インターフェイスからアクセスできます。使用するポートに関しては条件に合った構成を参照してください。

altova.com のマスターライセンスサーバーへの接続
Altova LicenseServer は、ライセンスに関連したデータを検証と認証し、Altova ライセンス使用許諾契約書への継続的な遵守を確認するため、`altova.com` のマスターLicensing Server と通信する必要があります。この通信はHTTPS を介して、ポート 443 を使用して行われます。`altova.com` のマスターLicensing Server との最初の検証の後、Altova LicenseServer が`altova.com` と 5 日間 (= 120 時間) 再接続できない場合、Altova LicenseServer はAltova LicenseServer に接続してAltova ソフトウェア製品を使用することを許可しません。

Altova マスターサーバーへの接続損失はAltova LicenseServer の構成ページのメッセージ (Messages) タブにログされます。更に、管理者は、`altova.com` への接続が失われた場合、自動的に警告の電子メールを送信するようにAltova LicenseServer を構成することができます。電子メールの設定の変更は、構成ページの設定 タブで行うことができます。

11.2 インストール (Windows)

Altova LicenseServer はWindows システムに 2通りの方法でインストールすることができます :

- 独立したインストール
- Altova サーバー製品の一部としてのインストール。 (Altova サーバー製品 :Altova FlowForce Server、Altova MapForce Server、Altova StyleVision Server、Altova RaptorXML(+XBRL) および Altova MobileTogether Server)。Altova サーバー製品をインストールする際、LicenseServer がシステムにインストールされていない場合、LicenseServer のインストールのオプションはインストールセットアップ中にデフォルトで選択されます。LicenseServer が既にインストールされている場合、インストールするオプションは解除されます。デフォルトのオプションは変更可能です。

LicenseServer を使用して、ライセンスを割り当てる方法に関しての情報は、ライセンスの割り当て方法セクションを参照してください。

システムの必要条件

▾ *Windows*

Windows Vista, Windows 7/8/10

▾ *Windows Server*

Windows Server 2008 R2 または以降

▾ LicenseServer のバージョンと他の Altova 製品との互換性

Altova サーバー製品の新しいバージョンは、サーバ製品のリリース時に最新のバージョンであるLicenseServer のバージョンによりのみライセンスを受けることができます。ですが、Altova サーバー製品の古いバージョンは新しいバージョンのLicenseServer と作動することができます。

ですから、新しいバージョンのAltova サーバー製品をインストールする場合、現在のLicenseServer のバージョンが最新でない場合、この古いLicenseServer バージョンをアンインストールし、Altova Web サイトで利用可能な最新バージョンをインストールしてください。古いバージョンのLicenseServer の全ての登録およびライセンス情報は、アンインストール時にサーバーマシンのデータベースに保存され、新しいバージョンに自動的にインポートされます。新しいバージョンのLicenseServer をインストールする際は、古いバージョンを新しいバージョンをインストールするまでにアンインストールします。

現在インストールされているLicenseServer のバージョンは、LicenseServer 構成ページ (全てのタブ) の下部に表示されます。

現在のバージョン: 2.3

サーバー製品の特定のバージョンに適切なLicenseServer のバージョン番号がのインストールプロセスの最中に表示されます。サーバ製品と共にこのバージョンのLicenseServer をインストールすることができます、また、新しいバージョンのLicenseServer を個別にインストールすることもできます。どちらのケースの、インストーラーは前のバージョンをアンインストールして、新しいバージョンをインストールします。

11.3 インストール (Linux)

Altova LicenseServer はLinux システム (Debian、Ubuntu、CentOS、RedHat) にインストールすることができます。

システムの必要条件

- *Linux*
  - CentOS 6 または以降
  - RedHat 6 または以降
  - Debian 7 または以降
  - Ubuntu 12.04 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+XBRL Server	qt4, krb5-libs, qt-x11	libqtcore4, libqtgui4, libgssapi-krb5-2	libqtcore4, libqtgui4, libgssapi-krb5-2

古いバージョン LicenseServer のアンインストール

Linux コマンドラインインターフェイス (CLI) で以下のコマンドを使用してLicenseServer がインストールされているか確認することができます:

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

LicenseServer がインストールされていない場合、以下のステップでインストールしてください。LicenseServer がインストールされていて、新しいバージョンをインストールしたい場合、以下のコマンドを使用して古いバージョンをアンインストールしてください:

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

Altova LicenseServer のインストール

Linux システムでは、LicenseServer は他のAltova サーバー製品と別途にインストールされる必要があり、Altova サーバー製品のインストールパッケージには含まれていません。 Altova Web サイトからAltova LicenseServer をダウンロードして、直接 Linux システムのディレクトリにパッケージをコピーします。

ディストリビューション	インストーラー拡張子

Debian	.deb
Ubuntu	.deb
CentOS	.rpm
RedHat	.rpm

ターミナルウィンドウで、Linux パッケージをコピーしたディレクトリに切り替えます。例えば (/home/User ディレクトリに存在する)、MyAltova という名のユーザーディレクトリにコピーした場合、以下のように切り替えます :

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

以下のコマンドを使用して LicenseServer をインストールします :

```
[Debian]:   sudo dpkg --install licenseserver-2.3-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-2.3-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-2.3-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-2.3-1.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
```

ライセンスの割り当ての手順に関しては、ライセンスの割り当て方法のセクションを参照してください。

LicenseServer のバージョンと他の Altova 製品との互換性

Altova サーバー製品の新しいバージョンは、サーバ製品のリリース時に最新のバージョンである LicenseServer のバージョンによりのみライセンスを受けることができます。ですが、Altova サーバー製品の古いバージョンは新しいバージョンの LicenseServer と作動することができます。

ですから、新しいバージョンの Altova サーバー製品をインストールする場合、現在の LicenseServer のバージョンが最新でない場合、この古い LicenseServer バージョンをアンインストールし、Altova Web サイトで利用可能な最新バージョンをインストールしてください。古いバージョンの LicenseServer の全ての登録およびライセンス情報は、アンインストール時にサーバーマシンのデータベースに保存され、新しいバージョンに自動的にインポートされます。新しいバージョンの LicenseServer をインストールする際は、古いバージョンを新しいバージョンをインストールするまでにアンインストールします。

現在インストールされている LicenseServer のバージョンは、LicenseServer 構成ページ (全てのタブ) の下部に表示されます。

現在のバージョン: 2.3

11.4 インストール (Mac OS X)

Altova LicenseServer は Mac OS X システム (バージョン 10.8 または以降) にインストールすることができます。前のバージョンがアンインストールする必要がある場合は、アンインストールを先に行ってください。

システムの必要条件

- *Mac OS X*

  OS X 10.10、10.11、macOS 10.12 または以降

古いバージョン LicenseServer のアンインストール

LicenseServer をアンインストールする前に、以下のコマンドでサービスを停止します :

```
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。

アプリケーションで、LicenseServer 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/LicenseServer
```

Altova LicenseServer のインストール

ダウンロードページ http://www.altova.com/ja/download.html を開き、Mac のためのサーバーソフトウェア製品の中から Altova LicenseServer を検索します。イメージ (`.dmg`) ファイルをダウンロード後、クリックして開きます。これにより、新しい仮想ドライブがコンピューターにマウントされます。仮想ドライブで、パッケージ (pkg) ファイルをダブルクリックして、画面上の指示に従います。手続きを続行するには、使用許可承諾書に同意する必要があります。

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
```

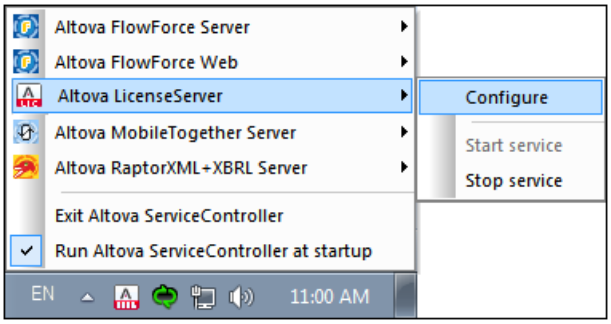
インストール後仮想ドライブを取り出すには、右クリックして、**「取り出し」**を選択します。

11.5 Altova ServiceController

Altova ServiceController (略してServiceController) は**Windows 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) と共にインストールされます。**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*Altova サーバー製品がサービスとしてインストールされている* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server) **スタート**メニューフォルダーでも利用可能です。) 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*)。



サービスの開始と停止

インストールされたAltova サービスコンポーネントはServiceController メニューでエントリとして表示されます (*上部スクリーンショット参照*)。Altova サービスはServiceController のサブメニューのコマンドを介して開始または停止することができます。更に、ServiceController メニューを介して、個別サービスの管理タスクにアクセスすることができます。上部のスクリーンショットでは、例えば、Altova LicenseServer サービスにはサブメニューがあり、**「構成」 (Configure)** コマンドを介してLicenseServer の構成ページにアクセスすることを選択できます。

11.6 ライセンスの割り当て方法

Altova LicenseServer を使用して、Altova 製品にライセンスを与えるには以下の手順を踏んでください:

1. LicenseServer の開始
2. Windows、Linux、または Mac OS X で LicenseServer の管理者のインターフェイスである LicenseServer 構成ページを開きます。
3. Altova から、Altova LicenseServer のライセンスプールへ受信されたライセンスをアップロードします。LicenseServer 構成ページのライセンス プール (License Pool) タブから行います。
4. Altova サーバー製品 (FlowForce Server、MapForce Server、StyleVision Server、RaptorXML(+XBRL) Server) を LicenseServer で登録します。製品の種類により、LicenseServer への登録方法は異なります:製品の Web UI または コマンドラインを介しての登録。詳細に関しては、Altova サーバー製品のドキュメンテーションを参照してください。
5. LicenseServer 構成ページのクライアント管理 タブで、Altova 製品にライセンスの割り当て を行うことができます。

コアとライセンスについてのメモ
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。

MobileTogether Server ライセンス
MobileTogether Server ライセンスには2つの種類があります。カスタマーは必要に応じてライセンスの種類を選択することができます。

- *コアライセンス*:サーバーマシンのコア数をベースにして MobileTogether Servers に割り当てられます。上の例を参照してください。上の説明を参照してください。コアライセンスは、無制限の数量の MobileTogether クライアントデバイスによりサーバーへの接続を許可します。しかしながら、「単一スレッドの実行」チェックボックスがチェックされていると、1度に MobileTogether Server に接続できるモバイルデバイスは1台です。これは、評価と小さい規模のテストを行う際に役に立ちます。この場合、2台目のモバイルデバイスが MobileTogether Sever に接続される場合、ライセンスは2台目が使用するようになります。最初のデバイスは、接続できないようになり、エラーメッセージが表示されます。
- *デバイスライセンス*: MobileTogether Server にいつでも接続することのできる MobileTogether Client デバイスの最高使用数を指定します。

11.6.1 LicenseServer の開始

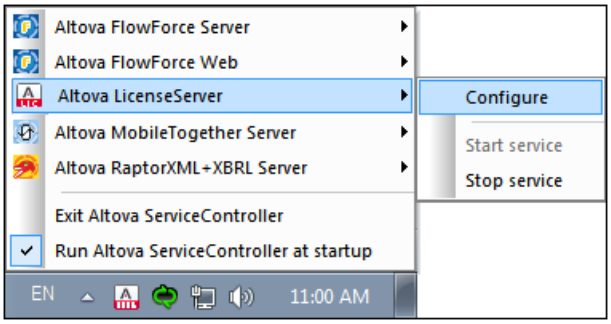
このセクション:

- Windows システム での LicenseServer の開始方法
- Linux システム での LicenseServer の開始方法
- Mac OS X システム での LicenseServer の開始方法
- altova.com への接続 についてのメモ

Windows システム

システムトレイにある Altova ServiceController を介して、LicenseServer を開始します。

最初に、**「スタート | すべてのプログラム | Altova LicenseServer | Altova ServiceController」**をクリックして、Altova ServiceController を開始して、システムトレイのアイコンを表示します (*下のスクリーンショット参照*)。*スタートアップオプションで Altova ServiceController の実行*を選択すると、Altova ServiceController が開始し、システムトレイにアイコンが利用可能になります。



LicenseServer を開始するには、システムトレイのサービスコントローラー (ServiceController) アイコンをクリックします。ポップアップしたメニューの **Altova LicenseServer** をポイントして、(*下のスクリーンショット参照*)、LicenseServer サブメニューから **「サービスの開始」 (Start Service)** 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。

Linux システム

LicenseServer をサービスとして Linux システムで開始するには、ターミナルウィンドウで以下のコマンドを実行します :

```
[Debian 7]:          sudo /etc/init.d/licenseserver start
[Debian 8]:          sudo systemctl start licenseserver
[Ubuntu <=14]:       sudo initctl start licenseserver
[Ubuntu 15]:         sudo systemctl start licenseserver
[CentOS 6]:          sudo initctl start licenseserver
[CentOS 7]:          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。)

Mac OS X システム

LicenseServer をサービスとして Mac OS X システムで開始するには、ターミナルウィンドウで以下のコマンドを実行します：

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

LicenseServer を停止する必要がある場合、以下を使用します：

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

---

altova.com のマスターライセンスサーバーへの接続

Altova LicenseServer は、ライセンスに関連したデータを検証と認証し、Altova ライセンス使用許諾契約書への継続的な遵守を確認するため、`altova.com` のマスター Licensing Server と通信する必要があります。この通信は HTTPS を介して、ポート 443 を使用して行われます。`altova.com` のマスター Licensing Server との最初の検証の後、Altova LicenseServer が `altova.com` と 5 日間 (= 120 時間) 再接続できない場合、Altova LicenseServer は Altova LicenseServer に接続して Altova ソフトウェア製品を使用することを許可しません。

Altova マスターサーバーへの接続損失は Altova LicenseServer の構成ページの メッセージ (Messages) タブ にログされます。更に、管理者は、`altova.com` への接続が失われた場合、自動的に警告の電子メールを送信するように Altova LicenseServer を構成することができます。電子メールの設定の変更は、構成ページの 設定 タブ で行うことができます。

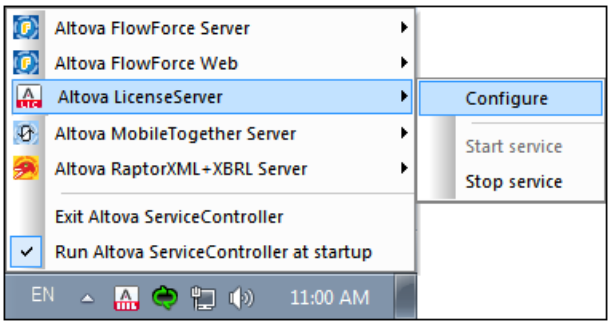
11.6.2 LicenseServer の構成ページの開きかた (Windows)

このセクション:

LicenseServer が同じコンピューターにある場合の構成ページの開きかた

Windows システムで、LicenseServer が既にコンピュータにある場合、LicenseServer の構成ページを 2通りの方法で開くことができます :

- **「スタート | すべてのプログラム | Altova LicenseServer | LicenseServer 構成ページ (Configuration Page)」** をクリックします。構成ページはインターネットブラウザーの新しいタブとして開かれます。
- システムトレイの Altova ServiceController アイコンをクリックします。ポップアップしたメニューの **Altova LicenseServer** (*下のスクリーンショット参照*) をポイントして **「構成」 (Configure)** を LicenseServer サブメニューから選択します。



構成ページは新しいブラウザーウィンドウで開かれ、ログインマスクが表示されます (*下のスクリーンショット*)。

LicenseServer が他のコンピューターにある場合の設定ページの開きかた

LicenseServer 構成ページをローカルネットワークの LicenseServer がインストールされている他の Windows マシーンから開く場合、ブラウザーのアドレスバーに LicenseServer 構成ページ URL を入力して、 **「Enter」** を押します。構成ページのデフォルトの URL 以下の通りです :

```
http://<serverIPAddressOrName>:8088/
```

構成ページ自身の HTML コードで示された `WebUI.html` という名前の URL は以下で見つけることができます :

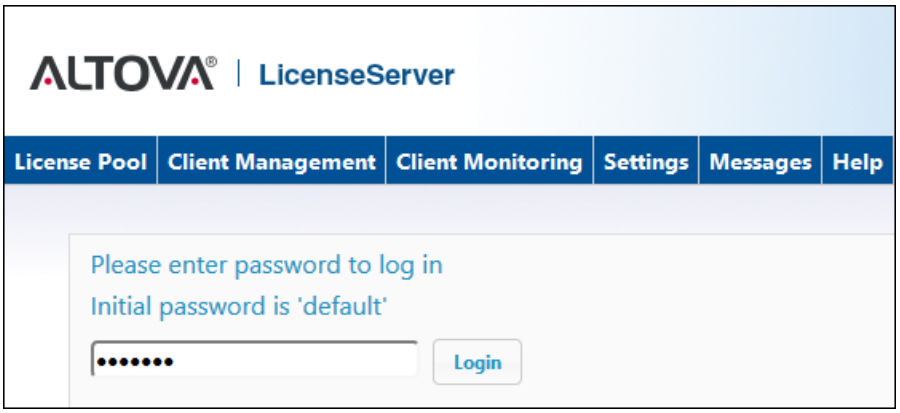
```
C:/ProgramData/Altova/LicenseServer/WebUI.html
```

構成ページの URL の設定を動的に生成した場合、 (構成ページの設定タブで) LicenseServer を開始する都度、新しい URL が生成されます。`WebUI.html` の現在のバージョンをチェックして、構成ページの 現在の URL を確認してください。

WebUI.html 内で動的に生成されたURL は以下のようなフォームで表示されます：
`http://127.0.0.1:55541/optionally-an-additional-string`、 `<head>` 要素の終わり近くのスクリプト内の関数`checkIfServiceRunning()`にあります。URL 内のポート番号のみが動的に割り当てられますが、IP アドレスは部分的にLicenseServer がインストールされたサーバーを識別します。LicenseServer 構成ページを他のマシーンからアクセスする場合、URL のIP アドレスがLicenseServer がインストールされているサーバーの正確なIP アドレスまたは名前であることを確認してください。例えば、URL は以下のようになります：`http://SomeServer:55541`。

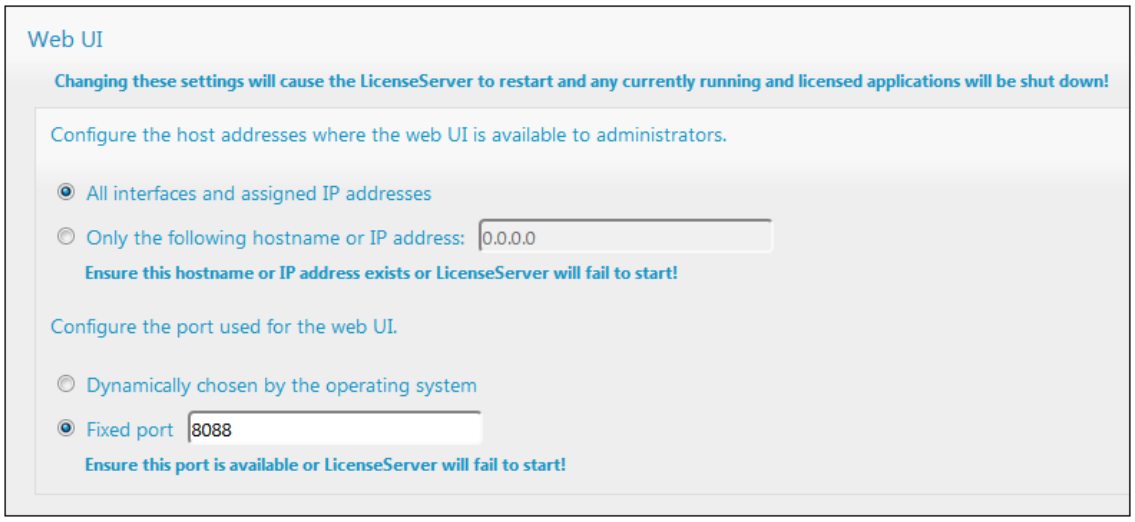
初回パスワードでのログイン

上記のステップを踏んだ後、構成ページのログインマスクが表示されます (下のスクリーンショット)。初回パスワード `default` でログインすることができます。ログインした後、設定 (Settings) タブでパスワードを変更することができます。



構成ページの固定または動的ポートの設定

構成ページ (Web UI)のポート? と結果的にアドレス? は設定 (Settings) ページにて指定することができます。デフォルトのポートは `8088` です。LicenseServer 構成ページ (下のスクリーンショット参照)の他のポートを設定することもできます。また、LicenseServer が開始されるたびにポートを動的に選択することも許可されています。この場合、構成ページのURL をファイル`WebUI.html`から検索する必要があります。 (LicenseServer 構成ページ (Windows) を開く、LicenseServer 構成ページを開く (Linux) とLicenseServer 構成ページ (Linux) を開くを参照してください)。



固定ポートの利点は、ページURL が事前に把握することができ、そのため、簡単にアクセスすることができます。ポートが動的に割り当てられる場合、URL のポートのパートはLicenseServer が開始されるたびにをファイル`WebUI.html`から検索される必要があります。

11.6.3 LicenseServer の構成ページの開きかた (Linux)

このセクション:

- 返されたURL で構成ページを初めて開く
- LicenseServer 構成ページのURL
- 初回パスワードでのログイン
- ページ構成ページの固定ポートの設定

返されたURL で構成ページを初めて開く

Linux システムでは、CLI を介してLicenseServer にAltova サーバー製品を登録した場合、LicenseServer の構成ページのURL が返されます。ブラウザーでこのURL を開く際、ライセンス使用許諾契約書を読んで合意するようにプロンプトされます。ライセンス使用許諾契約書に合意した後、構成ページのログインマスクが表示されます (*下のスクリーンショット*)。

メモ: Altova デスクトップ製品は、Windows のみで使用することができます。

LicenseServer 構成ページのURL

LicenseServer 構成ページ開くには、アドレスバーにURL を入力して、「Enter」を押します。構成ページのデフォルトのURL は以下の通りです:

```
http://<serverIPAddressOrName>:8088/
```

構成ページ自身のHTML コードで示された`WebUI.html` という名前のURL は以下で見つけることができます:

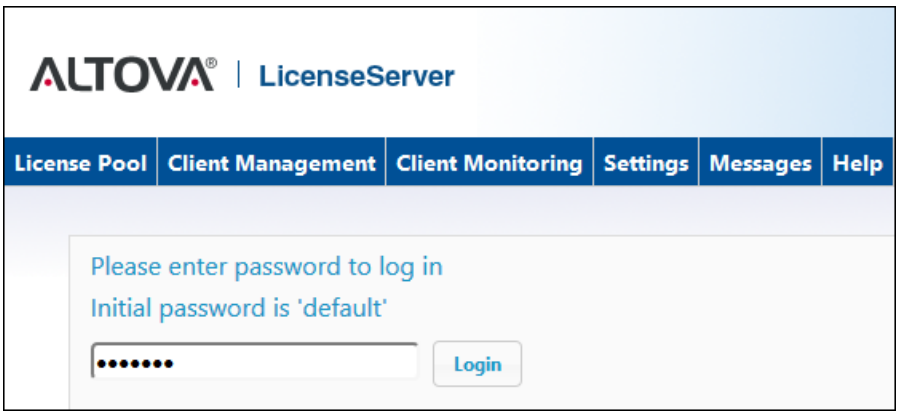
```
/var/opt/Altova/LicenseServer/webUI.html
```

構成ページのURL の設定 を動的に生成した場合、(構成ページの設定タブで)、LicenseServer を開始する都度、新しいURL が生成されます。`WebUI.html` の現在のバージョンをチェックして、構成ページの 現在のURL を確認してください。

`WebUI.html` 内で動的に生成されたURL は以下のようなフォームで表示されます:
`http://127.0.0.1:55541`。`<head>` 要素の終わり近くのスクリプト内の関数`checkIfServiceRunning()` にあります。URL 内のポート番号のみが動的に割り当てられますが、IP アドレスは部分的にLicenseServer がインストールされたサーバーを識別します。LicenseServer 構成ページを他のマシーンからアクセスする場合、URL のIP アドレスがLicenseServer がインストールされているサーバーの正確なIP アドレスまたは名前であることを確認してください。例えば、URL は以下のようになります:`http://MyServer:55541`。

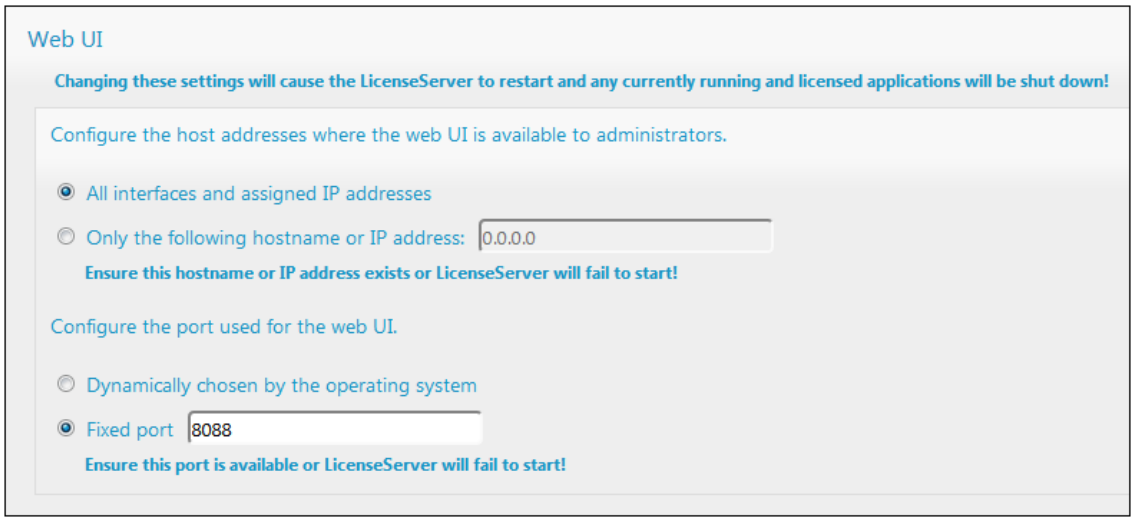
初回パスワードでのログイン

上記のステップを踏んだ後、構成ページのログインマスクが表示されます (*下のスクリーンショット*)。初回パスワード `default` でログインすることができます。ログインした後、設定 (Settings) タブでパスワードを変更することができます。



構成ページの固定または動的ポートの設定

構成ページ (Web UI) のポート? と結果的にアドレス? は設定 (Settings) ページにて指定することができます。デフォルトのポートは 8088 です。LicenseServer 構成ページ (*下のスクリーンショット参照*) の他のポートを設定することもできます。また、LicenseServer が開始されるたびにポートを動的に選択することも許可されています。この場合、構成ページの URL をファイル `WebUI.html` から検索する必要があります。 (LicenseServer 構成ページ (Windows) を開く、LicenseServer 構成ページを開く (Linux) と LicenseServer 構成ページ (Linux) を開くを参照してください)。



固定ポートの利点は、ページ URL が事前に把握することができ、そのため、簡単にアクセスすることができます。ポートが動的に割り当てられる場合、URL のポートのパートは LicenseServer が開始されるたびにをファイル `WebUI.html` から検索される必要があります。

11.6.4 LicenseServer の構成ページの開きかた (Mac OS X)

このセクション:

返されたURL で構成ページを初回開く

Mac OS X システムでは、CLI を介してLicenseServer にAltova サーバー製品を登録した場合、LicenseServer の構成ページのURL が返されます。ブラウザーでこのURL を開く際、ライセンス使用許諾契約書を読んで合意するようにプロンプトされます。ライセンス使用許諾契約書に合意した後、構成ページのログインマスクが表示されます (*下のスクリーンショット*)。

メモ: Altova デスクトップ製品は、Windows のみで使用することができます。

LicenseServer 構成ページのURL

LicenseServer 構成ページ開くには、アドレスバーにURL を入力して、**「Enter」** を押します。構成ページのデフォルトのURL は以下の通りです:

```
http://<serverIPAddressOrName>:8088/
```

構成ページ自身のHTML コードで示された`WebUI.html` という名前のURL は以下で見つけることができます:

```
/var/Altova/LicenseServer/webUI.html
```

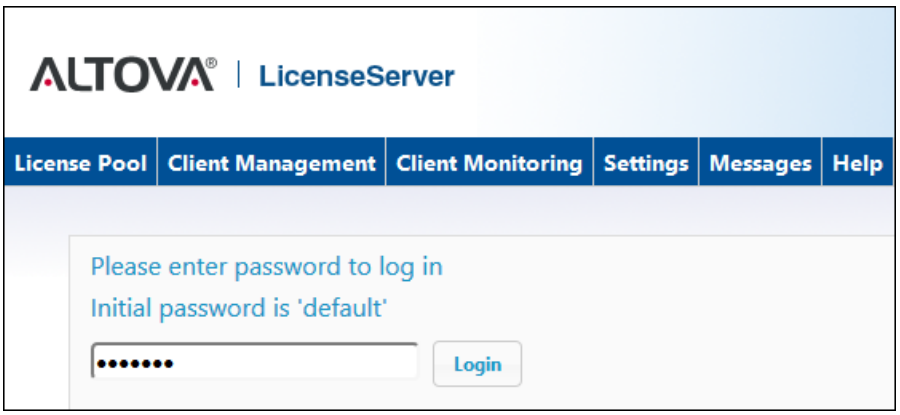
構成ページのURL の設定 を動的に生成した場合、(構成ページの設定タブで)、LicenseServer を開始する都度、新しいURL が生成されます。`WebUI.html` の現在のバージョンをチェックして、構成ページの 現在のURL を確認してください。

`WebUI.html` 内で動的に生成されたURL は以下のようなフォームで表示されます:
`http://127.0.0.1:55541`。`<head>` 要素の終わり近くのスクリプト内の関数`checkIfServiceRunning()` にあります。URL 内のポート番号のみが動的に割り当てられますが、IP アドレスは部分的にLicenseServer がインストールされたサーバーを識別します。LicenseServer 構成ページを他のマシーンからアクセスする場合、URL のIP アドレスがLicenseServer がインストールされているサーバーの正確なIP アドレスまたは名前であることを確認してください。例えば、URL は以下のようになります:`http://MyServer:55541`。

メモ: 構成ページはまた、**「ファインダー | アプリケーション | Altova License Server」** アイコンを介してアクセスすることができます。

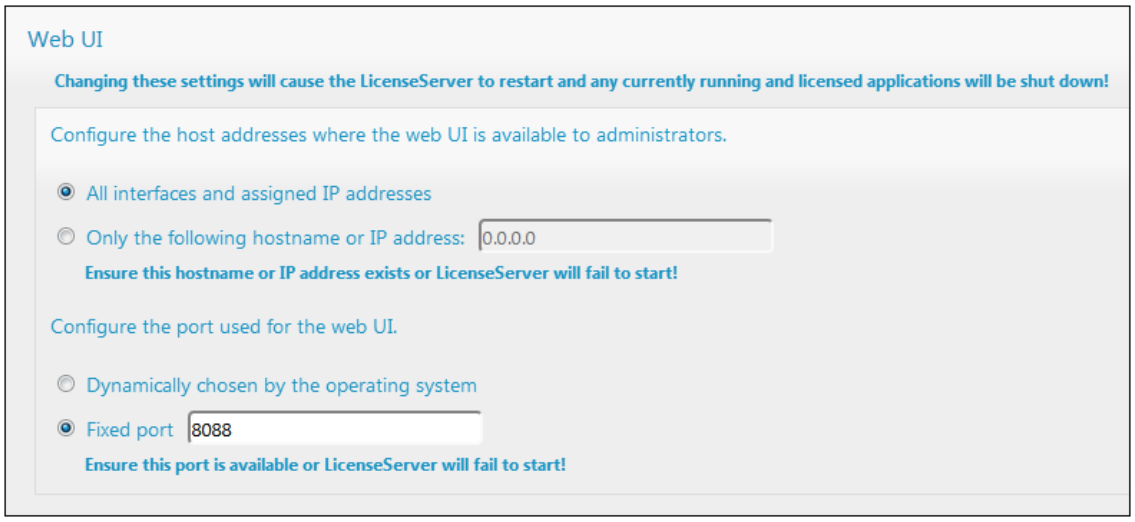
初回パスワードでのログイン

上記のステップを踏んだ後、構成ページのログインマスクが表示されます (*下のスクリーンショット*)。初回パスワード `default` でログインすることができます。ログインした後、設定 (Settings) タブでパスワードを変更することができます。



構成ページの固定または動的ポートの設定

構成ページ (Web UI) のポート? と結果的にアドレス? は設定 (Settings) ページにて指定することができます。デフォルトのポートは 8088 です。LicenseServer 構成ページ (*下のスクリーンショット参照*) の他のポートを設定することもできます。また、LicenseServer が開始されるたびにポートを動的に選択することも許可されています。この場合、構成ページの URL をファイル WebUI.html から検索する必要があります。 (LicenseServer 構成ページ (Windows) を開く、LicenseServer 構成ページを開く (Linux) と LicenseServer 構成ページ (Linux) を開くを参照してください)。



固定ポートの利点は、ページ URL が事前に把握することができ、そのため、簡単にアクセスすることができます。ポートが動的に割り当てられる場合、URL のポートのパートは LicenseServer が開始されるたびにをファイル WebUI.html から検索される必要があります。

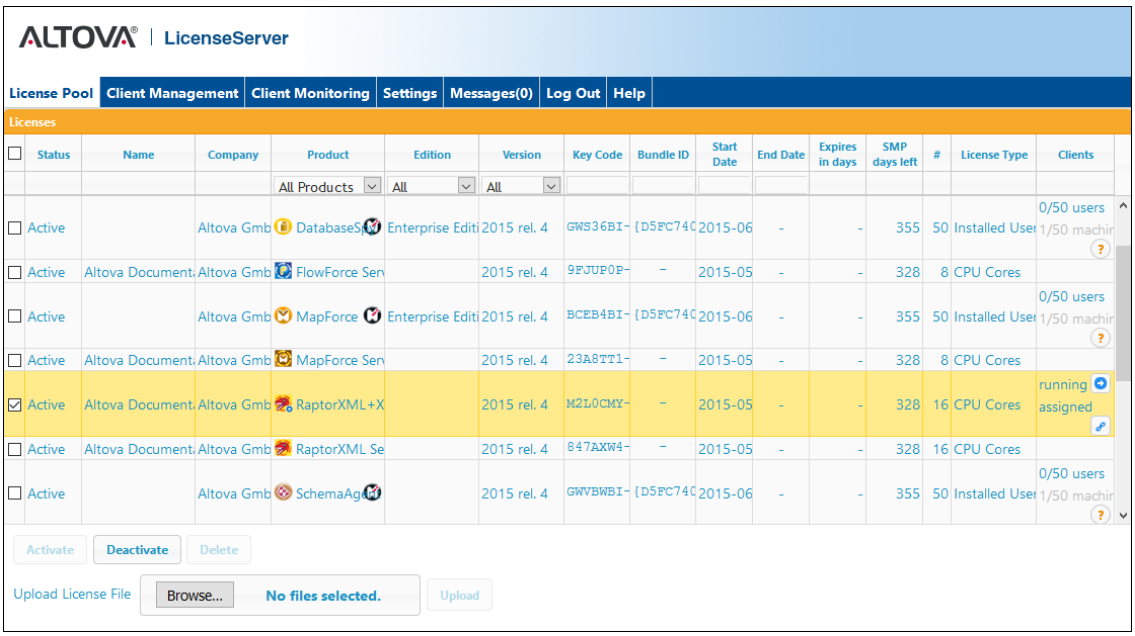
11.6.5 ライセンスの LicenseServer へのアップロード

このセクション:

ライセンスのを LicenseServer のライセンス プールへアップロード

Altova からライセンスを取得した後、ライセンスを Altova LicenseServer にアップロードする必要があります。各ライセンスファイルは購入により 1 つ以上のライセンスを含みます。ライセンスファイルをアップロードする際、ファイルのすべてのライセンスが LicenseServer のライセンスプールにアップロードされ、LicenseServer に登録された Altova 製品に割り当てられます。アップロードされた 1 つまたは 1 以上のライセンスファイルからの Altova 製品のライセンスは、すべて LicenseServer の 1 つのライセンスプールに収集されます。ライセンスプールは LicenseServer の構成ページのライセンスプールタブに表示されます (*下のスクリーンショット*)。

ライセンスはライセンスプールタブのアップロード機能を使用して LicenseServer にアップロードされます (*スクリーンショット参照*)。



ライセンスファイルは、ライセンスプール (License Pool) タブの ライセンスファイルのアップロード (Upload License File) 機能を使用して、LicenseServer にアップロードされます (上のスクリーンショット参照)。 **参照」 (Browse)** ボタンをクリックして希望するライセンスファイルを選択します。ライセンス ファイルのアップロード (Upload License File) テキスト フィールドにライセンス ファイルが表示され、 **「アップロード」 (Upload)** ボタンが有効化されます。 **「アップロード」 (Upload)** ボタンをクリックしてライセンス ファイルをアップロードします。ファイルの全てのライセンスは、アップロードされライセンスプールタブに表示されます。下のスクリーンショットは、複数のライセンスファイルからアップロードされた複数のライセンスを表示しています。

ライセンスの状態

ライセンスの状態の値は以下の通りです：

- *アクティブ化*：ライセンスが、LicenseServer のライセンスプールにアップロードされると、サーバーはライセンスに関連したデータを altova.com マスター ライセンス サーバーに、検証、認証、与えられたライセンスをアクティブ化するために送信します。これは、Altova ライセンス使用許諾契約書への順守を確認するために必要です。通常 30 秒から数分かかる、初回アクティブ化と認証トランザクション中、インターネットの接続スピードとネットワークの交通量にもよりますが、ライセンスの状態は*アクティブ化 (Activating...)* と表示されます。

- *失敗した検証*：altova.com マスターライセンスサーバーへの接続が確立しなかった場合、プール内のライセンスの状態は*失敗した検証 (Failed Verification)* と表示されます。これは起こり得ることですので、インターネットの接続とファイアウォールのルールを確認して、LicenseServer が altova.com マスターライセンスサーバーと通信できるようご確認してください。

- *アクティブ*：ライセンスが認証されてアクティブ化されると、状態は*アクティブ (Active)* に変更されます。

- *非アクティブ*：ライセンスは検証されたが、ネットワークの他の LicenseServer に存在する場合、状態は*非アクティブ (Inactive)* と表示されます。非アクティブ状態は、管理者がライセンスプール内でのライセンスを手動で非アクティブ化に設定した際におこります。

- *保留*：ライセンスの開始の日付が未来の日付である場合、ライセンスは*保留*として表示されます。この状態は、製品に割り当てることができ、現在のライセンスの有効期限が切れた場合でも、製品に対するライセンスが、継続されることを保証します。（製品に対して一度に2つのアクティブなライセンスを割り当てることが許可されています。）

- *ブロックされた*：ライセンスの認証に問題がある場合、ライセンスは*ブロックされた (Blocked)* と表示されます。また、altova.com マスターライセンスサービスが、このライセンスを使用する許可を与えていない場合も表示されます。使用許諾契約書の違反、ライセンスの過度の使用、または他の順守問題などにより引き起こされます。ライセンスが*ブロックされた (Blocked)* と表示されている場合、Altova サポートにライセンスおよび他の関連情報と共に連絡してください。

これらの状態は以下のテーブルにまとめられています：

状態	意味
アクティブ化 *Activating...*	アップロードする際、ライセンスの情報は altova.com に検証のために送信されます。アップデートされた状態を確認するためにブラウザーを更新してください。検証とアクティブ化は数分かかります。
失敗した検証 *Failed Verification*	altova.com への接続が確立しませんでした。接続を確立し、サーバーを再開始するか、**[Activate]** ボタンを使用してライセンスをアクティブ化します。
アクティブ *Active*	検証に成功し、ライセンスはアクティブ化されました。
非アクティブ *Inactive*	検証はお成功しましたが、ライセンスがネットワークの他の LicenseServer に存在します。ライセンスは **[Deactivate]** ボタンにより非アクティブ化することができます。
ブロックされた *Blocked*	検証が成功しませんでした。ライセンスは無効でブロックされています。Altova サポートに連絡してください。

メモ： ライセンスが altova.com に検証のため送信された後、アップデートされた状態を確認するためにブラウザーを更新する必要があります。検証とアクティブ化は数分かかります。

メモ: altova.com への接続が確立しない場合、状態は*失敗した検証 (Failed Verification)* と表示されます。接続を確立した後、への接続が確立しませんでした。接続を確立し、サーバーを再開始するか、**[Activate]** ボタンを使用して)ライセンスをアクティブ化します。

メモ: ライセンス状態が*非アクティブ*または*ブロック*されたと表示されている場合、*ステータス*を説明したメッセージがメッセージログに追加されます。

製品のインストールにはアクティブな、または、保留されているライセンスのみを割り当てることができます。非アクティブなライセンスはアクティブ化されるか、またはライセンスプールから削除することができます。ライセンスがライセンスプールから削除された場合、ライセンスファイルを再度アップロードすることでアップロードできます。ライセンスファイルがアップデートされると、プールに存在しないライセンスのみがアップロードされます。ライセンスをアクティブ化、非アクティブ化、または削除するには、それぞれ、**[Activate]**、**[Deactivate]** または **[Delete]** ボタンをクリックしてください。

使用を希望するライセンスのアクティブ化

Altova 製品へライセンスを割り当てる前に、ライセンスをアクティブ化する必要があります。ライセンスがアクティブ化されていることを確認してください。非アクティブの場合、選択して **「アクティブ化」 (Activate)** してください。

次のステップ

LicenseServer にライセンスファイルをアップロードし、希望するライセンスがアクティブ化されていることを確認した後、以下を行います:

1. Altova サーバー製品 (FlowForce Server、MapForce Server、StyleVision Server) を LicenseServer に登録する。(ライセンスファイルのアップロード前にこの手順を既に済ませている場合、ライセンスの割り当てを開始することができます。)
2. LicenseServer に登録されたAltova 製品にライセンスの割り当てを行います。

11.6.6 製品の登録

Altova サーバー製品にライセンスの割り当てる前に、LicenseServer に製品のインストールを登録しなければなりません。Altova サーバー製品から登録がされ、Web UI があるサーバー製品と、コマンドラインのみで作動する製品ではプロセスが異なります。登録を実行するには、LicenseServer がインストールされているマシンのサーバー名またはIP アドレスが必要です。

- *デスクトップ製品*: ソフトウェアのライセンス認証ダイアログを使用して、登録が行われます。
- *Web UI を持つサーバー製品*: FlowForce Server とMobileTogether Server の登録は、Web UI の *セットアップ* タブまたは製品のCLI により行うことができます。
- *Web UI を持たないサーバー製品*: MapForceServer、RaptorXML(+XBRL) Server、と StyleVisionServer の登録は、これらの製品のCLI を使用して行います。LicenseServer がインストールされているマシンのサーバー名またはIP アドレスが登録のために必要になります。

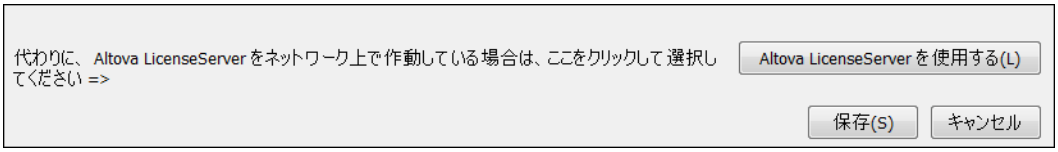
異なるAltova サーバー製品の登録方法を説明します:

- Altova デスクトップ製品の登録
- FlowForce Server の登録
- MapForce Server の登録
- MobileTogether Server の登録
- RaptorXML(+XBRL) Server の登録
- StyleVision Server の登録

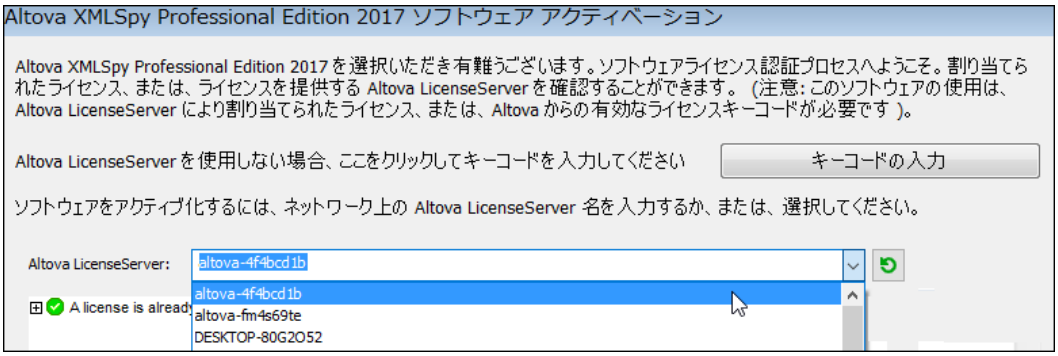
Altova デスクトップ製品の登録

Altova LicenseServer にAltova デスクトップ製品を登録するには、以下を行います:

1. メニューコマンド **「ヘルプ | ソフトウェアのライセンスの認証」** を選択して、製品のソフトウェアライセンス認証ダイアログに移動します。ライセンスの承認は、(i) Altova LicenseServer を使用して、または、(i) キーコードの詳細を入力しておこなうことができます。このドキュメントでは、Altova LicenseServer を使用した場合のライセンスの認証について説明します。
2. LicenseServer を使用して製品のライセンスの認証をおこなうには、(ダイアログの下にある) **「Altova LicenseServer を使用する」** をクリックします (下のスクリーンショットを参照してください)。



3. これによりダイアログがLicenseServer のライセンス認証モードに切り替えられます (*下のスクリーンショット参照*)。*Altova LicenseServer* コンボボックスのドロップダウンリストから、LicenseServer を選択します。



選択されたLicenseServer への接続が構築されると、製品はすぐに選択されたLicenseServer に登録されます。クライアント管理タブ, 内で使用中の製品リストに製品が表示されます。

デスクトップ製品の登録解除

デスクトップ製品の登録を解除するには、LicenseServer のクライアント管理タブに移動し、*製品のライセンス*ペイン内の右側にある製品の **「製品の登録解除」** ボタンをクリックします。

FlowForce Server の登録

このセクション:

- LicenseServer にFlowForce Server を登録する方法
- FlowForce Server セットアップページへのアクセス (Windows)
- FlowForce Server セットアップページへのアクセス (Linux)
- セットアップページを介してのFlowForce Server の登録
- FlowForce CLI を介してのFlowForce Server の登録 (Windows)
- FlowForce CLI を介してのFlowForce Server の登録 (Linux)
- 次のステップ

LicenseServer に FlowForce Server を登録する方法

FlowForce Server のLicenseServer への登録は以下の方法が使用できます:

- FlowForce Server セットアップページを介して
- FlowForce CLI を介して (Windows)
- FlowForce CLI を介して (Linux)

FlowForce Server セットアップページへのアクセス (Windows)

FlowForce Server セットアップページへは以下の方法でアクセスできます:

- **スタート** メニューから:
  スタート| Altova FlowForce Server 2017 | FlowForce Server セットアップページ
- Altova ServiceControllerから:システムトレイのServiceController アイコンをクリックします。ポップアップしたメニューから*Altova FlowForce Web | Setup* を選択します。

FlowForce Server セットアップページ (*上部スクリーンショット*) がポップアップします。

FlowForce Server セットアップページへのアクセス (Linux)

Linux に FlowForce Server をインストールした後、手順に関しては FlowForce Server ユーザードキュメンテーションを参照してください)。以下のコマンドを使用して FlowForce Web Server をサービ

スとして開始します：

```
sudo /etc/init.d/flowforcewebserver start
```

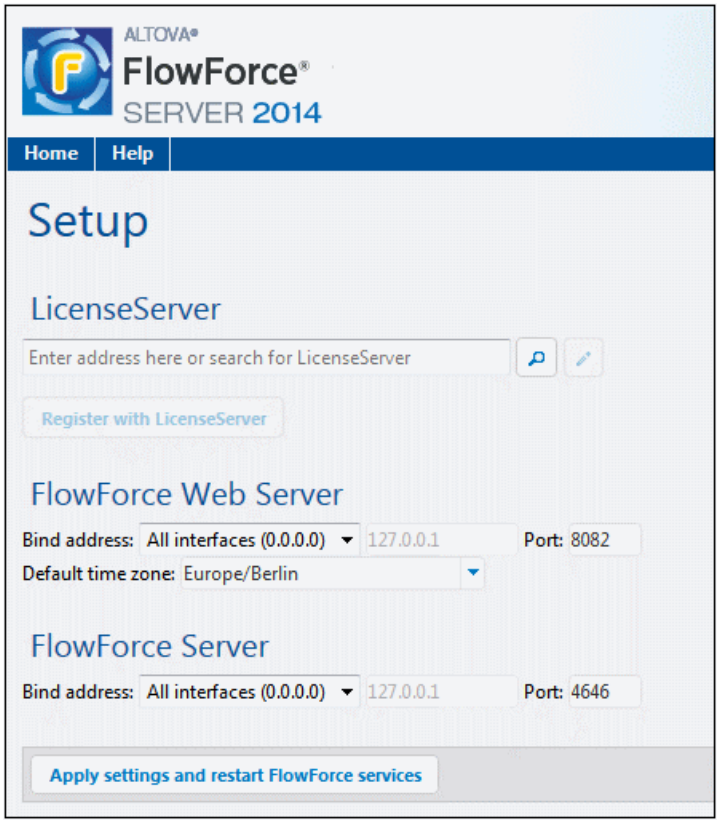
FlowForce Server のURL を含んだメッセージがターミナルウィンドウに表示されます：

```
FlowForceWeb running on http://127.0.1.1:3459/setup?key=52239315203
```

アドレスフィールドに URL を入力して、FlowForce Server セットアップページにアクセスするために「**Enter**」を押します。 (*下のスクリーンショット*)。

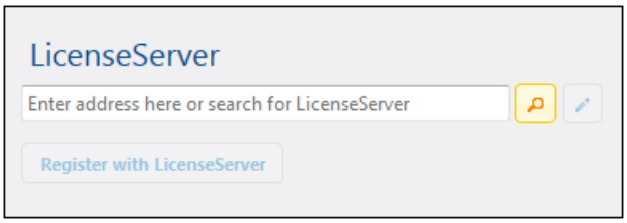
セットアップページを介しての FlowForce Server の登録

セットアップページ (*下のスクリーンショット*) ？へのアクセス方法は上記されています？ LicenseServer フィールドは Altova LicenseServer を登録するために指定されています。

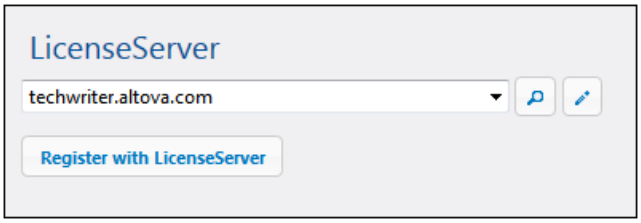


LicenseServer 以下の 2 つの方法で指定できます。

- 現在ネットワークで使用可能な、つまり現在作動している、Altova LicenseServers を検索することができます。この手順は、「**Altova LicenseServers 検索**」(Search for Altova LicenseServers) ボタンをクリックすることで実行できます (*下のスクリーンショットで黄色にハイライトされています*)。



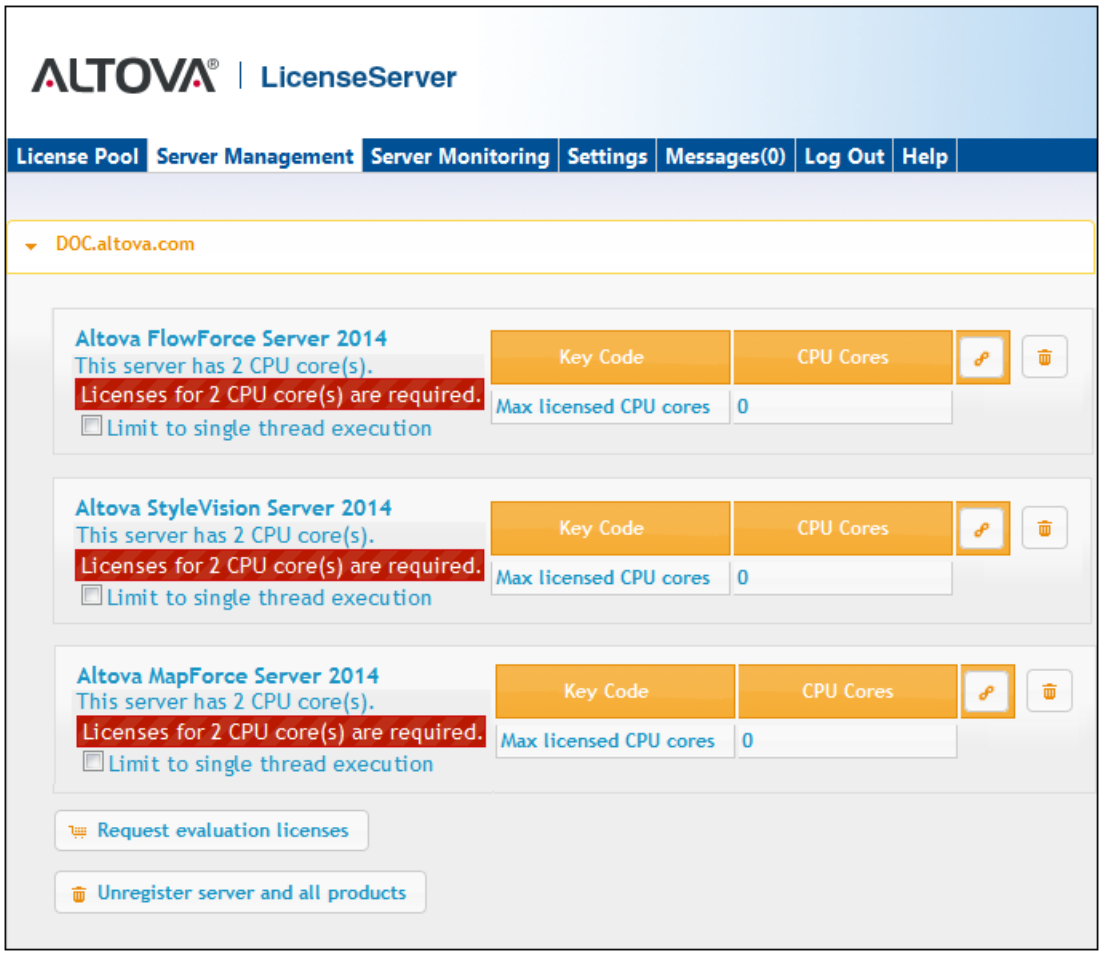
検索によりネットワーク上で使用可能なAltova LicenseServers のリストが返されます。1つの LicenseServer が選択され、(*下のスクリーンショット*) 他はコンボボックスのドロップダウンリストで使用可能です。FlowForce ライセンスが保管されているLicenseServer を選択します。



- または、LicenseServer のアドレスをLicenseServer のフィールドに入力します。現在作動するがドロップダウンリストで使用可能な場合、**手動でアドレスを入力」(Manually Enter Address)** ボタンをクリックして、LicenseServer フィールドにアドレスを入力することができます。

LicenseServer を指定した後、**「LicenseServer により登録」(Register with LicenseServer)** をクリックします。指定されたLicenseServer により、サーバーアプリケーションが登録されLicenseServer の構成ページのクライアント管理タブがブラウザーで開かれます (*下のスクリーンショット*)。

メモ: LicenseServer 構成ページを表示するためにポップアップを許可しなければならないかもしれません。



上部のスクリーンショットでは、3つの製品が`DOC.altova.com`のAltova LicenseServerに登録されています。ライセンスの割り当て方法に関しては、次のセクション登録された製品へのライセンスの割り当てで説明されています。

FlowForce CLI を介しての FlowForce Server の登録 (Windows)

Windows マシーンでは、FlowForce Server は`licenseserver`コマンドを使用し、コマンドライン(CLI)を介してネットワーク上のAltova LicenseServerに登録することができます:

```
FlowForceServer licenseserver Server-Or-IP-Address
```

例えば、LicenseServerが`http://localhost:8088`で作動している場合、FlowForce Serverを以下で登録します:

```
FlowForceServer licenseserver localhost
```

FlowForce Serverが他のサーバー製品のサブパッケージとしてインストールされている場合、FlowForce Serverの登録は自動的にAltovaサーバー製品も登録します。FlowForce Serverの登録に成功すると、LicenseServerに移動して、FlowForce Serverにライセンスを割り当てます。手順は登録された製品へのライセンスの割り当てのセクションに説明されています。

FlowForce CLI を介しての FlowForce Server の登録 (Linux)

Linux マシーンでは、FlowForce Server は FlowForce Server CLI の `licenseserver` コマンドを使用して LicenseServer に登録することができます。FlowForce Server はルート権限とともに開始されなければならないことに注意してください。

```
sudo /opt/Altova/FlowForceServer2017/bin/flowforceserver licenseserver
localhost
```

上記コマンドでは、`localhost` は、LicenseServer がインストールされているサーバーの名前です。FlowForce Server 実行可能ファイルの場所は以下の通りです：

```
/opt/Altova/MapForceServer2017/bin
```

FlowForce Server の登録が成功すると、LicenseServer に移動して、FlowForce Server にライセンスを割り当てます。手順は登録された製品へのライセンスの割り当てのセクションに説明されています。

次のステップ

Altova 製品を LicenseServer に登録した後、以下を行ってください：

1. LicenseServer にまだライセンスファイルをアップロードしていない場合、前述のセクション ライセンスのアップロードを参照してください)、ライセンスファイルをアップロードし、アクティブ化したいライセンスをチェックします。既に、この手順が済んでいる場合、次のステップライセンスの割り当てに進んでください。
2. LicenseServer に既に登録されている Altova 製品にライセンスの割り当てを行ってください。

MapForce Server の登録

このセクション:

MapForce Server は FlowForce Server の一部として、またスタンドアロンのサーバー製品としてインストールすることができます。どちらの場合でも、Altova LicenseServer に登録されなければなりません。LicenseServer に登録された後のみ、LicenseServer からライセンスが割り当てられます。Windows システムでは、MapForce Server が FlowForce Server の一部としてインストールされる場合、FlowForce が登録される際自動的に登録されます。Linux システムでは、MapForce Server が FlowForce Server の後にインストールされる場合、FlowForce Server が登録される際に自動的に登録されます。MapForce Server が FlowForce Server の前にインストールされると、両方の製品を個別に登録する必要があります。

FlowForce Server からの MapForce Server の登録 (Windows)

MapForce Server は FlowForce Server にパッケージされており、FlowForce Server がネットワークの Altova

LicenseServer に登録されている場合、MapForce Server は自動的にLicenseServer に登録されます。FlowForce Server の登録方法は、のドキュメンテーションのLicenseServer にFlowForce Server を登録するセクションに説明されています。

登録の後、LicenseServer に移動してMapForce Server ライセンスをMapForce Server に割り当てます。手順は登録された製品にライセンスを割り当てるセクションに説明されています。

スタンドアロンの MapForce Server の登録 (Windows)

MapForce Server をスタンドアロンパッケージとしてインストールした場合、ネットワークのAltova LicenseServer に登録し、Altova LicenseServer からライセンスを与える必要があります。MapForce Server をコマンドラインインターフェイス(CLI) 介して `licenseserver` コマンドを使用して登録することができます :

```
MapForceServer licenseserver Server-Or-IP-Address
```

例えば、LicenseServer が以下で作動している場合、`http://localhost:8088`、MapForce Server を以下で登録します :

```
MapForceServer licenseserver localhost
```

MapForce Server の登録に成功すると、LicenseServer に移動して、MapForce Server にライセンスを割り当てます。手順はセクション登録された製品にライセンスを割り当てるに説明されています。

MapForce Server の登録 (Linux)

Linux マシンでは、 MapForce Server をLicenseServer にMapForce Server CLI の `licenseserver` コマンドを使用して登録することができます。MapForce Server はルート権限とともに開始されなければならないことに注意してください。

```
sudo /opt/Altova/MapForceServer2017/bin/mapforceserver licenseserver
localhost
```

上記コマンドでは、`localhost` は、LicenseServer がインストールされているサーバーの名前です。MapForce Server 実行可能ファイルの場所は以下の通りです :

```
/opt/Altova/MapForceServer2017/bin
```

MapForce Server の登録が成功すると、LicenseServer に移動して、MapForce Server にライセンスを割り当てます。手順は登録された製品へのライセンスの割り当てのセクションに説明されています。

次のステップ

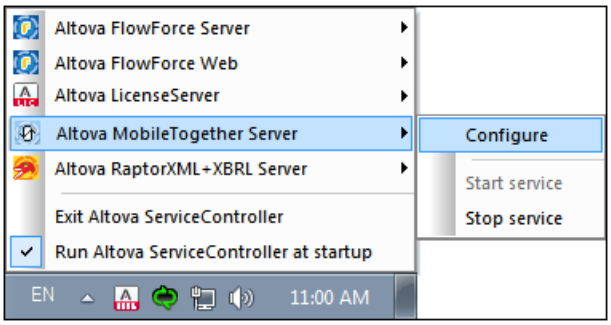
Altova 製品をLicenseServer に登録した後、以下を行ってください :

1. LicenseServer にまだライセンスファイルをアップロードしていない場合、(前述のセクション ライセンスのアップロードを参照してください)、ライセンスファイルをアップロードし、アクティブ化したいライセンスをチェックします。既に、この手順

が済んでいる場合、次のステップライセンスの割り当てに進んでください。
2. LicenseServer に既に登録されているAltova 製品にライセンスの割り当て を行ってください。

MobileTogether Server の登録

MobileTogether 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。ポップアップしたメニュー **Altova MobileTogether Server** をポイントし、 (*下のスクリーンショット参照*)、MobileTogether Server サブメニューから **「サービスの開始」(Start Service)** を選択します。MobileTogether Server が既に作動している場合、 *サービスの開始 (Start Service)* オプションは無効化されます。



MobileTogether Server の登録 :

- MobileTogether Server Web UI の設定タブ : (i) ServiceController を介して、MobileTogether を開始する (*前述のポイント参照*)。 (i) 構成ページにアクセスするためにパスワードを入力する。 (ii) 設定タブを選択する。(v)ページ下のLicenseServer ペインに移動する。LicenseServer 名またはアドレスを入力し、 **「LicenseServer により登録」 (Register with LicenseServer)** をクリックする。

- CLI の `licenseserver` コマンドを使用する :
  `MobileTogetherServer licenseserver [options]` ***`ServerName-Or-IP-Address`***
  例えば、LicenseServer がインストールされているサーバー名 `localhost` の場合 :
  `MobileTogetherServer licenseserver localhost`

登録に成功した後、LicenseServer の構成ページのサーバー管理ページに移動して、MobileTogether Server にライセンスを割り当てます。

RaptorXML(+XBRL) Server の登録

このセクション:

- RaptorXML(+XBRL) Server の登録 (Windows)
- RaptorXML(+XBRL) Server の登録 (Linux)
- 次のステップ

RaptorXML(+XBRL) Server はLicenseServer が接続されているサーバーマシンにインストールされ、サービスとして開始される必要があります。また、LicenseServer に登録されていなければなりません。登録後のみ、LicenseServer からライセンスの割り当て を行うことができます。このセクションでは、RaptorXML(+XBRL) Server のLicenseServer での登録方法を説明します。

RaptorXML(+XBRL) Server の登録 (Windows)

RaptorXML(+XBRL) Server をコマンドラインインターフェイス CLI を介し `licenseserver` コマンドを使用して登録することができます :

RaptorXML Server:	**`RaptorXML licenseserver Server-Or-IP-Address`**
RaptorXML+XBRL Server:	**`RaptorXMLXBRL licenseserver Server-Or-IP-Address`**

例えば、LicenseServer が以下で作動している場合 `http://localhost:8088`、RaptorXML(+XBRL) Server を以下で登録します :

RaptorXML Server:	**`RaptorXML licenseserver localhost`**
RaptorXML+XBRL Server:	**`RaptorXMLXBRL licenseserver localhost`**

RaptorXML(+XBRL) Server の登録に成功すると、LicenseServer に移動して、RaptorXML(+XBRL) Server にライセンスを割り当てます。手順はセクション登録された製品にライセンスを割り当てるに説明されています。

RaptorXML(+XBRL) Server の登録 (Linux)

Linux マシーンでは、RaptorXML(+XBRL) Server を LicenseServer に RaptorXML(+XBRL) Server CLI の `licenseserver` コマンドを使用して登録することができます。RaptorXML(+XBRL) Server はルート権限とともに開始されなければならないことに注意してください。

```
sudo /opt/Altova/RaptorXMLServer2017/bin/raptorxmlserver licenseserver
localhost
sudo /opt/Altova/RaptorXMLXBRLServer2017/bin/raptorxmlxbrlserver
licenseserver localhost
```

上記コマンドでは、`localhost` は、LicenseServer がインストールされているサーバーの名前です。RaptorXML(+XBRL) Server 実行可能ファイルの場所は以下の通りです :

```
/opt/Altova/RaptorXMLServer2017/bin
/opt/Altova/RaptorXMLXBRLServer2017/bin
```

RaptorXML(+XBRL) Server の登録に成功すると、LicenseServer に移動して、RaptorXML(+XBRL) Server にライセンスを割り当てます。手順はセクション登録された製品にライセンスを割り当てるに説明されています。

次のステップ

Altova 製品を LicenseServer に登録した後、以下を行ってください :

1. LicenseServer にまだライセンスファイルをアップロードしていない場合、前述のセクション ライセンスのアップロードを参照してください)、ライセンスファイルをアップロードし、アクティブ化したいライセンスをチェックします。既に、この手順

が済んでいる場合、次のステップライセンスの割り当てに進んでください。
2. LicenseServer に既に登録されているAltova 製品にライセンスの割り当て を行ってください。

StyleVision Server の登録

このセクション:

- FlowForce Server からのStyleVision Server の登録 (Windows)
- スタンドアロンのStyleVision Server の登録 (Windows)
- StyleVision Server の登録 (Linux)
- 次のステップ

StyleVision Server はFlowForce Server の一部として、またスタンドアロンのサーバー製品としてインストールすることができます。どちらの場合でも、Altova LicenseServer に登録されなければなりません。LicenseServer に登録された後のみ、LicenseServer からライセンスが割り当てられます。Windows システムでは、StyleVision Server が FlowForce Server の一部としてインストールされる場合、FlowForce が登録される際自動的に登録されます。Linux システムでは、StyleVision Server がFlowForce Server の後にインストールされる場合のみ、FlowForce Server が登録される際に自動的に登録されます。

FlowForce Server からの StyleVision Server の登録 (Windows)

StyleVision Server はFlowForce Server にパッケージされており、FlowForce Server がネットワークのAltova LicenseServer に登録されている場合、StyleVision Server は自動的にLicenseServer に登録されます。FlowForce Server の登録方法は、のドキュメンテーションのLicenseServer にFlowForce Server を登録するセクションに説明されています。

登録の後、LicenseServer に移動してStyleVision Server ライセンスをStyleVision Server に割り当てます。手順は登録された製品にライセンスを割り当てるセクションに説明されています。

スタンドアロンの StyleVision Server の登録 (Windows)

StyleVision Server をスタンドアロン パッケージとしてインストールした場合、ネットワークのAltova LicenseServer に登録し、Altova LicenseServer からライセンスを与える必要があります。 StyleVision Server を コマンドラインインターフェイス(CLI) 介して `licenseserver` コマンドを使用して登録することができます :

```
StyleVisionServer licenseserver Server-Or-IP-Address
```

例えば、LicenseServer が以下で作動している場合、`http://localhost:8088`、StyleVision Server を以下で登録します :

```
StyleVisionServer licenseserver localhost
```

StyleVision Server の登録に成功すると、LicenseServer に移動して、StyleVision Server にライセンスを割り当てます。手順はセクション登録された製品にライセンスを割り当てるに説明されています。

StyleVision Server の登録 (Linux)

Linux マシンでは、StyleVision Server をLicenseServer にStyleVision Server CLI の`licenseserver` コマンドを使用して登録することができます。StyleVision Server はルート権限とともに開始されなければならないことに注意してください。

```
sudo /opt/Altova/StyleVisionServer2017/bin/stylevisionserver licenseserver
localhost
```

上記コマンドでは、`localhost` は、LicenseServer がインストールされているサーバーの名前です。StyleVision Server 実行可能ファイルの場所は以下の通りです：

```
/opt/Altova/StyleVisionServer2017/bin
```

StyleVision Server の登録が成功すると、LicenseServer に移動して、StyleVision Server にライセンスを割り当てます。手順は登録された製品へのライセンスの割り当てのセクションに説明されています。

次のステップ

Altova 製品をLicenseServer に登録した後、以下を行ってください：

1. LicenseServer にまだライセンスファイルをアップロードしていない場合、(前述のセクション ライセンスのアップロードを参照してください)、ライセンスファイルをアップロードし、アクティブ化したいライセンスをチェックします。既に、この手順が済んでいる場合、次のステップライセンスの割り当てに進んでください。
2. LicenseServer に既に登録されているAltova 製品にライセンスの割り当て を行ってください。

11.6.7 登録された製品へのライセンスの割り当て

このセクション:

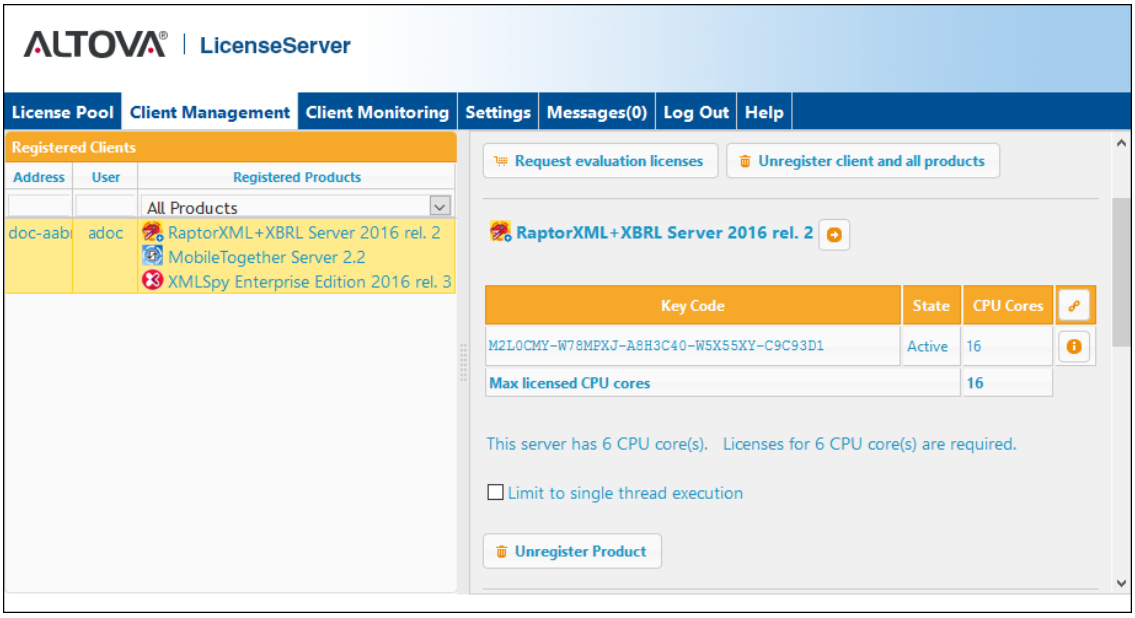
ライセンスの割り当ての前に

Altova 製品にライセンスを割り当てる前に以下を確認してください:

- LicenseServer のライセンスプール に対応したライセンスがアップロードされ、ライセンスがアクティブであること。
- Altova 製品がLicenseServer に登録されていること。

クライアント管理タブ

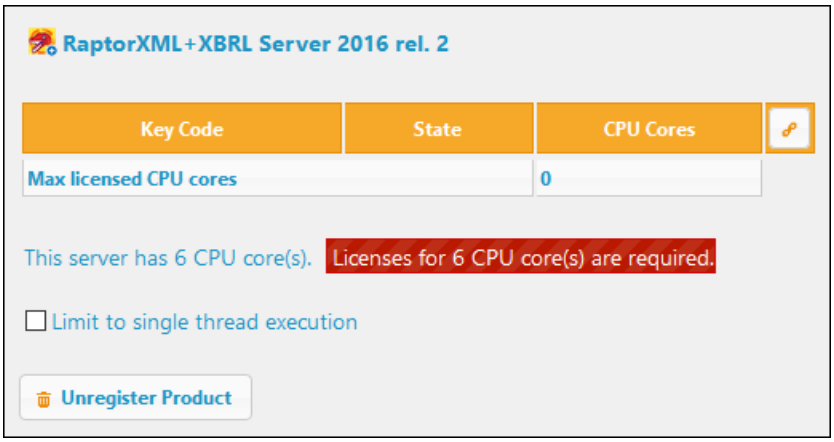
構成ページの クライアント管理タブ 内でライセンスは割り当てられます (*下のスクリーンショット*)。スクリーンショットは、左側のペイン内に、3つのAltova 製品がLicenseServer に登録されているマシンが1台あることを表示しています。



クライアント管理タブについての以下の点に留意してください:

- 左側のペインで、各製品は、クライアントマシンの名前の下にリストされています。上のスクリーンショットでは、1台のクライアントマシンがリストされています。このクライアントマシンには、3つのAltova 製品がLicenseServer に登録されています。Altova 製品が異なるクライアントマシンでこのLicenseServer に登録されている場合も左側のペインに表示されます。
- 左側のペインで、クライアントマシンを選択すると、マシンに登録されている製品の詳細は、右側のペインに表示されています。個々では、各製品のライセンスの割り当てを編集することができます。

- クライアントマシン上に登録されている各 Altova 製品には、自身のキーコードエントリがあり、ライセンスのキーコードを読み取ります。登録されている製品は、**「割り当てられたライセンスを編集する」**ボタンをクリックし、ライセンスプール内からその製品に使用することのできる必要なライセンスを選択することによりライセンスに割り当てられます（*下のアイコンのリストを参照してください*）。この手順に関しては、下で更に詳しく説明されています。
- サーバー製品にも、クライアント上でライセンスが作動するために必要なコア数を表示するラインがあります。ライセンスされているコアが、必要なコア数よりも少ない場合、この情報は赤でマークされます（*下のスクリーンショット参照*）。（ライセンスされる必要のあるコア数は、は、クライアント上のCPU コアの数で、LicenseServer によりクライアントマシンから取得されます）。



- 単一製品の**複数のバージョン**（例えば、StyleVision Server 2013 とStyleVision Server 2014）が 1 つのコンピュータにインストールされ、インストールが 1 つのLicenseServer と登録された場合、複数の登録は、クライアント管理タブ内で 1 つの登録として統合され、1 つの登録として表示されます。ライセンスがこの 1 つの登録に割り当てられる際、この登録により指定される全てのインストールにライセンスが与えられます。しかし、単一インストールの複数のインスタンスは、クライアントマシンで同時に作動することができます。例えば、StyleVision Server 2013 の複数のインスタンスまたは、StyleVision Server 2014 の複数のインスタンスは同時に作動することができますが、StyleVision Server 2013 の 1 つのインスタンスとStyleVision Server 2014の 1 つのインスタンスはできません。新しくインストールされたバージョンは作動するために登録されている必要があります。
- Altova サーバー製品の新しいバージョンは、製品のリリース時のLicenseServer の最新バージョンによりのみライセンスを受けることができます。古いAltova サーバー製品はLicenseServer の新しいバージョンと作動することができます。ですから、新しいバージョンのAltova サーバー製品をインストールする際、現在使用しているLicenseServer のバージョンが最新でない場合、古いバージョンのLicenseServer をアンインストールして、最新バージョンをインストールしてください。アンインストールの際、古いバージョンのLicenseServer の登録とライセンス情報はクライアントマシンのデータベースに保存され、新しいバージョンに自動的にインポートされます。（サーバー製品の特定のバージョンに適切なLicenseServer バージョン番号がサーバー製品のインストール中表示されます。サーバー製品と共にこのバージョンを選択することができます。現在インストールされているバージョンは LicenseServer 構成ページの下部に表示されます。）

クライアント管理タブのアイコン

割り当てられたライセンスの編集。各製品のリストで使用することができます。新しいライセンスを製品に割り当てることのできる、すでに割り当てられたライセンスを編集できる 割り当てられたライセンスの編集 がポップアップします。

ライセンスの表示。各ライセンスに表示されます License Pool タブに切り替えができ、選択されたライセンスをハイライトされることによりライセンスの詳細がわかります。

製品の登録解除。各製品で利用可能です。（選択されたクライアントマシン上の）選択された製品をLicenseServer から削除することができます。

コアとライセンスについてのメモ

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。

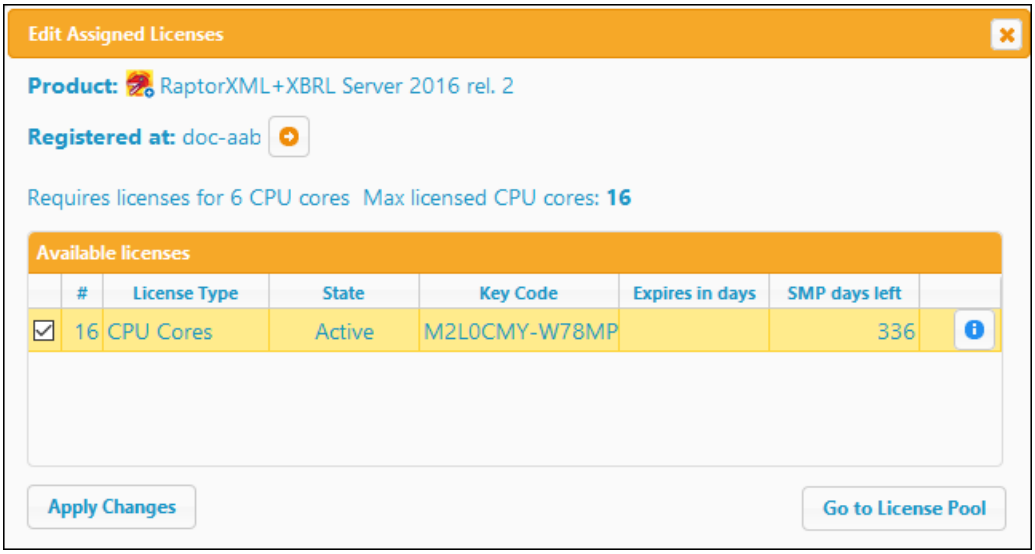
MobileTogether Server ライセンス

MobileTogether Server ライセンスには2つの種類があります。カスタマーは必要に応じてライセンスの種類を選択することができます。

- *コア ライセンス*: サーバー マシンのコア数をベースにして MobileTogether Servers に割り当てられます。上の例を参照してください。上の説明を参照してください。コア ライセンスは、無制限の数量の MobileTogether クライアント デバイスによりサーバーへの接続を許可します。しかしながら、「単一スレッドの実行」チェックボックスがチェックされていると、1 度に MobileTogether Server に接続できるモバイル デバイスは 1 台です。これは、評価と小さい規模のテストを行う際に役に立ちます。この場合、2 台目のモバイル デバイスが MobileTogether Sever に接続される場合、ライセンスは 2 台目が使用するようになります。最初のデバイスは、接続できないようになり、エラー メッセージが表示されます。
- *デバイス ライセンス*: MobileTogether Server にいつでも接続することのできる MobileTogether Client デバイスの最高使用数を指定します。

ライセンスの割り当て

登録されている製品にライセンスを割り当てるには、製品の **[割り当てられたライセンスの編集 (割り当てられたライセンスを編集する)]** ボタンをクリックします。ライセンスの管理 (Manage Licenses) 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*)。

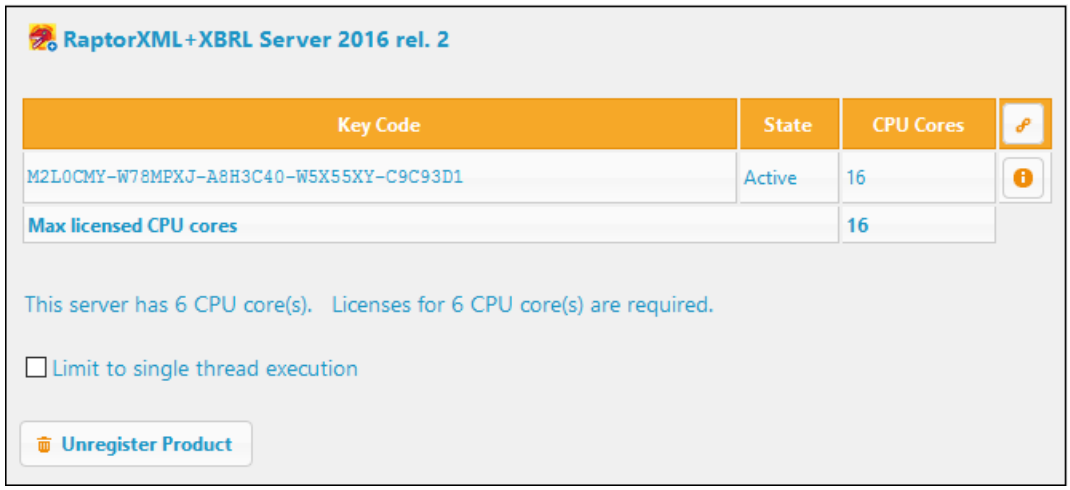


ライセンス管理ダイアログに表示されるライセンスについての以下の点に注意してください:

- ライセンスされる製品はダイアログの上部左にリストされます。上部のスクリーンショットでは、製品はAltova RaptorXML+XBRL Server です。
- サーバーがインストールされているマシン (上のスクリーンショットでは `doc-aab`) が横にリストされます。
- ダイアログは、ライセンスプールにあるその製品の現在アクティブなライセンスを表示します。スクリーンショットでは、現在アクティブなライセンスである、RaptorXML+XBRL Server ライセンスがライセンスプールにあります。LicenseServer は自動的にプール内の製品のために発行された各ライセンスを検知します。
- ライセンスの種類は、コア数ごと (*cores*) (MobileTogether Server を含むすべての Altova サーバー製品) または ユーザーごと (*Users*) (MobileTogether Server のみ) であることができます。ライセンスの種類はライセンスの種類 (*License Type*) カラムに表示されています。
- 上のスクリーンショット内のライセンスは 16 CPU コア分ライセンスされています。
- Altova サーバー製品がインストールされているサーバーのプロセッサーコア数を把握する必要があります。マシンがデュアル コア プロセッサの場合、2-コア (CPU コア数) ライセンスが必要です。サーバー製品の登録に必要なコア数はマシンの名前の下にリストされています。サーバーに割り当てるライセンスはコア数に対して十分有効である必要があります。必要なコア数を達成するためにライセンスを組み合わせることができます。マシンのプロセッサがオクタコア (8 コア) の場合、2 つの 4 コアライセンスを組み合わせることができます。
- 割り当てされたライセンスの編集ダイアログは、製品の現在アクティブなライセンスのみをリストします。他の Altova 製品のライセンスはリストされません。
- 既に割り当てられたライセンスに関しては、たとえば、ネットワークでの製品の他のインストールは、チェックボックスがチェックされています。ですからチェックされていないライセンスのみが選択できます。
- *CPU コア* (または、MobileTogether Server のためのユーザー数) カラムは、ライセンスに有効な CPU コア数 (または MobileTogether クライアント数) を表示しています。
- ライセンスプールの変更を希望する場合、例えば、ライセンスのアップロード、アクティブ化、非アクティブ化、および削除は **[ライセンスプールへの移動 (Go to License Pool)]** ボタンをクリックしてください。

割り当てを希望するライセンスの選択。ライセンスチェックボックスがチェックされます。また、製品のライセンスされた CPU コア数がダイアログ上部左に*最大限ライセンスされた CPU コア (Max licensed CPU コア)* とリストされます *(上部スクリーンショット参照)*。クライアントの製品のライセンスされた CPU コア数を増やしたい場合は更にライセンスを選択することができます。*最大限ライセンスされた CPU コア* はこの場合、選択されたすべてのライセンスのコア総数です。

ライセンスを選択した後、 **変更を適用」(Apply Changes)** をクリックします。製品に割り当てられたライセンスはクライアント管理タブに表示されます *(下のスクリーンショット参照)*。以下のスクリーンショットは Altova RaptorXML+XBRL に 16 CPU-コア ライセンスが割り当てられたことを表示しています。



LicenseServer からの製品の登録解除

LicenseServer により登録された各 Altova 製品は、*クライアントマシン名*の下の右側のペイン(*製品 ライセンス*)に表示されており、エントリの下に**製品の登録解除**ボタンがあります (*上のスクリーンショット参照*)。LicenseServer から製品の登録を解除するために、このボタンをクリックします。ライセンスが製品に割り当てられている場合、割り当ては登録が解除されると停止されます。全ての製品の登録を解除するには、(*製品 ライセンス*)ペインの右上にある**クライアントと全ての製品の登録を解除する**ボタンをクリックします (*このセクションの最初のスクリーンショットを参照してください*)。

LicenseServer から製品の登録を解除するには以下を行います :

- *サーバー製品* : サーバー Web UI 内の設定ページに移動します。Web UI がサーバーに存在しない場合、コマンドプロンプトウィンドウを開き、製品の CLI を使用して、製品を登録します。各サーバー製品のための手順は以下で説明されています: Register FlowForce Server、Register MapForce Server、Register MobileTogether Server、Register StyleVision Server、と Register RaptorXML(+XBRL) Server。
- *デスクトップ製品*: 製品の *ソフトウェアのライセンス認証ダイアログ* 「**Help | ソフトウェアのライセンス認証**」(**Help | Software Activation**) により、LicenseServer モードからライセンス承認を切り替えることができます。*Altova LicenseServer* フィールドから登録する LicenseServer を選択します。製品は登録され、LicenseServer のクライアント管理タブの登録された製品のリストに表示されます。

LicenseServer に登録された各 Altova 製品はクライアント管理タブにクライアントマシン名の下にリストされ、**「登録解除」(Unregister)** アイコンがペインの下にあります (*上のスクリーンショット参照*)。このアイコンをクリックして製品の登録を解除します。ライセンスが製品に割り当てられている場合、製品の登録が解除されると、割り当ては終了します。全ての製品の登録を解除するには、クライアント管理タブの下にある **「クライアントとすべての製品の登録解除」(Unregister Client and All Products)** ボタンをクリックします (*このセクションの最初のスクリーンショットを参照してください*)。

製品を再登録する場合、**「割り当てられたライセンスを編集する」(Edit Assigned Licenses)** ボタンをクリックします。

11.7 構成ページ レファレンス

LicenseServer 構成ページはLicenseServer (Web UI) の管理者インターフェイスです。このページにより、LicenseServer の管理とLicenseServer により登録されたAltova 製品(FlowForce Server、MapForce Server、StyleVision Server 、RaptorXML(+XBRL) Server) へのライセンス供与を行うことができます。LicenseServer 構成ページはWeb ブラウザーで閲覧できます。構成ページの開き方は以下のセクションで説明されています :LicenseServer 構成ページの開き方 (Windows) とLicenseServer 構成ページの開き方 (Linux)。

このセクションは構成ページのユーザーレファレンスが構成ページのタブにより整理されています :

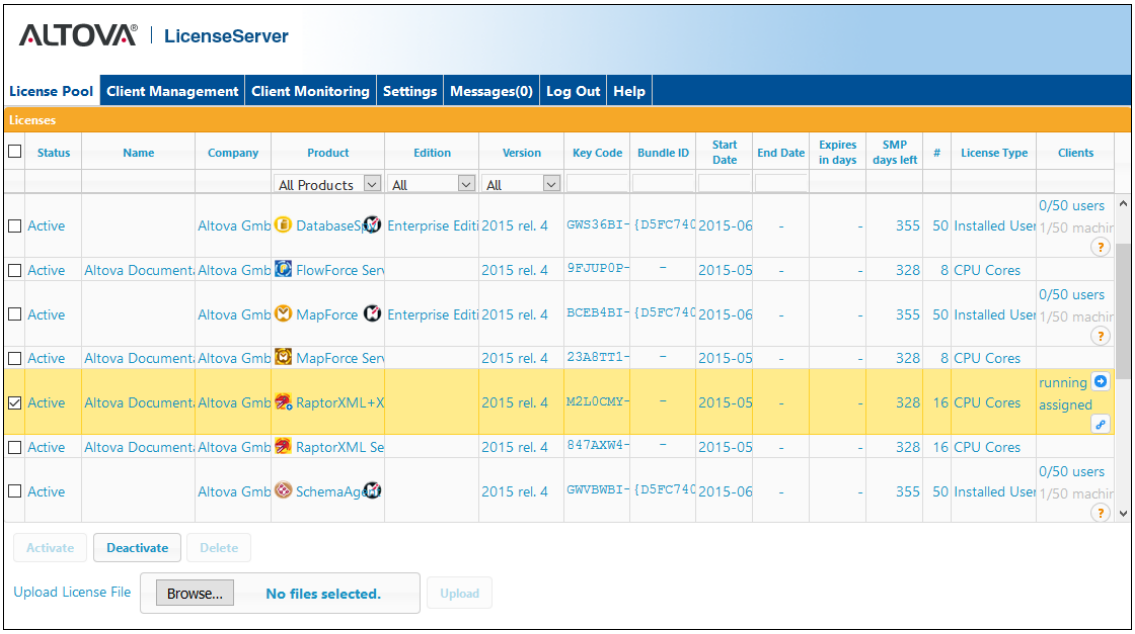
- License Pool
- クライアント管理
- クライアントの監視
- 設定
- メッセージ、ログアウト

LicenseServer でのライセンスの割り当てについてのステップ バイ ステップの手順は、ライセンスの割り当て方法セクションを参照してください。

11.7.1 ライセンスプール

このセクション:

ライセンスプール (**License Pool**) タブは、LicenseServer で現在使用することのできるライセンスに関する情報を表示します (*下のスクリーンショット参照*)。ライセンスファイル、このページの **「アップロード」(Upload)** ボタンを使用して、LicenseServer にアップロードされると、ライセンスファイル内に含まれている全てのライセンスが LicenseServer 上のライセンスプールに置かれます。ライセンスプールページは、ですから、上で現在使用することのできる全ての Altova 製品ライセンスの概要を、これらのライセンスの詳細と共に提供します。このページでは、更にライセンスプールにライセンスをアップロードできるだけではなく、選択されたライセンスを認証、認証の解除、または削除することができます。



ライセンスのアップロード

(株式会社 Altova から Altova サーバー製品に与えられた `altova_licenses` ファイル) をアップロードするには、**「参照」(Browse)** ボタンをクリックします。ライセンスファイルを参照し、選択します。**「アップロード」(Upload)** をクリックして、ライセンスファイルに含まれている全てのライセンスをライセンスプールにプールすると、ライセンスプールページにライセンスが表示されます (*下のスクリーンショット*)。

ライセンスの状態

ライセンスの状態の値は以下の通りです：

- *アクティブ化*：ライセンスが、LicenseServer のライセンスプールにアップロードされると、サーバーはライセンスに関連したデータを altova.com マスター ライセンス サーバーに、検証、認証、与えられたライセンスをアクティブ化するために送信します。これは、Altova ライセンス使用許諾契約書への順守を確認するために必要です。通常 30 秒から数分かかる、初回アクティブ化と認証トランザクション中、インターネットの接続スピードとネットワークの交通量にもよりますが、ライセンスの状態は*アクティブ化 (Activating...)* と表示されます。

- *失敗した検証*：altova.com マスターライセンスサーバーへの接続が確立しなかった場合、プール内のライセンスの状態は*失敗した検証 (Failed Verification)* と表示されます。これは起こり得ることですので、インターネットの接続とファイアウォールのルールを確認して、LicenseServer が altova.com マスターライセンスサーバーと通信できるようご確認してください。

- *アクティブ*：ライセンスが認証されてアクティブ化されると、状態は*アクティブ (Active)* に変更されます。

- *非アクティブ*：ライセンスは検証されたが、ネットワークの他の LicenseServer に存在する場合、状態は*非アクティブ (Inactive)* と表示されます。非アクティブ状態は、管理者がライセンスプール内でのライセンスを手動で非アクティブ化に設定した際におこります。

- *保留*：ライセンスの開始の日付が未来の日付である場合、ライセンスは*保留*として表示されます。この状態は、製品に割り当てることができ、現在のライセンスの有効期限が切れた場合でも、製品に対するライセンスが、継続されることを保証します。（製品に対して一度に2つのアクティブなライセンスを割り当てることが許可されています。）

- *ブロックされた*：ライセンスの認証に問題がある場合、ライセンスは*ブロックされた (Blocked)* と表示されます。また、altova.com マスターライセンスサービスが、このライセンスを使用する許可を与えていない場合も表示されます。使用許諾契約書の違反、ライセンスの過度の使用、または他の順守問題などにより引き起こされます。ライセンスが*ブロックされた (Blocked)* と表示されている場合、Altova サポートにライセンスおよび他の関連情報と共に連絡してください。

これらの状態は以下のテーブルにまとめられています：

状態	意味
アクティブ化 *Activating...*	アップロードする際、ライセンスの情報は altova.com に検証のために送信されます。アップデータされた状態を確認するためにブラウザーを更新してください。検証とアクティブ化は数分かかります。
失敗した検証 *Failed Verification*	altova.com への接続が確立しませんでした。接続を確立し、サーバーを再開始するか、**[Activate]** ボタンを使用してライセンスをアクティブ化します。
アクティブ *Active*	検証に成功し、ライセンスはアクティブ化されました。
非アクティブ *Inactive*	検証にお成功しましたが、ライセンスがネットワークの他の LicenseServer に存在します。ライセンスは **[Deactivate]** ボタンにより非アクティブ化することができます。
ブロックされた *Blocked*	検証が成功しませんでした。ライセンスは無効でブロックされています。Altova サポートに連絡してください。

メモ： ライセンスが altova.com に検証のため送信された後、アップデートされた状態を確認するためにブラウザーを更新する必要があります。検証とアクティブ化は数分かかります。

メモ: altova.com への接続が確立しない場合、状態は*失敗した検証 (Failed Verification)* と表示されます。接続を確立した後、への接続が確立しませんでした。接続を確立し、サーバーを再開始するか、**[Activate]** ボタンを使用して)ライセンスをアクティブ化します。

メモ: ライセンス状態が*非アクティブ*または*ブロックされた*と表示されている場合、ステータスを説明したメッセージがメッセージログに追加されます。

製品のインストールにはアクティブな、または、保留されているライセンスのみを割り当てることができます。非アクティブなライセンスはアクティブ化されるか、またはライセンスプールから削除することができます。ライセンスがライセンスプールから削除された場合、ライセンスファイルを再度アップロードすることでアップロードできます。ライセンスファイルがアップデートされると、プールに存在しないライセンスのみがアップロードされます。ライセンスをアクティブ化、非アクティブ化、または削除するには、それぞれ、**[Activate]**、**[Deactivate]** または **[Delete]** ボタンをクリックしてください。

altova.com のマスターライセンスサーバーへの接続
Altova LicenseServer は、ライセンスに関連したデータを検証と認証し、Altova ライセンス使用許諾契約書への継続的な遵守を確認するため、altova.com のマスター Licensing Server と通信する必要があります。この通信は HTTPS を介して、ポート 443 を使用して行われます。altova.com のマスター Licensing Server との最初の検証の後、Altova LicenseServer が altova.com と 5 日間 (= 120 時間)、再接続できない場合、Altova LicenseServer は Altova LicenseServer に接続して Altova ソフトウェア製品を使用することを許可しません。

Altova マスターサーバーへの接続損失は Altova LicenseServer の構成ページの メッセージ (Messages) タブにログされます。更に、管理者は、altova.com への接続が失われた場合、自動的に警告の電子メールを送信するように Altova LicenseServer を構成することができます。電子メールの設定の変更は、構成ページの設定 タブで行うことができます。

ライセンスのアクティブ化、非アクティブ化、および削除

アクティブなライセンスは、ライセンスを選択して **非アクティブ化」(Deactivate)** をクリックすることで非アクティブ化することができます。使用されていないライセンスはアクティブ化 (**Activate** ボタン)または、削除 (**Delete** ボタン)することができます。ライセンスが削除された場合、ライセンスプールから除去されます。削除されたライセンスはライセンスファイルをライセンスプールにアップロードすることで、再度追加することができます。ライセンスが再アップロードされた場合、ライセンスプールに存在しないライセンスのみがライセンスプールに追加されます。既にライセンスプールに存在するライセンスは再度追加されません。

ライセンスプール (License Pool) タブのアイコン

Altova MissionKit ロゴ。デスクトップ製品ライセンスが MissionKit の一部である場合、Altova デスクトップ製品名の横に表示されます。*次を参照してください:デスクトップ製品のライセンスに関するメモ*。

割り当てられたクライアントの表示。 割り当たられたライセンスの*クライアント列* 内に表示されます。クライアントの登録されている製品のライセンスを管理する クライアント管理 タブに移動します。

実行中のクライアントの表示。現在さぢょう中のソフトウェアに割り当てられているライセンスの*クライアント列* 内に表示されます。ソフトウェアを差ぢょうしているクライアントマシンの*クライアントの監視*に移動します。ここで、選択されたクライアントと登録されたソフトウェアが表示されます。

情報の表示。割り当てられていないライセンスの*クライアント列*内に表示されます。ユーザーの人数、ライセンスがライセンスバンドルの一部であるか等のライセンスに関する情報を表示します。

ライセンス情報

次のライセンス情報が表示されます：

- *状態*：以下の値であることができます：*アクティブ化* | *失敗した検証* | *アクティブ* | *非アクティブ* *ブロックされた*。次を参照してください：ライセンスの状態 。
- *名前*、*会社*：ライセンスの名前と会社名です。この情報は、購入の際に提供された情報を基にしています。
- *製品*、*エディション*、*バージョン*：ライセンスされている製品のバージョンとエディションです。各列の一番上は、ライセンスをカテゴリ別にフィルターするコンボボックスです。
- *キーコード*、*バンドルID*：製品のロックを解除するライセンスキーです。単一のAltova MissionKit バンドル内の全ての製品は、バンドルID同じを有しています。バンドルされていない製品には、バンドルID は存在しません。
- *開始日*、*終了日*：ライセンスの有効期限を示します。有効期限の無いライセンスには、終了日がありません。
- *有効期限日数*、*SMP (残りの日数)*：ライセンスの有効期限が切れるまでの日数。ライセンスされている各購入には、特定の日数の間有効なサーポート & メンテナンスパッケージが付随します。*SMP* 列は、有効なSMP 日数を表示しています。
- #、*ライセンスの型*：カラム内にリストされている許可されているユーザー数またはCPU コア数。許可がユーザーまたはコアに与えられるかは、ライセンスの型カラムに表示されています。Altova MobileTogether Server 製品の場合、ライセンスは、MobileTogether Server に接続されているクライアントデバイスの数に基づいています。つまりサーバーの**ユーザー数**。その他のAltova サーバー製品では、ライセンスは**CPU コア数**のみをベースに割り当てられています (*下を参照*)。Altova デスクトップ製品の場合、ライセンス**ユーザー**の数をベースに割り当てられます。*デスクトップ製品のライセンスに関するメモ*を参照してください。
- *クライアント*：この列は、MobileTogether Server ライセンスとデスクトップ製品 ライセンスのためのみのエントリです。サーバー製品 ライセンスのためのエントリは存在しません。MobileTogether Server device ライセンスのために、ライセンスが割り当てられているかを表示しています。この列は、デスクトップ製品のために、列は、マシンの台数とユーザーの人数を下記に説明されるように表示します。

デスクトップ製品：マシンの台数 とユーザーの人数

- *マシンの台数* は、与えられたライセンスでソフトウェアを実行することできるライセンスされたマシンの台数を表します。例えば、`7/10` マシンは、ソフトウェアインスタンスを10台のマシンで使用することができ、現在 7台のマシンでソフトウェアのために使用されていることを意味します。 **「割り当てられているクライアントを表示」 (Show Assigned Client)** ボタンをクリックし、クライアント管理 タブに移動し、クライアントマシンのライセンスの詳細を確認します。
- *ユーザーの人数* は、許可されているユーザーの総数内の現在のユーザーの数を表しています。現在作動しているライセンスされたソフトウェアのインストールのみが数えられます。例えば、`3/10 users` は、使用を許可されている10名のユーザー中 3名のユーザーが現在ライセンスを使用していることを意味します。ライセンスされているソフトウェアのインストールが現在実行中の場合、 **「現在作動中のクライアント」(Show Running Client)** ボタンをクリックし、クライアントの監視タブ を開き、ネットワーク上のクライアントマシンで作動中のAltova 製品の詳細を確認します。
- *ユーザーの人数* と *マシンの台数* は共に、現在のライセンスの許容量と与えることのできるライセンスの使用状況についての情報を表します。例えばインストールされているユーザーライセンス のマシンの台数が、`7/10` であり、ユーザーの任数が`3/10` の場合、以下を意味します：(i) 製品 ソフトウェアは、10台のマシンにライセンスを与えることができます。 (i) ソフトウェアは、7台のマシンにライセンスを与えました。(ii) ライセンスを与えられた7台のソフトウェアのうち3台が現在作動中です。

ライセンスの割り当ての解除

マシン上のソフトウェアインストールからライセンスの割り当てを解除するには、 クライアント管理 タブに移動します。割り当てを解除するマシンとソフトウェアを選択します。 **「割り当てられたライセンスを編集する」** ボタンをクリックして、ライセンスの割り当てを解除し、 **「変更の適用」(Apply Changes)** をクリックします。

デスクトップ製品のライセンスに関するメモ

ユーザーライセンスには3つの種類があります：

- *インストールされているユーザー*：ライセンスがソフトウェアをインストールする台数分購入されます。 例えば、10台

分のユーザーインストールライセンスを購入すると、10台までのマシンにソフトウェアをインストールし使用することができます。各ライセンスを与えられているマシンでは、同時に複数のソフトウェアのインスタンスを開始することができます。各 「インストールされているユーザー」のためのライセンスは、そのマシン上で使用される製品を意味します。

- *同時に使用するユーザー*:この種類のライセンスは、同時に使用するユーザーの人数の10倍のコンピューターの台数にソフトウェアをインストールすることのできるライセンスです。すべてのインストールは、同じ物理ネットワーク上に存在しなくてはなりません。ソフトウェアは、同時に使用するユーザーの人数に対して許可されている数のみ使用することができます。例えば、同時にしようするユーザーのために10個のライセンスを購入したとします、この場合、ソフトウェアは200台までのコンピューターに同じ物理ネットワーク上でインストールすることができ、20台のコンピューター上で使用されることができます。*同時に使用するユーザー*ライセンスを異なる物理ネットワーク上で使用する場合、各ネットワークのために個別のライセンスを購入する必要があります。*同時に使用するユーザー*のライセンスを複数のネットワークで使用することはできません。
- *名前の与えられたユーザー*:名前の与えられているユーザーライセンスは、それぞれ5台のマシンまで、ソフトウェアをインストールすることができます。しかしながら、ライセンス内で名前が挙げられているユーザーのみがソフトウェアを使用することができます。このライセンスを使用すると、ソフトウェアが**1つのインスタンスのみ**を使用すると仮定される場合、ユーザーは異なるマシンで作業することができます。

Altova MissionKit ライセンスに関するメモ

Altova MissionKit は、Altova デスクトップ製品のパッケージです。Altova MissionKit ライセンスは、MissionKit パッケージ内で、各デスクトップ製品のための個別のライセンスから構成されています。これら個別の製品ライセンスには、異なる一意のキーコードが存在しますが、同一のMissionKit バンドルID を有します。Altova MissionKit ライセンスをライセンスプールにアップロードすると、(Altova MissionKit ロゴ が横に表示され)MissionKit を構成する各製品の個別のライセンスがライセンスプールに表示されます。これらの製品ライセンスの1つを特定のユーザーに割り当てると、MisisonKit バンドル内の他の全ての製品のライセンスもこのユーザーに割り当てられます。この結果、この特定のMissionKit バンドル内の他の製品を他のユーザーに割り当てることはできません。

ライセンスのチェックアウト

ライセンスが製品マシン上に保管されるように、ライセンスをライセンスプールから 30 日間チェックアウトすることができます。これにより、オフラインで作業することが可能になります。これはとても役に立ちます。Altova LicenseServer にアクセスできない環境 (例えば、旅行中にAltova 製品がインストールされたラップトップコンピューターで作業する) 場合などが挙げられます。ライセンスはチェックアウトされていますが、LicenseServer は、ライセンスが使用中と表示し、ライセンスは他のマシンで使用することができません。ライセンスはチェックアウトの期間が終わると自動的にチェックインされた状態を戻します。または、チェックアウトされたライセンスはソフトウェアのライセンスの認証ダイアログのボタンを使用して**チェックイン**することができます。ライセンスをチェックアウトするには、Altova デスクトップ製品の**ヘルプ**メニューに移動し、**ソフトウェアのライセンスの認証**を選択します。詳細に関しては Altova 製品のマニュアルを参照してください。

コアとライセンスについてのメモ

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 6CPU 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。

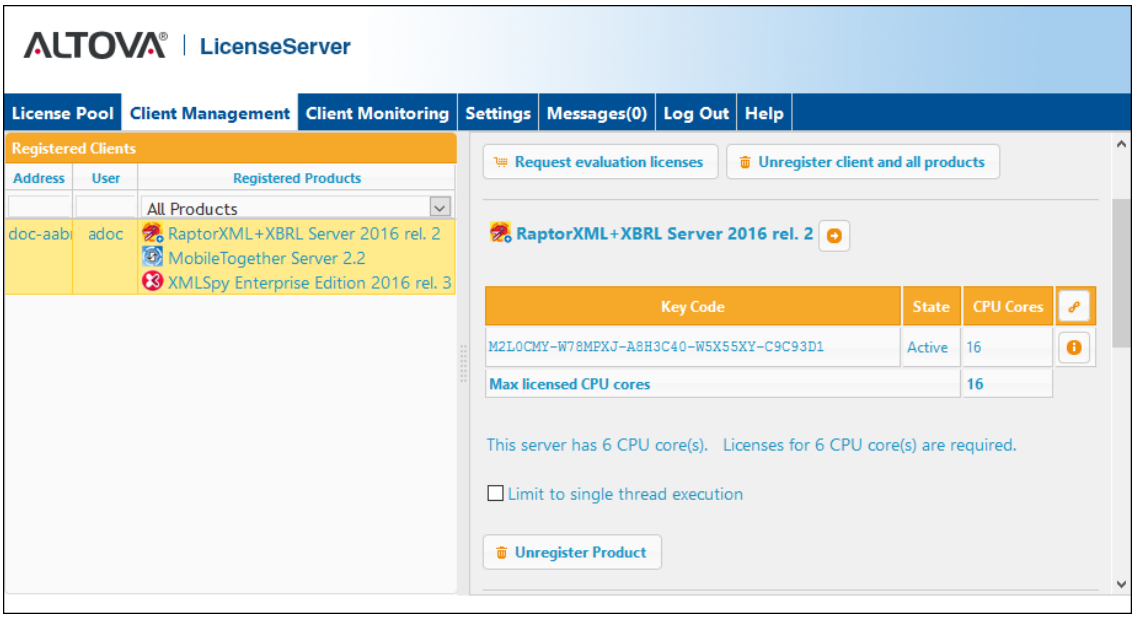
MobileTogether Server ライセンス
MobileTogether Server ライセンスには2つの種類があります。カスタマーは必要に応じてライセンスの種類を選択することができます。

- *コアライセンス*：サーバーマシンのコア数をベースにしてMobileTogether Servers に割り当てられます。上の例を参照してください。上の説明を参照してください。コアライセンスは、無制限の数量のMobileTogether クライアントデバイスによりサーバーへの接続を許可します。しかしながら、「単一スレッドの実行」チェックボックスがチェックされていると、1度にMobileTogether Server に接続できるモバイルデバイスは1台です。これは、評価と小さい規模のテストを行う際に役に立ちます。この場合、2台目のモバイルデバイスがMobileTogether Sever に接続される場合、ライセンスは2台目が使用するようになります。最初のデバイスは、接続できないようになり、エラーメッセージが表示されます。
- *デバイスライセンス*：MobileTogether Server にいつでも接続することのできるMobileTogether Client デバイスの最高使用数を指定します。

11.7.2 クライアント管理

このセクション:

「クライアント管理」(Client Management) タブ (*下のスクリーンショット*) は、2つのペインに分割されています:



- *登録されているクライアント:* 左側のペインは、LicenseServer に登録されている Altova 製品を少なくとも1つテーブルに表示します。このようなマシンは、*登録されているクライアント* と呼ばれます。各登録されているクライアントは、左側のペインに登録されている全ての商品をリストしています。LicenseServer に製品を登録する方法は、製品の登録で説明されています。このペイン内の表示は、ペインの列の上にフィルターを入力することによりフィルターできます。
- *製品のライセンス:* これは右側のペインです。登録されているクライアントが左側のペインで選択されると、(*登録されているクライアント*) クライアントの登録されている製品のライセンスに関する情報が右側のペインに表示されます。各登録されている製品のライセンスを管理することができます (この点については下で説明されています)。

クライアント管理タブ内のアイコン

割り当てられたライセンスの編集。 各製品のリストで使用することができます。新しいライセンスを製品に割り当てることのできる、すでに割り当てられたライセンスを編集できる 割り当てられたライセンスの編集 がポップアップします。

ライセンスの表示。各ライセンスに表示されます。License Pool タブに切り替えができ、選択されたライセンスをハイライトされることによりライセンスの詳細がわかります。

製品の登録解除。(選択されたクライアントマシン上の)各製品で利用可能です。選択された製品をLicenseServer から削除することができます。*製品の登録解除* を参照してください。クライアントとその全ての製品の登録解除を行うには、ペインの上の **「クライアントとその全ての商品の登録を解除する」(Unregister client and all products)** をクリックしてください。

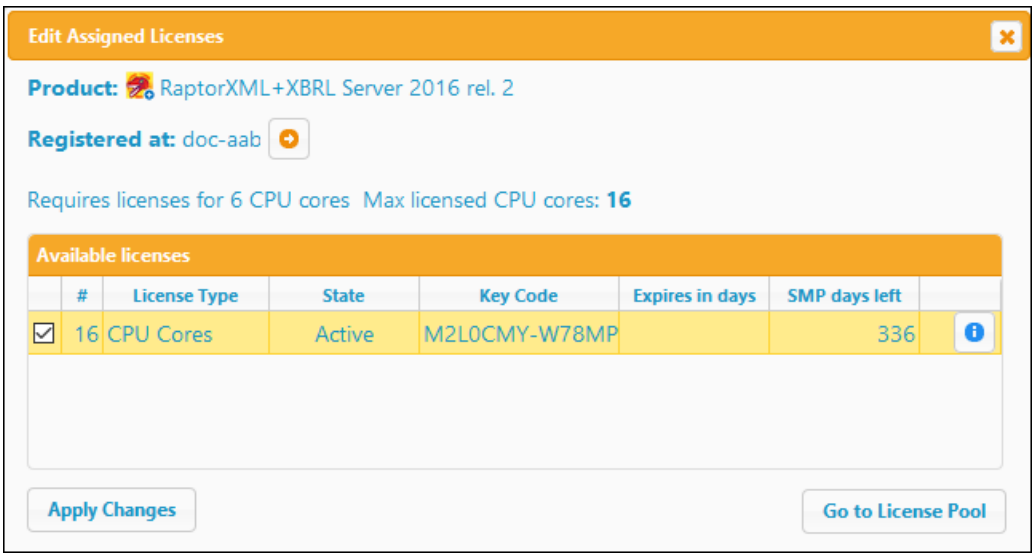
製品のリストペイン内のライセンスの管理

右側の *製品のライセンス* ペインでは、以下を行うことができます:

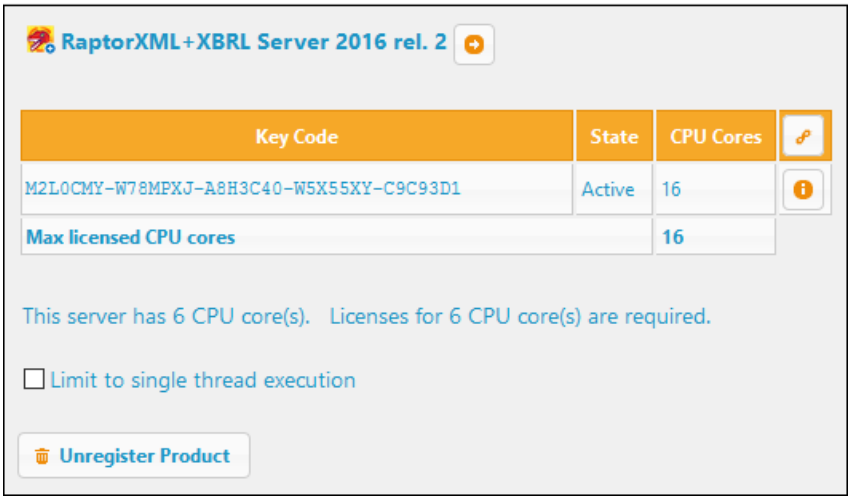
- *割り当て*、*割り当ての解除*、*製品のライセンスの変更*:製品の **「割り当てられたライセンスを編集する」(Edit Assigned Licenses)** ボタンをクリックして行います。次を参照してください:*ライセンスの割り当て*。各サーバー製品には、クライアント上で製品を作動するためにライセンスされることが必要なCPU コア数が表示されています。ライセンスされているコアが必要なコア数よりも少ない場合、情報は赤でマークされています(ライセンスされる必要があるCPU コアの数は、そのクライアント上のCPU コアの数で、LicenseServer によりクライアントマシンから取得されます)。
- *単一コア、クライアントの単一コアを使用するためのサーバー製品のセットアップ*:次を参照してください:*単一スレッドの実行*。
- *LicenseServer を製品から登録解除する*:製品の **「製品の登録解除」** ボタンを使用します。次を参照してください:*製品の登録解除*。

ライセンスの割り当て

登録されている製品にライセンスを割り当てるには、その製品の **「割り当てられたライセンスを編集する」(Edit Assigned Licenses)** ボタンをクリックしてください。これは、「割り当てられたライセンスを編集する」ダイアログを表示します(*下のスクリーンショット*)。



割り当てるライセンスを選択し、**「変更の適用」(Apply Changes)** をクリックします。ライセンスは、その製品に割り当てられ、クライアント管理タブの *製品のライセンス* タブ内に表示されます(*下のスクリーンショット参照*)。



単一スレッドの実行
ライセンスプール内で、製品ライセンスが、1つのコアのためにのみ有効な場合、複数のコアを持つマシンが1つのコアのライセンスに割り当てられることができます。このような場合、マシンはその製品を単一のコアで作動します。ですから、処理は、(複数のコアのみで可能な)複数のスレッドが使用できないため遅くなります。製品は、そのマシン上で単一のスレッドモードで実行されます。

単一コアのライセンスを複数のコアマシンに割り当てるには、その製品のために*単一スレッドの実行に制限*チェックボックスをチェックします。

MobileTogether Server (MTS) の場合、MTS コアライセンスのために単一スレッドの実行が選択されている場合、MobileTogether Server に接続することのできるモバイルデバイスは1台です。2台目のデバイスがMobileTogether Sever に接続されると、ライセンスは引き継がれます。最初のデバイスは接続することができなくなり、このためにエラーメッセージが表示されます。

異なる名前で使用される1つのクライアントマシン
クライアントマシンが1度以上 LicenseServer に登録されている場合、クライアント管理タブ内で複数の名前が表示される可能性があります。これは、複数のエントリが表示されるという意味です。これは、マシンが異なるフォームのホスト名で再登録された場合発生します。

追加ライセンスが同じマシンに異なる名前で登録されないように確認してください。(*製品のライセンス*) ペインの右上にある **『クライアントと全ての製品の登録を解除する』** ボタンを使用して余計なクライアントマシンの登録を解除してください。また、同じライセンスが同じマシンに複数回割り当てられるライセンスの競合が発生する場合があります。ですので、これら2つのシチュエーション(冗長ライセンスと単一ライセンスの複数回の割り当て)を回避するために、単一クライアントマシンの複数回のエントリは、登録解除されることが奨励されます。

クライアント管理タブ内で取られるマシン名の例です:

- *ドメイン名を持つホスト名(完全修飾されたドメイン名、FQDN)例:"Win80-x64_1.my.domain.com" または "Doc3.my.domain.com"。*これは(ドメイン情報を持つ、または、持たない)マシンのホスト名がLicenseServer に登録するために使用される `licenseserver` CLI コマンドの引数としてパスされた場合に発生します。例:`<AltovaServerProduct> licenseserver Doc3`. これは以下を含むFQDN を作成します:`Doc3.my.domain.com`.

  FQDN は、また`localhost` がWindows 7 と10システム上でホスト名として与えられた場合に生成されま

す。

- *ドメイン名を持たないホスト名。例：Win80-x64_1" または "Doc3"。*これは、Windows 8 システム上で `localhost` がマシン名として与えられた場合、発生します。
- *localhost*. 一部の場合、`localhost` は、マシン名として表示されます。

メモ: Windows マシンにAltova サーバー製品をインストール中、マシンが自動的にLicenseServer に登録される場合、`localhost` がインストーラーマシン名として使用されます。

評価ライセンスのリクエスト

30-日間無料の評価ライセンスをクライアントにインストールされているLicenseServer に登録されているAltova 製品のために取得することができます。ペインの右上にある **「評価ライセンスのリクエスト」(Request Evaluation Licenses)** ボタン (*製品のライセンス*) をクリックします。 (クライアントマシン上の)LicenseServer に登録されているAltova 製品のリストを含むダイアログが表示されます。評価ライセンスを必要とする製品がチェックされ選択されていることを確認し、登録フィールドに記入し、リクエストを送信します。 30 日間有効な 評価ライセンスが含まれる電子メールをAltova から受信します。 サーバー製品に関しては、リクエストが送信された時点で製品が必要とする有効なコア数が含まれます。 ライセンスをディスクに保存して、ライセンスプールにアップロードします。

製品の登録解除

LicenseServer に登録されている各 Altova 製品 が右側のペイン (*製品のライセンス*) で クライアントマシン名の下に表示されます。 **「製品の登録解除」(Unregister Product)** ボタンがエントリの下に表示されています。 LicenseServer から製品の登録を解除するためにこのボタンをクリックします。製品にライセンスが割り当てられている場合、割り当ては、製品の登録が解除されると解消されます全ての製品の登録を解除するには、*(製品のライセンス)*ペインの右上にある **「クライアントと全ての製品の登録を解除する」(Unregister client and all products)** ボタンをクリックしてください (*このセクションの最初のスクリーンショットを参照してください*)。

LicenseServer から登録の解除を行うには、以下を行います：

- *サーバー製品*：サーバー製品のWeb UI 内の設定ページに移動し、サーバー製品にWeb UI が存在しない場合は、コマンドプロンプトウィンドウを開き、製品のCLI を使用して登録します。各製品のための手順は以下で説明されています: FlowForce Server の登録, MapForce Server の登録、MobileTogether Server の登録、StyleVision Server の登録、とRaptorXML(+XBRL) Server の登録。
- *デスクトップ製品*: 製品の ソフトウェアのライセンス認証 ダイアログ 「**Help | ソフトウェアのライセンス認証」(Help | Software Activation)** により**LicenseServer モードからソフトウェアの認証に切り替えます。** *Altova LicenseServer* フィールド内から、製品を登録するLicenseServer を選択します。製品は、LicenseServer の クライアント管理タブ内の登録された製品リスト内にに表示されます。

詳細に関しては、次のセクションを参照してください：登録された製品へのライセンスの割り当て。

11.7.3 クライアントの監視

クライアントの監視 タブにより選択されたクライアントマシンの概要を確認することができます。タブには以下が表示されます：

チェックアウトされているクライアント
(サーバー製品ではなく) XMLSpy または MapForce などの、Altova デスクトップ製品 ,のエンドユーザーは、LicenseServer に登録されているライセンスをチェックアウトすることができます。エンドユーザーのマシンが一定の期間オフラインであることが想定される場合、この機能が使用されます。LicenseServer からライセンスをマシンがオフラインである一定の期間チェックアウトすることができます。この期間内で、エンドユーザーは Altova デスクトップ製品を LicenseServer に通信を取ることなく使用し続けることができます。現在チェックアウトされているライセンスとユーザーは、この見出しと共にリストされます。

メモ: エンドユーザーは、Altova デスクトップ製品のソフトウェアのライセンスの認証ダイアログ (**Help | Software Activation**) によりライセンスをチェックアウトすることができます。

実行中のクライアント
現在クライアント上で実行されている Altova 製品のリストです。製品の複数のインスタンスが実行されている場合、これらのインスタンスがリストされています。

Running Clients

Product	Edition	Version	User	Address	State	Failover	Last seen (seconds ago)	
RaptorXML+XBRL Serv		2016 rel. 2	DOBRA	doc-aab	Running		8	
XMLSpy	Enterprise Editic	2016 rel. 3	adoc	doc-aab	Running		11	

メモ: Failover LicenseServers は、v2015rel3 または以降であるクライアントアプリケーションと作動します。(Altova MobileTogether Server の場合、バージョン 1.5 または以降)。古いクライアントにはフラグが立てられます。

メモ: フェールオーバー LicenseServers は、v2015rel3 または以降であるクライアントアプリケーションと作動します。(Altova MobileTogether Server の場合、バージョン 1.5 または以降)。古いクライアントにはフラグが立てられます。

クライアントの監視タブ内のアイコン

ライセンスの表示。製品のインスタンスに表示されます。License Pool タブに切り替えができ、選択された製品のインスタンスがハイライトされることによりライセンスの詳細がわかります。

クライアントの管理。各 製品のインスタンスに表示されます。 クライアント管理 タブに切り替えができ、選択された製品のインスタンスをハイライトします。

11.7.4 設定

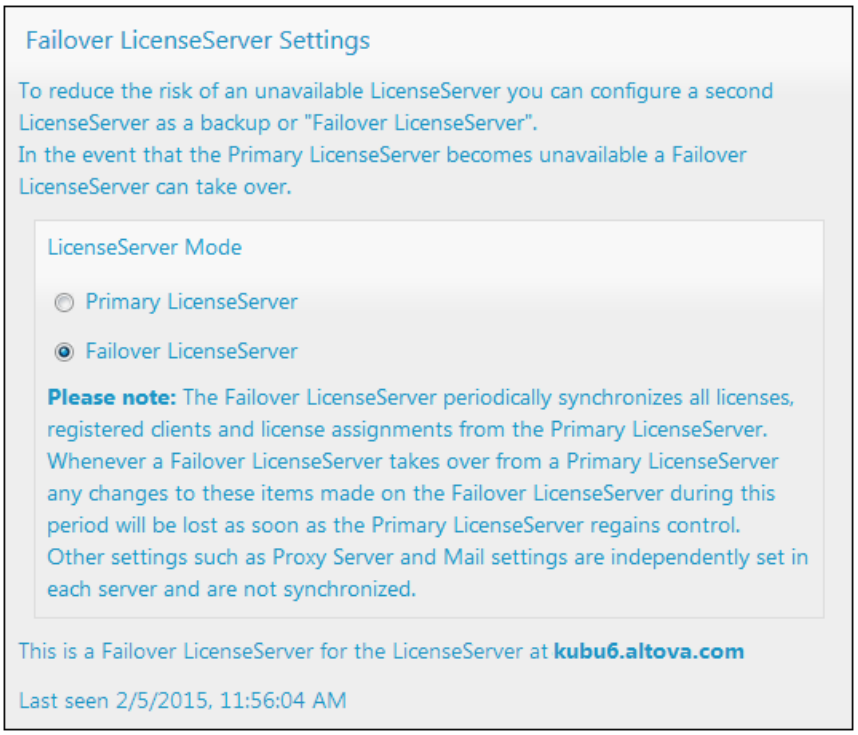
このセクション:

設定 タブに関しては下で説明されています。次を設定することができます:

- **LicenseServer をシャットダウンするまでの待ち時間。** シャットダウンは、通常サーバーのメンテナンスのために実行されます。シャットダウンする時間は、Altova デスクトップ製品を実行中のクライアントの作業を減らすために使用することができます。選択されたシャットダウンタイムは、シャットダウンの最長の時間です。デスクトップ製品を作動するクライアントにLicenseServer が接続されていない場合、LicenseServer は即時シャットダウンされます。シャットダウンまでの待ち時間は、**「シャットダウン」**をクリックすると開始されます。シャットダウンをキャンセルするには、**「シャットダウンの中断」**をクリックします。LicenseServer のシャットダウン中に、クライアントの作動を有効化するには、**フェールオーバー LicenseServer**を構成してください。
- プライマリLicenseServer が使用できなくなった場合、2番目のLicenseServer が、プライマリ LicenseServer から引き継ぐように構成することができます。この2番目のLicenseServer は、**フェールオーバー LicenseServer** と呼ばれます。この2番目のLicenseServer は、**フェールオーバー LicenseServer** この設定の指定方法はここから確認することができます。
- この2番目のLicenseServer にログインするためのパスワード。希望するパスワードを入力し、**「パスワードの変更」(Change Password)** をクリックします。
- Altova への接続をテストするには、**「Altova への接続をテストする」(Test Connection to Altova)** をクリックします。接続をテストする前に、ページの下の **「保存」(Save)** ボタンをクリックして 新しい設定を保存する必要があることにご注意してください。**「Altova への接続をテストする」(Test Connection to Altova)** ボタンは、テスト中は無効化されており、テストが完了すると有効化されます。
- ウェブベースの構成ページ (Web UI) のためのネットワーク設定は、(存在する場合) インターネットに接続するために使用されるプロキシサーバー、およびライセンスサービスの使用のためです。これらの設定に関しては下の ネットワーク設定 で説明されています。
- 電子メールサーバー設定と、LicenseServer に関する重要な事項が発生した場合に電子メールが送信される宛先です。これらの設定に関しては下の 電子メールの設定の変更 で説明されています。
- 設定を変更した後、ページの下の **「保存」**をクリックします。変更された設定は、保存されるまで効果が適用されません。

フェールオーバー LicenseServer 設定

プライマリLicenseServer が使用不可能になった場合、プライマリLicenseServer から第 2 LicenseServer への LicenseServer 切り替えを構成することができます。この第 2LicenseServer は **フェールオーバー LicenseServer** と称されます。



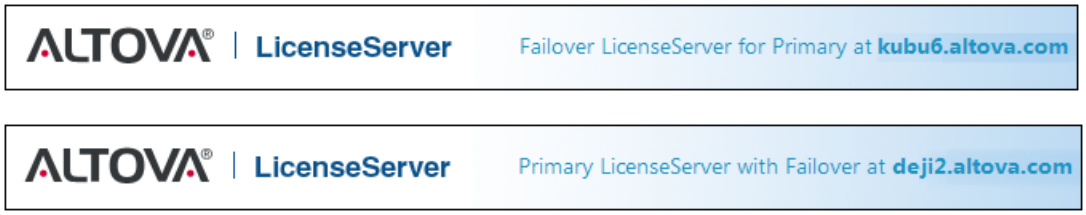
LicenseServer をフェールオーバーLicenseServer と設定するには、以下をおこないます：

1. インストールセクションの指示に従いLicenseServer をインストールします。
2. LicenseServer のモードを、対応するラジオボタンを選択して、*フェールオーバーLicenseServer* に設定します。（*上のスクリーンショットを参照*）。（デフォルトでは、LicenseServer モードは*プライマリLicenseServer* に設定されています。）
3. 表示される*プライマリ*LicenseServer を検索ダイアログ（*下のスクリーンショット*）にフェールオーバーLicenseServer によりバックアップされる*プライマリ*LicenseServer を入力します。手順は以下の方法で行うことができます：(i) **「LicenseServers の検索」(Search for LicenseServers)** をクリックして、コンボボックス内で検索されたLicenseServer リストからバックアップするLicenseServer を選択します。(ii) **手動で アドレスを入力」(Manually Enter Address)** をクリックして、バックアップするLicenseServer のアドレスを入力します。プライマリLicenseServer を入力して、 **手動でLicenseServer へ接続」(Connect to Primary LicenseServer)** をクリックします。

4. 確認ダイアログが表示され、現在のLicenseServer が選択されたプライマリLicenseServer のフェールオーバーLicenseServer として設定するかが問われます。確認すると、インストールされたライセンスと登録されたクライアントが削除されます。続行する場合は **「はい」** をクリックします。続行を確認することは、インストールされたライセンスを削除し、現在のLicenseServer から登録されたクライアントの登録を解除することに注意してください。

フェールオーバーLicenseServer が構成されると、プライマリLicenseServer とフェールオーバーLicenseServer の

双方に対するモードに関する通知が構成ページの上に表示されます。下のスクリーンショットで、最初に フェールオーバー LicenseServer が、次に プライマリLicenseServer が表示されています。

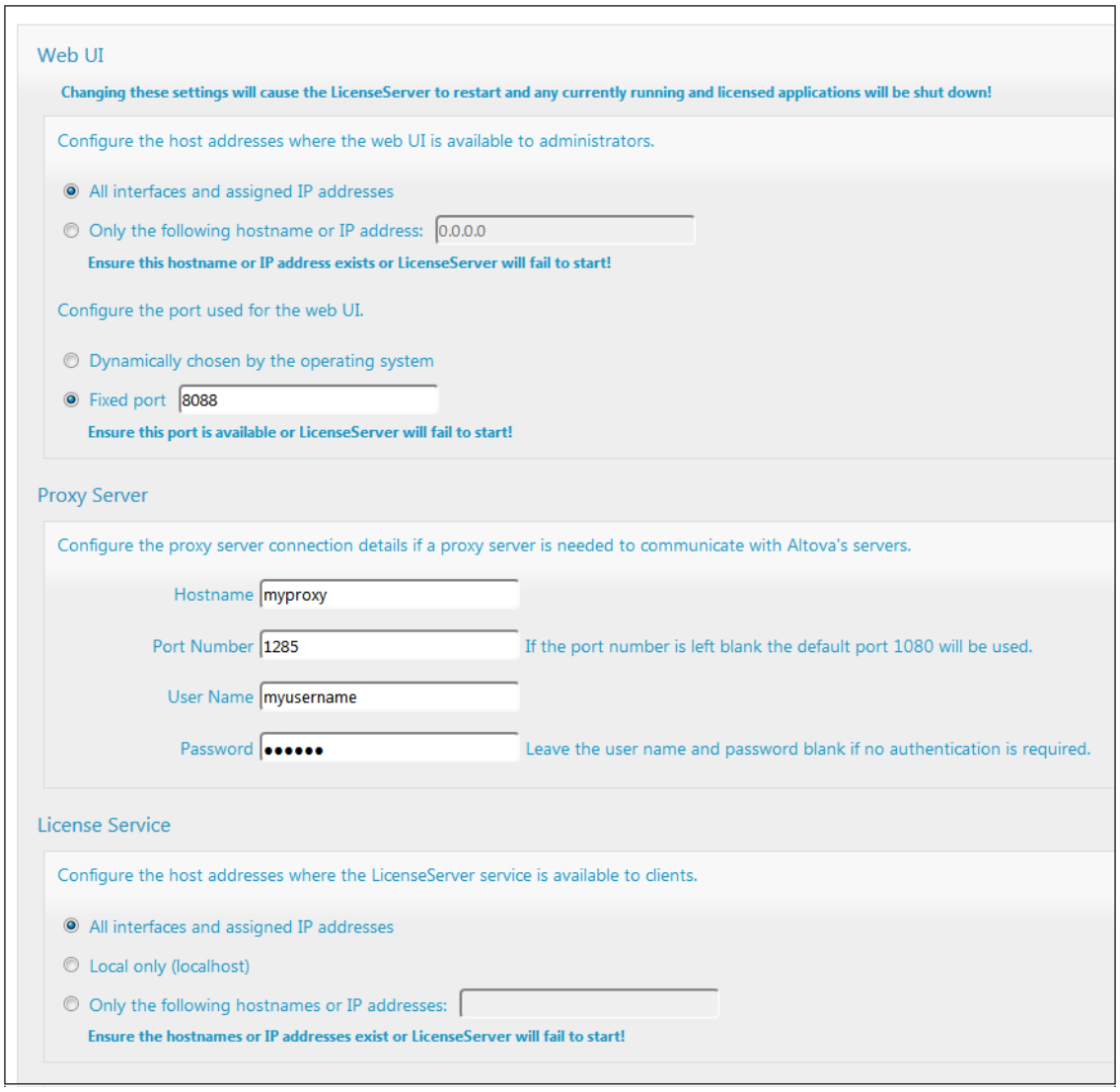


以下の点に注意してください:

- フェールオーバーLicenseServer が構成された後に、定期的にプライマリからのすべてのライセンス、登録されたクライアント、使用許諾契約を同期化します。プライマリが使用不可能になると、フェールオーバーが LicenseServer の役割を引き継ぎます。プライマリが再び使用可能になると、プライマリがフェールオーバーを引き継ぎます。 フェールオーバーに加えられたライセンスに関連する変更は、プライマリが管理を再び開始すると失われます。
- フェールオーバーLicenseServer は、 2015rel 3以降、(Altova MobileTogether Server の場合は、v 1.5 または以降)のバージョンのクライアントのみにライセンスを提供します。プライマリLicenseServer (下のスクリーンショット)の クライアント管理タブ で古いクライアントはフラグされます。フェールオーバーLicenseServer 機能を使用する場合は、クライアントのアプリケーションを 2015rel 3 以降にアップグレードするか、または、後に行ってください。(Altova MobileTogether Server の場合は、v 1.5 または以降)

ネットワークの設定

管理者はLicenseServer 構成ページおよびLicenseServer にポイントされるネットワークアクセスを指定することができます。



- *Web UI:* 許可されたIP アドレスの、すべてのインターフェイス、マシンのIP アドレス、固定アドレス、ポートは動的に計算されるか固定されることができます。これにより広範囲のIP-アドレス、ポート設定が許可されます。デフォルトのポート設定は**8088** です。
- *プロキシ サーバー (v1.3 以降使用可能):* プロキシサーバーがインターネットに接続する際使用される場合、プロキシサーバーの詳細はプロキシサーバーペインに入力される必要があります (*上部スクリーンショット参照*)。これらのフィールドはプロキシサーバーが使用時のみ記入される必要があります。プロキシサーバーを使用するために LicenseServer を構成するには、プロキシサーバーのホスト名と、必要であれば、ポート番号を入力します。プロキシサーバーが認証を必要としない場合、ユーザー名とパスワードのフィールドは空白にしておくことができます。
- *License サービス:* がインストールされているマシンは、1つまたは複数のネットワークインターフェイスから、複数または1つのネットワークに接続することができます。それぞれのネットワークで、License Server マシンはホスト名と IP アドレスにより検出されます。*License Service* 設定により、どのネットワークライセンスサービスを使用することができるか知ることができます。`localhost` オプションは、ローカルマシンのみでのサービスを許可します。ホスト名またはおよびIP アドレスをリストする場合、スペースを使用せず、コンマのみでリストを区切ります。 (例 : `hostname1,IPAddress1,hostname2`)。サービスのポート番号は、**35355** に固定されています。

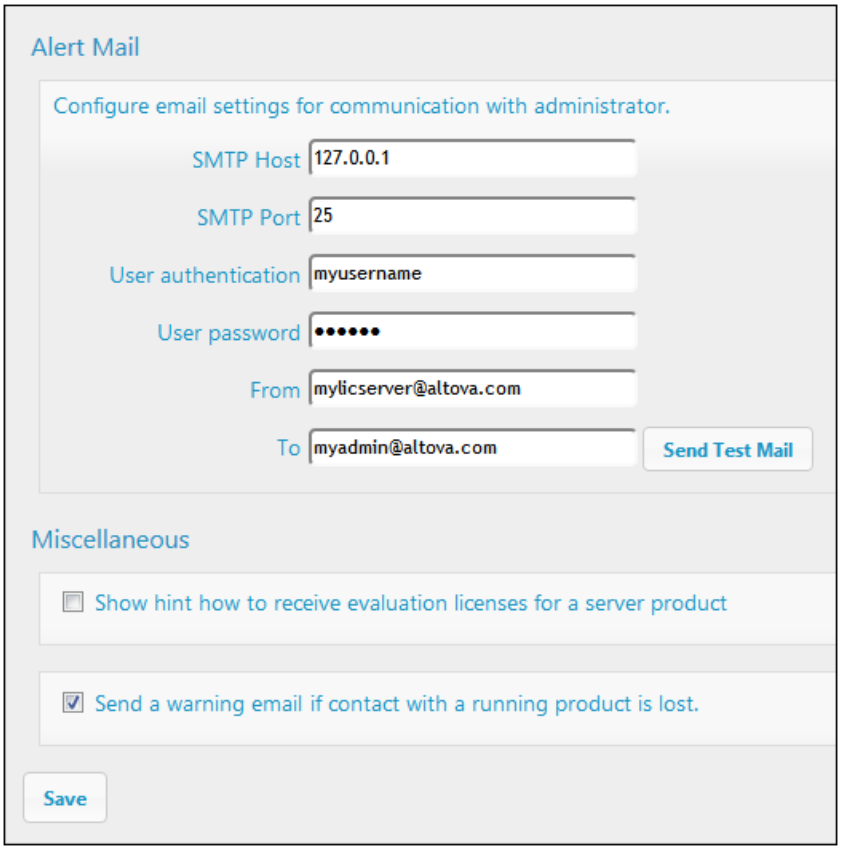
デフォルトでは、これらの設定はLicenseServer とLicenseServer が接続されているネットワークの構成ページへの制限のないアクセスを許可します。LicenseServer または、構成ページへのアクセスを制限したい場合は、適切な設定を入力して**[保存 (Save)]** をクリックしてください。

接続テスト (*上部参照*) 実行して設定が正しいか確認してください。

メール通知の設定

Altova LicenseServer は、`altova.com` サーバーに接続されている必要があります。接続が24*5時間 (5日間) 途切れた場合、LicenseServer はライセンスを許可しません。この結果、LicenseServer にライセンスされたAltova製品との作業セッションが失われる可能性があります。

接続エラー状態を管理者に通知するために、通知メールを電子メールアドレスに送信することができます。管理者の電子メールアドレスに通知メールを送信する通知メールペイン (*下のスクリーンショット参照*) に設定を入力します。



SMTP ホストおよびSMTP ポートは電子メール通知が送信される電子メールサーバーのアクセスの詳細です。 ユーザー認証 (*User Authentication*) とユーザーパスワード (*User Password*) は電子メールサーバーにアクセスするためのユーザーの資格情報です。*From* フィールドに電子メールの送信者の電子メールアカウントのアドレスを入力します。*To* フィールドは受信者の電子メールアドレスを入力します。

完了すると**[保存 (Save)]** をクリックしてください。メール通知の設定タブを保存した後、電子メール通知が、`altova.com` への接続エラーなどの重大な出来事が起きた際に指定されたアドレスに送信されます。この様なエラーの際はメッセージタブにも記録されますので、メッセージタブで確認することもできます。

その他の設定

評価ライセンスの受け取りとデプロイのヒントの表示
構成ページ下部のこのボックス (*上のスクリーンショット参照*) をチェックすることにより、簡単な評価ライセンスを評価しデプロイする簡単な説明が表示されます。

作動している製品とのコンタクトエラーが発生した場合に警告電子メールを送信する
ライセンスされ作動している製品とのコンタクトエラーが発生した場合、*From* アドレスから警告メッセージが送信されます。

11.7.5 メッセージ、ログアウト

メッセージ (Messages) タブは LicenseServer のライセンスプール内のライセンスに関連したにすべてのメッセージを表示します。各メッセージには **削除 (Delete)** があり、特定のメッセージを削除することができます。

ログアウト (Log Out) タブはログアウトボタンとして機能します。タブをクリックすることにより、すぐにログインマスクが表示されます。

11.8 パスワードのリセット

LicenseServer パスワードを忘れた場合、から**passwordreset** コマンドを使用してパスワードをデフォルトにリセットすることができます。

1. コマンド ライン ウィンドウを開く
2. LicenseServer アプリケーションまたは実行可能ファイルがインストールされているディレクトリに変更する
3. 次のコマンドを入力する:**licenseserver passwordreset**
   これによりLicenseServer 管理者のパスワードを`default` に設定します
4. 管理者にパスワード`default` を使用して、ログインすることができます。

Index

.

A

C

F

H

J

L

M

P

R

S

W

X

Z