Altova MapForce 2021 Basic Edition

ユーザーマニュアル

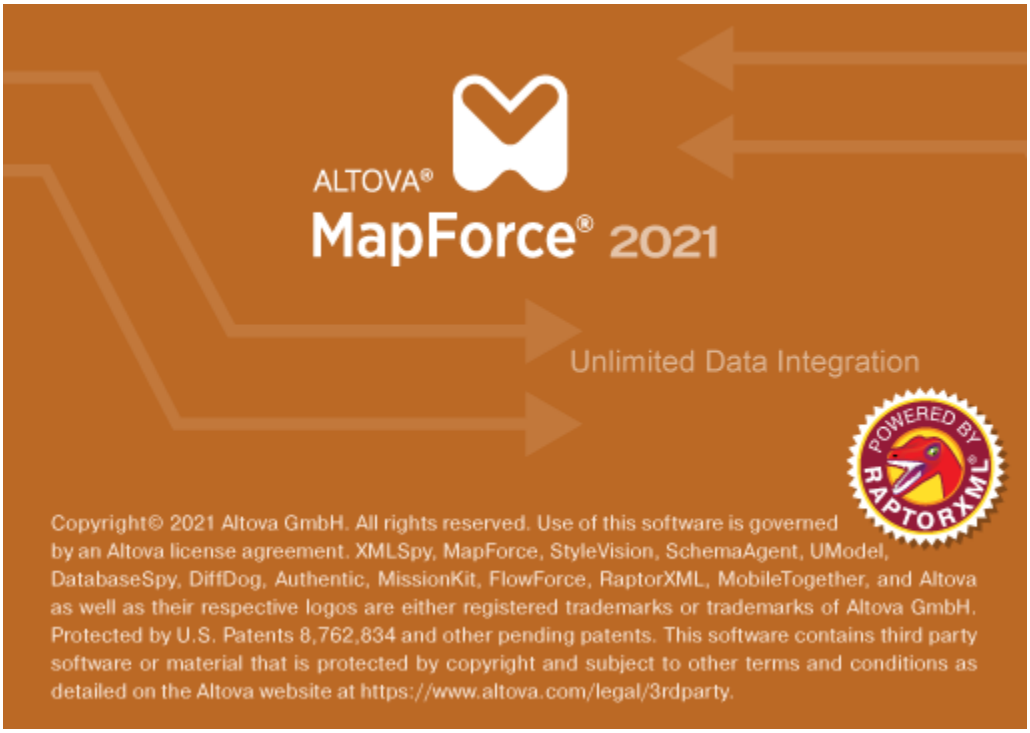
Altova MapForce 2021 Basic Edition
ユーザーマニュアル

公開日: 2015-2021

目次

1 はじめに

MapForce® 2021 Basic Edition は高度なデータ統合プロジェクトにて決定的な役割を果たす視覚的なデータマッピングツールです。MapForce® は、プラットフォーム更新済みのW indows 7 SP1、W indows 8、W indows 10 とプラットフォーム更新済みのW indows Server 2008 R2 SP1 またはは以降 上で作動する32/64-bit W indows アプリケーションです。64 ビットサポートは、Enterprise とProfessional エディションでご利用いただくことができます。



最終更新日: 2021 年 02 月 24 日

1.1 新規機能

MapForce 2021 リリース2 の新規機能:
を参照してください。

- XSLT 3.0 はマッピング言語としてサポートされるようになりました。XSLT コードの生成 を参照してください。また、MapForce にはマッピング言語がXSLT 3.0 である場合サポートされる新規のビルトイン関数が搭載されています。関数ライブラリレファレンスを参照してください。
- 内部のアップデートと最適化。

MapForce 2021 の新規機能:

- 内部のアップデートと最適化

2020 Release 2 の新規機能:

- 新規のライブラリの管理 ウィンドウによりドキュメントまたはプログラムレベルでインポートされたすべての関数ライブラリを表示管理できるようになりました(これにはMapForce ユーザー定義関数と他のライブラリが含まれています)。例えば、1つのマッピングから他のマッピングに簡単にユーザー定義関数をコピーして貼り付けることができます。マッピング間でUDF をコピーし張り付ける用法を参照してください。
- マッピングファイルがライブラリをインポートする際に、インポートされたライブラリファイルのパスはデフォルトで相対パスです。相対的なライブラリパスを参照してください。以前のリリース同様アプリケーションレベルでマッピングファイルをインポートすることは可能ですが、この場合ライブラリのパスは絶対パスです。
- マッピングファイルがXSLT ライブラリをインポートする場合、相対的なパスを使用してインポートされたライブラリを参照するXSLT コードを生成することができます。新規のオプションはマッピング設定 ダイアログボックス内で使用することができます。
- 内部のアップデートと最適化。

MapForce リリース2020 の新規機能:

- ルックアップテーブルを使用して値を置き換える場合、テーブルデータ(キー値ペア) をCSV またはExcel などの外部ソースからマッピングに張り付けることができます。また、代替の使用を必要としない値や不足する関数など定義済みのルックアップテーブル内で値が見つからない場合、このような値の処理が `substitute-missing` 関数の使用を必要としないため扱いが簡単になります。値マップの使用を参照してください。
- 内部のアップデートと最適化

MapForce 2019 リリース3 の新規機能:

- MapForce からJava 仮想マシンを明示的に設定するためのサポート。Java 設定 を参照してください。
- 内部のアップデートと最適化

MapForce リリース2019 の新規機能:

- 内部の更新と最適化。

MapForce 2018 リリース2 の新規機能:

- ビルトイン関数 と定数 をマッピングの空のエリアをクリックして便利に追加できるようになりました。
- 内部のアップデートと最適化

MapForce リリース2018 の新規機能:

- 内部のアップデートと最適化

MapForce 2017 リリース3 の新規機能:

- 内でのテキストの検索オプション。「出力 ペインと「XSLT」 ペイン が強化されました(次を参照: テキストビュー内の検索)。また、テキストのハイライトも上記のペインで使用することができます(次を参照: テキストのハイライト)。
- 内部のアップデートと最適化

MapForce リリース2017 の新規機能:

- ソースXML からノード名を読み取るを読み取ることは可能でターゲットにこの情報をマップすることができます。また、動的に新規の XML 属性、または要素をソースから与えられた値をベースにしたターゲット内に作成することもできます。ノード名のマッピング を参照してください。
- 要素レベルでカスタム化された名前空間を持つXML インスタンスファイルを作成することができます(次を参照してください: カスタム名前空間の宣言)
- 内部のアップデートと最適化

MapForce 2016 R2 の新規機能:

- XSLT ペイン 内の更に直感的なコードの使用: 折りたたまれたテキストが、省略シンボルと共に表示され、ツールヒントとして、プレビューすることができます。
- アクティブなマッピング内で関数が使用される全ての箇所を検索することができます(ライブラリウィンドウ 内の関数を右クリックし、「全ての呼び出しを検索する」を選択します)。
- 内部のアップデートと最適化

MapForce リリース2016 の新規機能:

- XSLT 1.0 コードの改善された生成(生成されたスタイルシートは読み込みやすく、より早く実行することができます)。
- 新規 集計関数がMapForce コアライブラリで使用することができます: `min-string` と `max-string`。これらの関数は文字列のシーケンスから最大値および最小値を取得することができます。

MapForce バージョン 2015 R4 の新規機能:

- 内部的な更新と最適化

MapForce バージョン 2015 R3 の新規機能:

- XML 出力内の `<?xml ... ?>` 宣言を抑制するオプション
- 新しいコンポーネント型: 単純型出力
- 内部的な更新と最適化

MapForce リリース2015 の新規機能:

- 新規 language 引数はformat-date とformat-dateTime 関数で使用することができます
- 新規 シーケンス関数: replicate-item

MapForce バージョン 2014 R2 の新規機能:

- 新規 シーケンス関数: シーケンスの生成、item-at など。
- 出力コンポーネント内のCDATA セクションを定義する機能のためのタイムアウトの値を定義する機能
- コンポーネントの削除後接続を保存する方法
- ターゲットコンポーネント内の必須アイテムの自動ハイライト

MapForce リリース 2014 の新規機能:

- RaptorXML 検証への統合と、XML Schema 1.1への基本的なサポート
- 新規 RaptorXML XSLT エンジンの統合
- XML スキーマワイルドカードへのサポート、xs:any とxs:anyAttribute
- XML ターゲットコンポーネント内のコメントと処理命令へのサポート

MapForce バージョン 2013 R2 SP1 の新規機能:

- 高速変換エンジン

MapForce バージョン 2013 R2 の新規機能:

- 内部的な更新と最適化

MapForce リリース 2013 の新規機能:

- 内部的な更新と最適化

MapForce バージョン 2012 R2 の新規機能:

- XSLT 2.0、XQuery、内蔵の実行エンジンにてソートコンポーネント をサポート
- ユーザー定義コンポーネント名

MapForce リリース 2012 の新規機能:

- ウィンドウマッピングウィンドウにおけるコンポーネントの自動配置
- ターゲット親 ノードへの接続を通知
- マッピングにてコンポーネントのシーケンスを制御するルール

MapForce バージョン 2011R3 の新規機能:

- 中間変数

MapForce バージョン 2011R2 の新規機能:

- ライブラリウィンドウライブラリウィンドウにおける検索機能
- 反転 マッピング
- 拡張可能な IF-ELSE 関数
- Core ライブラリにおけるノード名ならびに解析関数

MapForce リリース 2011 の新規機能:

- マッピングチェーンにて、中間コンポーネントのプレビュー(パススループレビュー)
- サポートされている全ての言語において dateTime と numbers のフォーマット機能が利用可能に
- 自動連番 機能の強化

MapForce バージョン 2010 リリース 3 の新規機能:

- Nillable 値 ならびに XML インスタンスファイルにおける xsi:nil 属性をサポート
- XML ドキュメントにてターゲットに対する自動キャスティングを無効化

MapForce バージョン 2010 リリース 2 の新規機能:

- 親コネクタの接続時に、対応する子接続を自動的に接続
- 入力文字列のトークン化により便なる処理

MapForce リリース 2010 の新規機能:

- コンポーネント毎に複数の入力 / 出力サポート
- 相対パス サポートのアップグレード
- xsi:type のサポートによる派生型の使用
- 新たに追加されたデータ型システム
- 改善されたユーザー定義関数のナビゲート (ユーザー定義関数のナビゲートを参照してください)
- XML 要素における 複合コンテンツ の処理を強化

MapForce バージョン 2009 SP1 の新規機能:

- ユーザー定義関数内のパラメーターの順序はユーザーにより定義されることができます (ユーザー定義関数内のパラメーターを参照してください)
- XML スキーマに対して妥当ではない XML ファイルの処理
- 全般的(標準的)なユーザー定義関数における複雑な階層構造の引数サポート(インラインと正規ユーザー定義関数を参照してください)

MapForce バージョン 2009 の新規機能:

- ノードコンテンツやノードのグループ化
- シーケンス内の ノードの位置 をベースにしたデータのフィルタリング
- QName サポート

- コンポーネント内にあるアイテム/ノードの検索

MapForce バージョン 2008 リリース 2 の新規機能:

- XML ファイルのために XML スキーマを自動生成
- Altova グローバルリソースのサポート
- パフォーマンスの最適化

MapForce バージョン 2008 の新規機能:

- 集計 関数
- Value-Map 検索コンポーネント
- XML 出力オプションの強化: XML 出力の整形、個々のコンポーネントに対してのエンコーディング設定や XML スキーマ参照の省略
- 内部における様々なアップデート

1.2 サポートメモ

MapForce® は32/64 ビット W indows アプリケーションで次のオペレーティングシステムで作動します:

- プラットフォーム更新済みのW indows 7 SP1、W indows 8、W indows 10
- プラットフォーム更新済みのW indows Server 2008 R2 SP1 またはは以降

64 ビットのサポートはEnterprise とProfessional エディションでご利用いただくことが出来ます。

生成されたコード内のサポートされる機能

MapForce Basic Edition 内の各言語内のコード生成とサポートに関連する機能を次のテーブルはリストしています。

機能	XSLT 1.0	XSLT 2.0	XSLT 3.0
マッピングにパラメータを提供する	●	●	●
マッピングから動的に入力ファイル名を提供する	●	●	●
マッピング入力としてワイルドカードファイル名を提供する[1]		●	●
マッピングから動的に出力ファイル名を生成する		●	●
マッピングから文字列の値を返す	●	●	●
変数		●	●
コンポーネントの並べ替え		●	●
関数のグループ別け		●	●
フィルター	●	●	●
Value-Map コンポーネント	●	●	●
動的なノード名		●	●

脚注:

1. XSLT 2.0、XSLT 3.0 とXQuery は `fn:collection` 関数を使用します。Altova XSLT 2.0、XSLT 3.0 と XQuery エンジン内の実装はワイルドカードを解決します。他のエンジンは異なる振る舞いをする可能性があります。

1.3 MapForce とは？

Altova Web サイト: データマッピングルール

MapForce は、プログラムコードを作成することなく視覚的なドラッグアンドドロップを使用するグラフィカルなユーザーインターフェイスを用いてデータを異なるデータ間、スキーマ間で変換する Windows ベースの多目的の IDE（統合された開発環境 統合開発環境）です。また、事実上、MapForce は時際のデータ変換（またデータマッピング）を行うプログラムコードを生成します。プログラムコードを生成しない場合、（MapForce Professional または Enterprise Editions で使用することのできる）MapForce 内蔵の変換言語 を実行します。

MapForce を使用してデザインされたマッピングによりデータを便利に、ファイルベース、または他のフォーマットに変換することができます。使用するテクノロジーに関わらず MapForce は通常自動的にデータの構造を決定、またはデータのためにスキーマを提供するオプションを与えます。MapForce はサンプルインスタンスファイルから自動的に生成することもできます。例えば、XML インスタンスファイルが存在し、スキーマ定義が存在しない場合、MapForce はスキーマ定義を生成し、他のファイル、またはフォーマットのために XML ファイル内のデータを使用できるようにします。

マッピングソースまたはターゲットとしてサポートされるテクノロジーは、以下のとおりです。

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XML と XML スキーマ	• XML と XML スキーマ • コンマで区切られた値（CSV）と固定長フィールドフォーマット（FLF）を含むフラットファイル • データベース（Microsoft Access や SQLite データベースなどを含む全てのメジャーなデータベース） • バイナリファイル（生 BLOB コンテンツ）	• XML と XML スキーマ • コンマで区切られた値（CSV）と固定長フィールドフォーマット（FLF）を含むフラットファイル • MapForce FlexText を使用してマップし、他のフォーマットに変換することのできるレガシーテキストファイルからのデータ。 • データベース（Microsoft Access や SQLite データベースなどを含む全てのメジャーなデータベース） • （UN/EDIFACT、ANSI X12、HL7、IATA PADIS、SAP IDoc、TRADACOMS などを含む）EDI フォーマットのファミリー • JSON ファイル • Microsoft Excel 2007 と以降のファイル • XBRL インスタンスファイルとタクソノミ • プロトコルバッファー • バイナリファイル（生 BLOB コンテンツ）

MapForce エディションにより、データの変換言語を以下のように選択することができます。

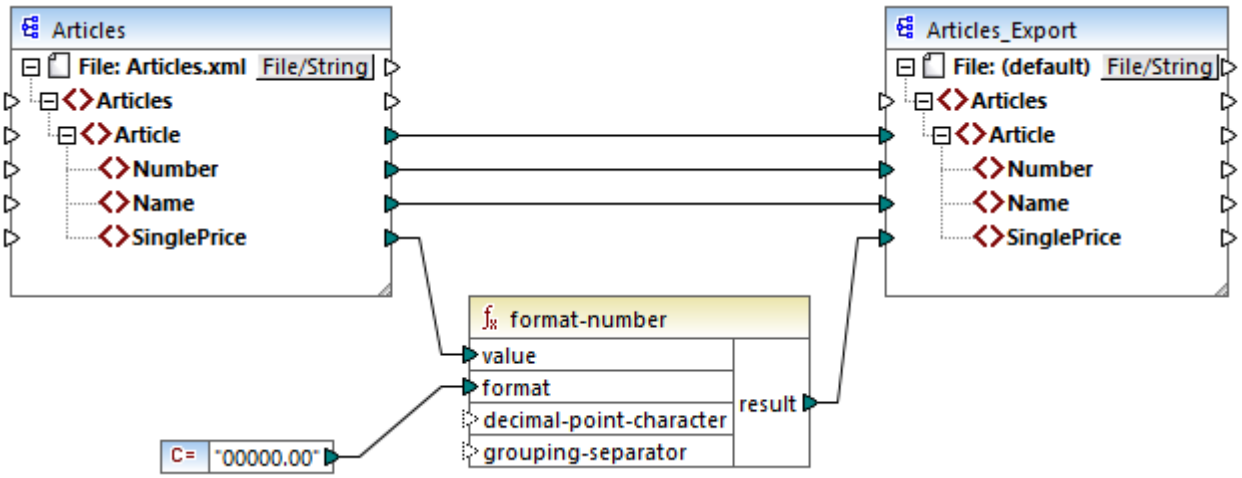
MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
• XSLT 1.0 • XSLT 2.0 • XSLT 3.0	• MapForce 内蔵の変換言語 • XSLT 1.0 • XSLT 2.0 • XSLT 3.0 • XQuery • Java • C# • C++	• MapForce 内蔵の変換言語 • XSLT 1.0 • XSLT 2.0 • XSLT 3.0 • XQuery • Java • C# • C++

全ての変換の結果、および生成された XSLT または XQuery コードをグラフィカルユーザーインターフェイスから移動することなくプレビューすることができます。デザインまたはマッピングをプレビューする際、MapForce はスキーマおよび変換の整合性を検証し、検証エラーを専用のウィンドウで表示するため、すぐに確認して、エラーの修正を行うことができます。

Java、C#、または C++ を変換言語として選択する場合、MapForce は必要とされるプロジェクトとソリューションを生成し、Visual Studio または Eclipse で直接開き、生成されたデータマッピング プログラムを実行することができます。高度なデータ統合のシナリオの場合、自身のコードを用い Altova と MapForce API を使用して、生成されたプログラムを拡張します。

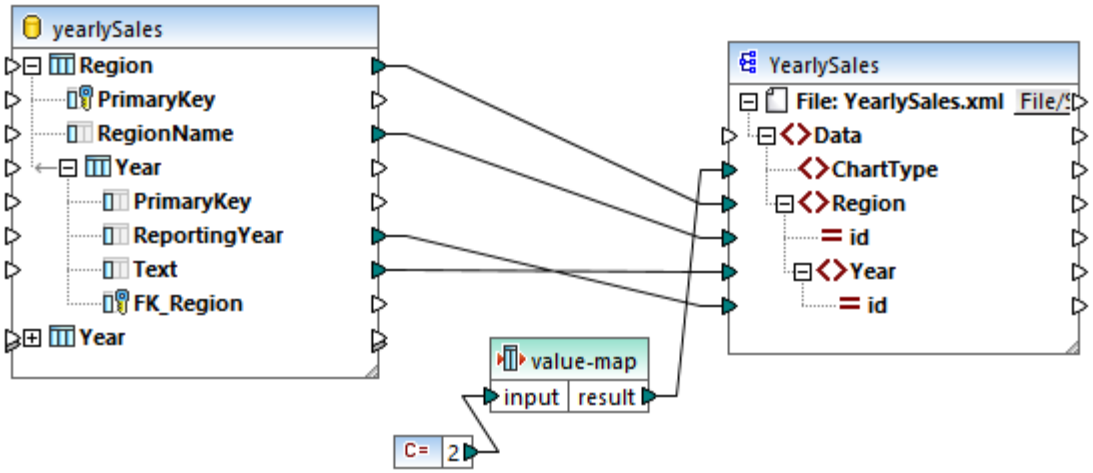
MapForce マッピングの変換言語を変更すると、その特定の言語のために特定の機能がサポートされていない場合があります。サポートメモを参照してください。

MapForce では、全てのマッピングの変換を視覚的にデザインすることができます。例えば、XML の場合、全ての要素と属性を接続することができます、または要素または属性 他の XML ファイルの要素または属性の XML ファイルにコメントを挿入します。これにより、MapForce にソース要素(または属性)からデータを読み込み、ターゲット 要素(または属性)に書き込むことを命令します。



2 つの XML ファイル間のサンプルデータ変換

同様に、MapForce Professional または Enterprise Editions でデータベースと作業する場合、MapForce のマッピングエリア内のデータベースカラムを確認し、データをマップし、視覚的な接続を作成することができます。Altova MissionKit 製品と同様、MapForce からデータベース接続を設定する場合、柔軟的にデータベースドライバーと接続の型 (ADO、ADO.NET、ODBC、または JDBC) を既存のインフラストラクチャとデータマッピングの必要に応じて選択することができます。更に、SQL クエリを視覚的に作成し、ストアドプロシージャを使用し、またはデータベースに直接問い合わせることができます。(データベース型、エディション、ドライバーによりサポートは異なります)。



XML ファイルとデータベース間のサンプルデータ変換

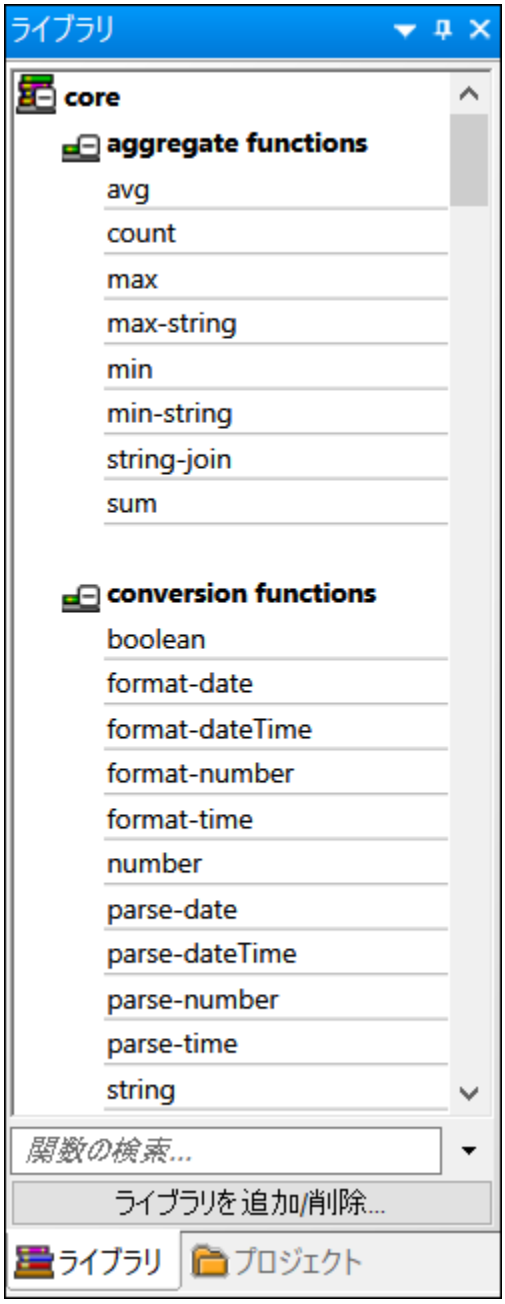
MapForce を使用したマッピングデザインのとても簡単な例として、[X からデータを読み込み Y に書き込む] が挙げられます。しかしながら、[X からデータを読み込み、Y に書き込む。そして Y からデータを読み込み Z へ書き込む] などの MapForce シナリオを簡単にデザインすることができます。これらは [パススルー] または [チェーン] マッピングとして知られ、これらにより変換プロセスの中間のステージでデータにアクセスすることができます（例えばファイルとして保存するため）。

MapForce 内で作成することのできるデータマッピングは、単一かつ定義済みファイルだけではありません。同じ変換内で、ディレクトリから複数の入力ファイルを動的に処理することができ、複数の出力ファイルを生成することができます。ですから、[複数の X ファイルからデータを読み込み、単一の Y ファイルに書き込む] または [ファイル X を読み込み複数のファイル Y を生成する] といったシナリオが可能になります。

重要な点は、同じ変換で、使用中の MapForce エディションでサポートされる全てのデータ型の複数のソースと複数のターゲットを混合できる点です。例えば、MapForce Professional または Enterprise の場合、2 つの異なるデータベースを単一の XML ファイルにマージすることが可能です。または、複数の XML ファイルからのデータをマージし、一部のデータを 1 つのデータベースとして書き込み、一部のデータを他のデータベースとして書き込むことができます。SQL ステートメントをデータベースにコミットする前にプレビューすることができます。

ソースからターゲットへのデータの直接の変換だけが達成する事柄ではありません。多くの場合、データを特定の方法で処理する必要がある場合があります（例: 並べ替え、グループ化、またはフィルター）などを最終プロセスの前に行う必要があります。この理由のため、MapForce は、プログラミング言語の構成を簡素化するその他の関数コンポーネント（規則、変数、SQL-WHERE 条件、フィルターと並べ替えコンポーネントなど）を搭載しています。他方、MapForce は、全てのデータ操作をアシストする豊富かつ広範囲の関数ライブラリも搭載しています。

必要であれば、ビルドインライブラリを、MapForce で直接関数（いわゆるユーザー定義関数、またはUDF）をデザインすることにより、またはXSLT、XQuery、Java、またはC# 言語で外部的に作成された関数やライブラリを使用して拡張することも可能です



ライブラリウィンドウ

データマッピングデザインファイルの数が多すぎる場合、マッピングをプロジェクトに整理することができます（MapForce Professional と Enterprise エディションで使用することができます）。これにより簡単なアクセスと管理を行うことができます。重要な点は、プロジェクト内の個別のマッピングのためにコードを生成し、更にプログラムコードをプロジェクト全体から生成することができます。

（MapForce Server API を使用したマッピング変換の実行時などの）高度なデータ処理のためにマッピングをデザインし、ランタイムで値をパス、またはランタイムから単純型文字列の値を取得することができます。この機能は、関数または単純型文字列の値を生成するマッピング全体を素早くテストすることを可能にします。MapForce のProfessional とEnterprise エディションは、他のプログラミング言語内で同様に動作する、ランタイムでの文字列の解析とシリアル化を行うコンポーネントも含みます。

MapForce Enterprise Edition では、W eb サービス言語定義（W SDL）ファイルをベースにしたSOAP 1.0 とSOAP 2.0 ウェブサービスを視覚的にデザインすることができます。W SDL-スタイル、またはREST-スタイルW eb サービスをマッピング内から呼び出すこともできます。

MapForce Professional とEnterprise Editions では、マッピングデザイン ファイルの詳細なドキュメントをHTML、およびRTF フォーマット で生成することができます。ドキュメンテーションデザインをカスタム化することができます（例: ドキュメントから特定のコンポーネントを包含、および、削除することが選択可能です）。

Altova MissionKit 製品と共にMapForce を使用している場合、MapForce はこれらの製品とAltova サーバーベース製品を以下のテーブルで示されている通り統合します。

MapForce Basic Edition	MapForce Professional Edition	MapForce Enterprise Edition
生成されたXSLT を直接 MapForce 内で実行し、データ変換の結果をすぐに確認することができます。パフォーマンスを向上するには、超高速 XML 変換エンジン RaptorXML Server を使用して、マッピングを処理します。		
XMLSpy がインストールされている場合、同じマシンで、関連するMapForce コンテキストからを直接 XMLSpy を開き、サポートされるファイル型を便利に開き編集することができます。（例: [コンポーネント | XMLSpy 内でスキーマ定義を編集] メニューコマンドは XML コンポーネントをクリックすると使用することができます）。		
	データ変換を、直接 MapForce 内で、またはコマンドラインや自動化された実行を使用して異なるマシンやオペレーティングシステムにデプロイし実行することができます。具体的な方法は、W indows 上でマッピングをデザインし、MapForce Server または FlowForce Server が動作するW indows、Linux、またはMac サーバーマシンで実行することができます。	
	StyleVision が同じマシンにインストールされている場合、デザインし、既存の StyleVision スタイルシートを再利用し、マッピングの結果をHTML、RTF、PDF、またはW ord 2007+ ドキュメントとしてプレビューすることができます。	

MapForce Professional とEnterprise エディションは、Visual Studio とEclipse の統合された開発環境のプラグインとしてインストールすることができます。これにより、優先する開発環境を移動することなく、MapForce の機能にアクセスしマッピングをデザインすることができます。

MapForce では、開発環境の概観や使用感を完全にカスタム化できるだけでなく（グラフィカルユーザーインターフェイス）、各技術と各マッピングのコンポーネントの型の他の関連する設定 もカスタム化することができます。例:

- XML から、またはXML へ、マッピングを行う際、スキーマレファレンスを含むか否かを選択することができ、またはXML 宣言 が XML 出力ファイル内で抑制されるべきかを選択することができます。生成されたファイルのエンコードを選択することができます（例: UTF-8）。
- データベースから、またはデータベースへマッピングする際、データベース ステートメントの実行のタイムアウト期間、MapForce がデータベーストランザクションを使用するか、またはコード生成の際データベーススキーマ名からテーブル名を削除するかなど、などの設定を定義することができます。
- XBRL の場合、MapForce が表示する構造ビューを選択することができます（「プレゼンテーション定義リンクベース」ビュー、[テーブルリンクベース] ビュー、または[全てのコンセプト] ビューなど）。

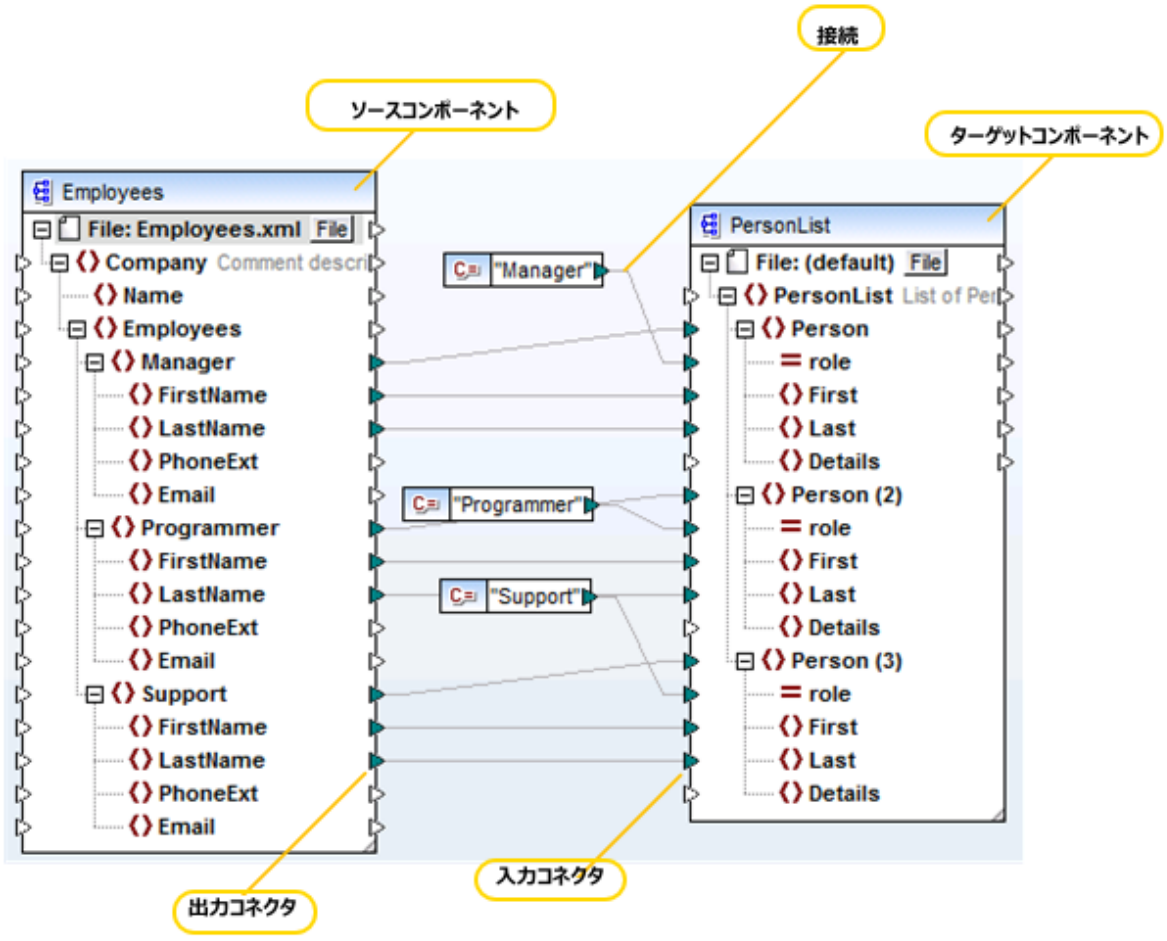
MapForce の全てのエディションは32-ビット アプリケーションで動作します。更に、MapForce Professional とEnterprise エディションは64-ビット アプリケーションで使用することができます。

1.4 基本概念

このセクションはデータマッピングを開始するにあたり必要な基本概念のアウトラインを紹介します。

マッピング

MapForce マッピングデザイン（または、略して「マッピング」）は、データがどのように1つのフォーマットから他のフォーマットに変換されるかの視覚的な表現です。マッピングは、データ変換を行うためのMapForce マッピングエリア内で1つのスキーマから他のスキーマに追加するコンポーネントから構成されています。（例: XMLドキュメントを1つのスキーマから他のスキーマに変換）。有効なマッピングは、1つまたは複数のターゲットコンポーネントに接続されている、1つまたは複数の複数のソースコンポーネントから構成されています。マッピングを実行して、結果を直接 MapForce 内で確認することができます。コードを生成し、外部で実行することも可能です。MapForce 実行可能ファイルにマッピングをコンパイルし、MapForce Server またはFlowForce Server を使用してマッピングの実行を自動化することもできます。
MapForce は、マッピングを.mfd の拡張子を持つファイルとして保存します。



MapForce マッピングの基本的構造

コンポーネント

MapForce 内では、「コンポーネント」という用語はデータの構造(スキーマ)またはデータがどのように変換(関数)されるかを視覚的に表します。コンポーネントはすべてのマッピングを構築する中心的な役割を果たします。マッピングエリアでは、コンポーネントは正方形の枠で表示されます。以下は、MapForce コンポーネントの例です:

- 定数
- フィルター
- 条件
- 関数コンポーネント
- EDI ドキュメント (UN/EDIFACT、ANSI X12、HL7)
- Excel 2007+ ファイル
- 単純型 入力コンポーネント
- 単純型 出力コンポーネント
- スキーマおよびDTD

コネクタ

コンポーネント の左右に表示される小さな三角形です。コンポーネントの左に表示されるコネクタはそのコンポーネントのエントリポイントを表します。コンポーネントの右に表示されるコネクタはそのコンポーネントからのデータの終了ポイントを表します。

接続

接続とは2つのコネクタ間に描くことができる線です。接続を描くことにより、MapForce にデータを特定の方法で変換するように命令します。(例: XML ドキュメントからデータを読み込み、他の XML ドキュメントに書き込みます)。

ソースコンポーネント

ソースコンポーネントとは、MapForce がデータを読み込む元のコンポーネントです。マッピングを実行すると、ソースコンポーネントのコネクタにより与えられたデータを読み込み、必要とされる型に変換し、ターゲットコンポーネントのコネクタに送信します。

ターゲットコンポーネント

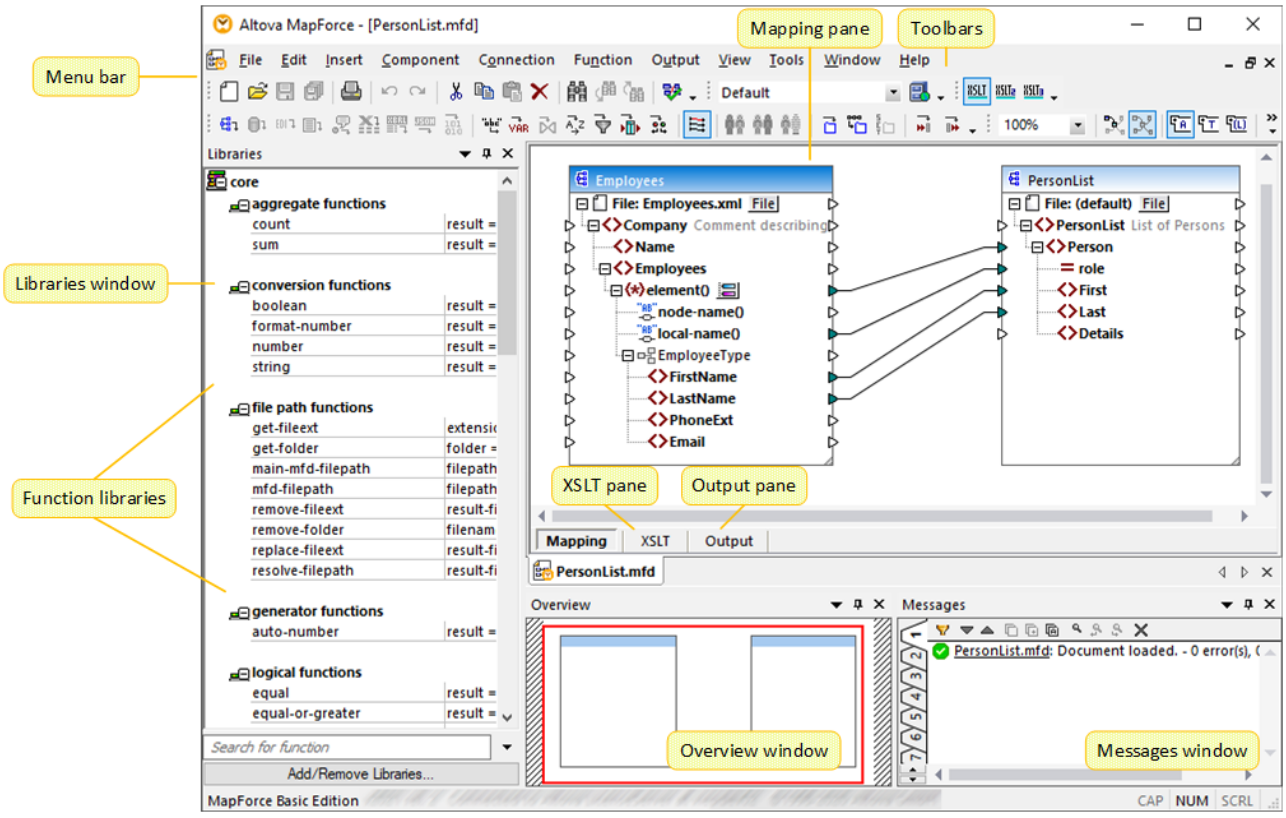
ターゲットコンポーネントとMapForce はがデータを書き込むコンポーネントです。マッピング を実行すると、ターゲットコンポーネントは、外部プログラム内で更に処理するためにMapForce にファイル(または複数のファイル)の生成、または結果を文字列の値として出力するように命令します。ターゲットコンポーネントは、ソースコンポーネントの逆です。

1.5 ユーザーインターフェイスの概要

MapForce のグラフィカルインターフェイスは、統合された開発環境として整理されています。メインのインターフェイス設定は「ツール | カスタマイズ」メニューコマンドを使用して行います。

各ウィンドウの右上に表示される ▼ 📌 ✕ ボタンを使用してウィンドウの、表示、非表示、ピン、ドックを行います。ツールバーとウィンドウをデフォルトの状態に復元する場合は、メニューコマンド「ツール | ツールバーとウィンドウの復元」を使用します。

下のイメージはMapForce グラフィカルなユーザーインターフェイス(MapForce Basic Edition)

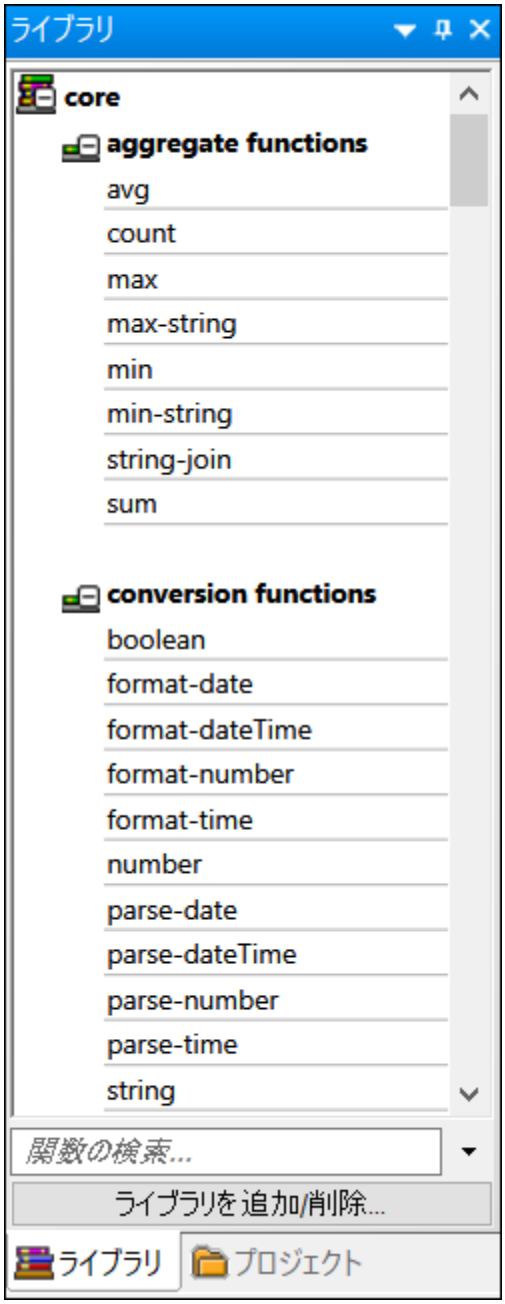


メニューバーとツールバー

メニューコマンドは、メニューアイテムを表示します。各ツールバーはMapForce コマンドのボタンのグループを表します。希望する場所にハンドルをドラッグすることにより、ツールバーを移動することができます。

ライブラリウィンドウ

ライブラリウィンドウは、ライブラリごとに整理されたMapForce 内蔵の関数をリストします。使用することのできる関数は「出力」メニューまたは言語選択ツールバーから選択される変換言語により異なります。変換言語の選択も参照してください。ユーザー定義関数を作成した場合、またはインポートした場合、これらの関数もライブラリウィンドウに表示されます。



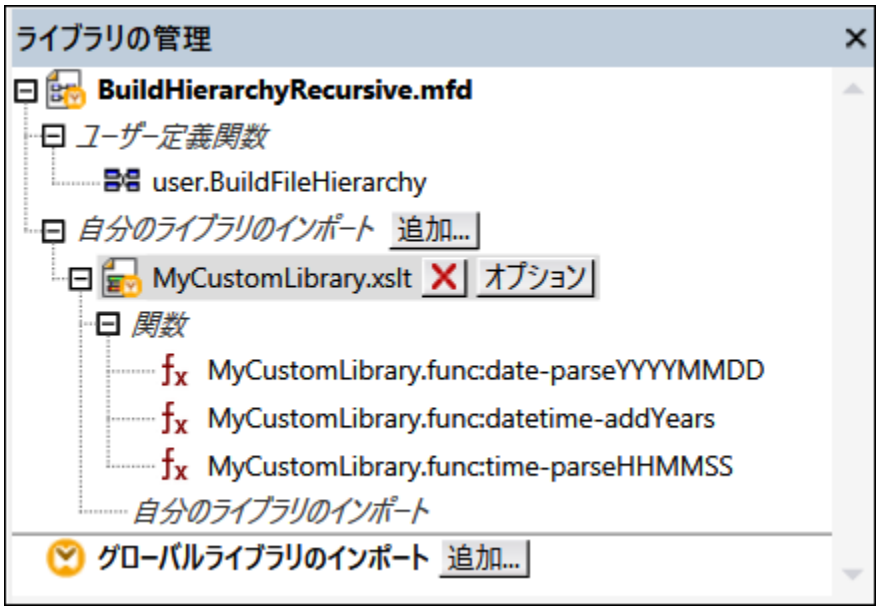
名前または詳細による関数の検索は、**ライブラリ**ウィンドウの下のテキストボックスに検索の値を入力することにより行うことができます。(現在アクティブなマッピング内で)関数の全ての発生を検索するには、関数を右クリックして、コンテキストメニューから**全ての呼び出し**を選択します。また、**ライブラリ**ウィンドウから直接関数データ型と詳細を確認することができます。詳細に関しては、関数と作業を参照してください。

ライブラリウィンドウの管理

このウィンドウから現在開かれているマッピングにより使用されているすべてのユーザー定義関数(UDF)とインポートされたカスタムライブラリを表示し管理することができます。

デフォルトではライブラリの管理ウィンドウは表示されていません。表示するには以下の一つを行うことができます:

- 「表示」メニューから「ライブラリの管理」をクリックします。
- ライブラリウィンドウのベースのライブラリの追加/削除 をクリックします。



UDF とライブラリを現在アクティブなマッピングのためにのみ、またはすべての開かれているマッピングのために表示することを選択できます。現在開かれているすべてのマッピングのためにインポートされた関数とライブラリを表示するには、ウィンドウ内を右クリックし、コンテキストメニューから「開かれているドキュメントの表示」選択します。

名前の代わりに開かれているマッピングのパスを表示するには、コンテキストメニューから「ファイルパスの表示」を選択します。

詳細に関しては関数ライブラリの管理を参照してください。

マッピング ペイン

マッピングペインはマッピングをデザインする作業エリアです。「挿入」メニュー(コンポーネントをマッピングに追加を参照)からマッピングエリアに(ファイル、スキーマ、コンスタント、変数などの)マッピングコンポーネント追加することができます。ライブラリウィンドウに表示されたマッピングペイン関数をドラッグすることもできます。マッピングに関数を追加を参照してください。

XSLT ペイン

XSLT ペインは、「XSLT」ボタンをクリックすると、マッピングから生成されたXSLT 変換コードを表示します。このペインを切り替えるにはXSLT、XSLT 2 またはXSLT3 を変換言語として選択し、同じ名前のタブをクリックします。

このペインは、ラインの付番とコードの折りたたみ機能を提供します。コードの一部を展開または折りたたむには、ウィンドウの左側の「+」と「-」アイコンをクリックします。折りたたまれたコードは、省略記号と共に表示されます。折りたたまれたコードを展開することなくプレビューするには、マウスで省略記号をポイントすると、プレビューされるコードを表示したヒントが下に表示されるように開かれます。プレビューされたテキストが大きくヒント内に表示できない場合、ヒントの後に追加表示記号が表示されます。

（インデント、行末マーカーを含む）表示設定を構成するには、ペインを右クリックして、コンテキストメニューから「**テキストビュー設定**」を選択します。または、「**テキストビュー設定**」（ ）ツールバーボタンをクリックします。

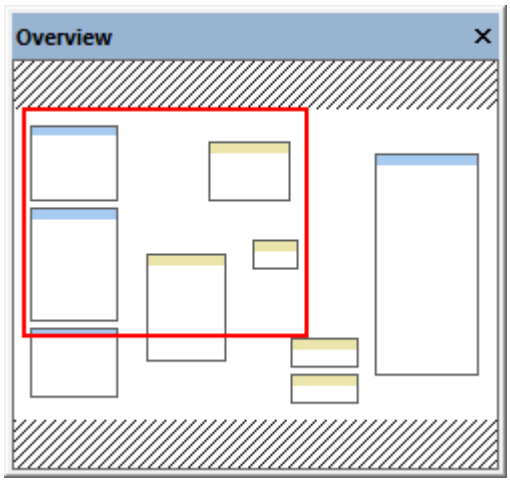
出力ペイン

出力ペインは、「**出力**」ボタンをクリックするとマッピングの変換結果を表示します（例: XML ファイル）。マッピングが複数のファイルを生成する場合、それぞれの生成されたファイルを順番にナビゲートすることができます。

XSLT ペインと同様の仕組みを持つラインの付番とコードの折りたたみ機能を提供します（上を参照）。

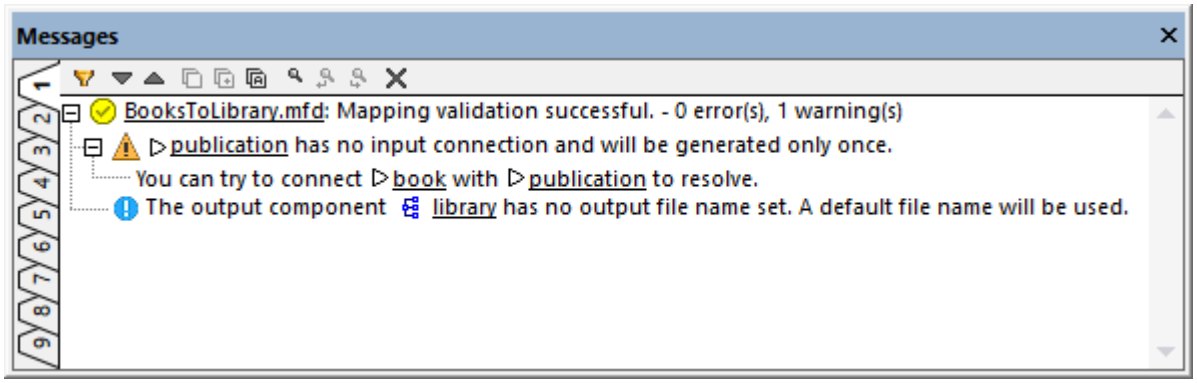
概要ウィンドウ

概要ウィンドウは、マッピングペインの概観図を表示します。マッピングのサイズが大きい場合、マッピングエリアの特定の場所に移動する場合に使用します。マッピングの特定の場所に移動する場合、赤い枠の長方形をクリックアンドドラッグします。



メッセージウィンドウ

メッセージウィンドウにはマッピングのプレビューまたは検証中の警告やエラーメッセージが表示されます。



情報、警告またはエラーメッセージをトリガーしたコンポーネントまたは構造のマッピングエリアをハイライトするには、メッセージウィンドウ内の下線のついたテキストをクリックします。

メッセージウィンドウ内のマッピング実行または検証オペレーションの結果は、以下のアイコンの1つと共に表示されます:

アイコン	説明
	オペレーションに成功しました。
	オペレーションは警告と共に終了しました。
	オペレーションに失敗しました。

メッセージウィンドウは次のメッセージを追加して表示する場合があります: 情報メッセージ、警告、およびエラー

アイコン	説明
	情報メッセージを表示します。情報メッセージはマッピングの実行を停止しません。
	警告メッセージを示します。警告メッセージを表示します。警告はマッピングの実行を停止しません。例えば、必須の入力コネクタへの接続が作成されなかった場合、警告メッセージが生成されます。このような場合、有効な接続を持つコンポーネントのために出力が生成されます。
	エラーメッセージを表示します。エラーが発生すると、マッピングの実行に失敗し、出力は生成されません。XSLT またはXQuery コードのプレビューの不可能です。

メッセージウィンドウ内の他のボタンにより以下のアクションを実行することができます:

アイコン	説明
	重要度(情報メッセージ、エラー、警告)別にメッセージをフィルターします。「**すべて選択**」を選択して、全ての重要度を選択します(これはデフォルトの振る舞いです)。 「**チェックをすべて解除**」を選択して、フィルターからすべての重要度を解除します。この場合、全般的な実行または検証の状況に関するメッセージのみが表示されます。

アイコン	説明
	次の行に移動させる。
	前の行に移動させる。
	選択された行をクリップボードにコピーする。
	ネストされたラインを含む選択されたラインをクリップボードにコピーします。
	メッセージウィンドウの全てのコンテンツをクリップボードにコピーします。
	メッセージウィンドウ内で特定のテキストを検索します。オプションで、特定の用語を検索する場合は、「**単語の完全マッチ**」を選択します。単語の大文字と小文字を区別する場合「**大文字と小文字を区別**」を選択します。
	特定のテキストを現在選択されているラインから最後まで検索します。
	特定のテキストを現在選択されているラインから最初まで検索します
	メッセージウィンドウをクリアする。

複数のマッピングファイルと同時に作業する場合、個々のマッピングのための個別のタブ内で情報、警告またはエラーを表示する必要があることがあります。この場合、マッピングを実行または検証する前に、メッセージウィンドウの左横にある番号の付いたタブをクリックします。

アプリケーションステータスバー

アプリケーションウィンドウの下部にあるアプリケーションステータスバーには、アプリケーションに関する情報が表示されます。ツールバーアイコンに対してマウスポインターを重ねると、ツールチップ情報がこのステータスバーに表示されます。アプリケーションステータスバーとデータベースクエリステータスバーは別のものです。64ビットバージョンのMapForce を使用している場合、アプリケーション名の後に(x64) という文字列が加えられます。32ビットバージョンでこのような表示は行われません。

1.6 規約

サンプルファイル

このドキュメントで参照される(.mfd 拡張子と他の伴うインスタンスファイルを持つファイル) データマッピングデザインファイルの多くは次のフォルダーで検索することができます:

- C:\Users\<username>\Documents\Altova\MapForce2021\MapForceExamples
- C:\Users\<username>\Documents\Altova\MapForce2021\MapForceExamples\Tutorials

MapForce に伴うサンプルマッピングとインスタンスファイルは動作の多くのアスペクトを紹介し、MapForce を使用するに当たり色々と試すことが奨励されます。オリジナルのサンプルへ直接変更を加えることを希望しない場合は、変更前にバックアップを作成することが奨励されます。

グラフィカルユーザーインターフェイス

このドキュメントに伴うイメージ(スクリーンショット) の一部は、使用中のMapForce エディションでは、使用することのできないグラフィカルユーザーインターフェイスの要素を含む場合があります。関連したコンテキストで、イメージは通常ソースマッピングデザイン(*.mfd) ファイルの名前、画像が作成されたMapForce のエディションを含みます。

2 チュートリアル

MapForce チュートリアルは、MapForce のデータの基本変換機能を短い時間で理解することをお手伝いします。チュートリアルの目的は全てのMapForce 機能を説明することではありませんが、MapForce の基本を順番に説明します。ですから、チュートリアルを順番どおりにフォローすることが奨励されます。コンセプトは順を追って複雑になるため、各コンセプトを次のコンセプトに移動する前に理解することが重要です。XML および XML スキーマに対する基本的な知識はとても有利です。

XML を新しいスキーマに変換

このチュートリアルは、コードを書く必要なく XSLT 2.0 言語を使用して XML 構造から他にデータを変換する方法について説明します。また、 MapForce シーケンスとアイテム、マッピング接続作成、関数の使用方法、マッピングの検証およびプレビュー、と出力結果をディスクに保存する方法などについて学ぶことができます。

複数のソースから1 つのターゲットにマップ

このチュートリアルは、異なるスキーマを持つ2 つのXML ファイルからのデータの読み込み方法や単一のターゲット XML ファイルへのマージの方法を説明します。また、各マッピングコンポーネントの名前とインスタンスファイルの変更方法や入力の複製の方法などについて学ぶことができます。

複数のターゲットスキーマとの作業

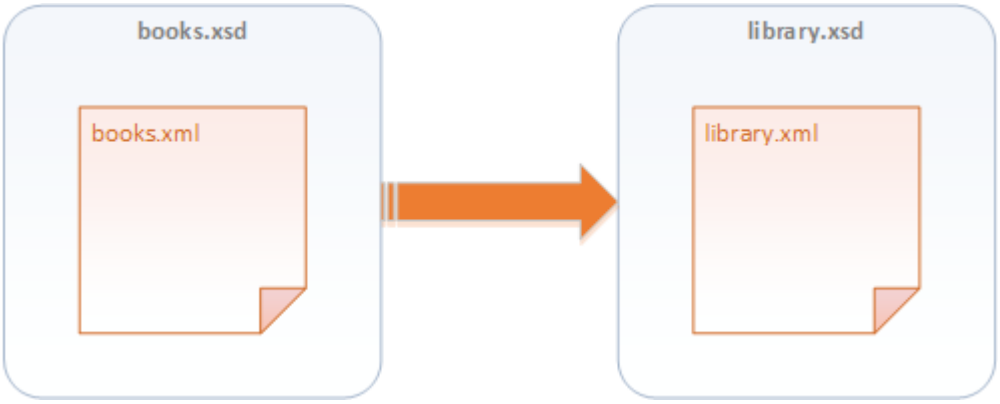
このチュートリアルは、2 つ以上のターゲット出力を作成する更に複雑なマッピングとの作業方法について説明します。具体的な方法は、同じマッピング内でブックレコードのリストを保管するファイル、最初のXML ファイル内のブックのサブセットのみを含む他のXML ファイルを生成し、発行年によりフィルターします。データのフィルタリングをサポートするには、「フィルター」コンポーネント、関数、数値定数を使用します。

ファイルを動的に処理と生成する

このチュートリアルは、同じフォルダーに存在する複数のXML インスタンスファイルからデータを読み込み複数のXML ファイルに書き込む方法を説明します。また、XML とスキーマ宣言 の削除および、関数を使用して、文字列を連結しファイル拡張子を抽出する方法についても学ぶことができます。

2.1 XML を新しいスキーマに変換

このチュートリアルは、MapForce 開発環境の基本を学びつつ、2 つの XML ファイル間のデータの変換方法について説明します。両方の XML ファイルはブックのリストを保管しますが、その要素は、異なる方法により名前を付けられ整理されます。(すなわち、2 つのファイルは異なるスキーマを持ちます)。



データ変換の抽象的なモデル

下にリストされるコードは、データソースとして使用されるファイルからのサンプルのデータを表示しています。簡略化するために、XML と名前空間宣言は省略されています。

```
<books>
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

books.xml

ターゲット (デスティネーション) ファイルではデータは以下のように表示されます:

```
<library>
   <last_updated>2015-06-02T16:26:55+02:00</last_updated>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
   <publication>
      <id>2</id>
```

```
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <genre>Fiction</genre>
      <publish_year>1912</publish_year>
   </publication>
</library>
```

library.xml

ソースとターゲット XML 内の一部の要素の名前は、同じではありません。このセクションの目的は、ソースファイル(<author>、<title>、<category>、<year>) 内の同等の要素から、ターゲットファイルの <author>、<title>、<genre> と <publish_year> 要素を表示することです。ソース XML ファイル内の属性 id は、ターゲット XML ファイル内の <id> 要素にマップされる必要があります。最後に、ターゲットファイルの <last_updated> 要素をファイルが最後に更新された日付と時刻として表示するように設定する必要があります。

必要とされるデータ変換を行うには、以下のステップを行います。

ステップ 1: XSLT2 を変換言語として選択する

これを以下の方法で行うことができます:

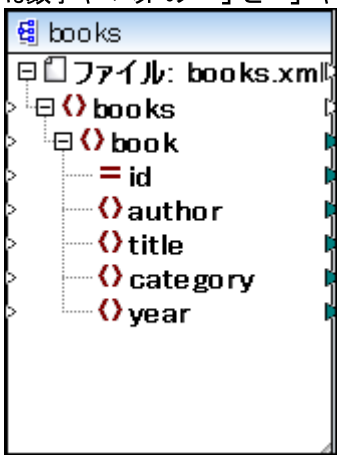
- 「XSLT2」  ツールバーボタンをクリックします。
- 「出力」 メニューから「XSLT 2.0」をクリックします。

ステップ 2: ソース XML ファイルをマッピングに追加する

このマッピングのためのソース XML ファイルは、は、以下のパスで見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial**books.xml。ファイルを以下の方法でマッピングに追加することができます:

- 「XML スキーマ / ファイルの挿入」  ツールバーボタンをクリックします。
- 「挿入」 メニューから「XML スキーマ/ファイル」をクリックします。
- XML ファイルをウィンドウエクスプローラーからマッピングエリアにドラッグします。

ファイルはマッピングエリアに追加され、構造を一見することができます。MapForce では、この構造はマッピングコンポーネントとして知られており、または単純にコンポーネント. コンポーネント内の要素を、折りたたむ(⊟) と展開するアイコン(⊞) をクリックすることにより、または数字キーパッドの「+」と「-」キーを押すことによりすることにより拡張することができます。



マッピングコンポーネント

マッピングペイン内でコンポーネントを移動するには、コンポーネントヘッダーをクリックして、マウスを新しい場所にドラッグします。コンポーネントのサイズを調整するには、コンポーネントの角をドラッグします。角をダブルクリックするとMapForce は自動的にサイズを調整します。

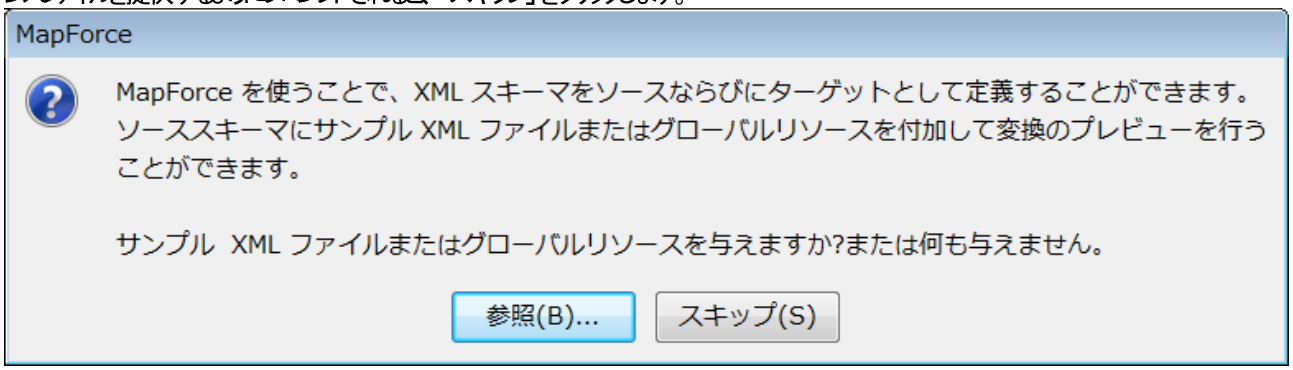
トップレベルノード は、ファイル名を表します。このサンプルの場合、は、XML インスタンスファイルの名前を表示します。構成内の、XML 要素は アイコンにより表示されており、XML 属性は アイコンにより表示されています。

コンポーネントの両側に表示されている小さな三角形は、(左側にある場合)データ入力を、(右側にある場合) データ出力を表します。MapForce では、それぞれ入力コネクタ、出力コネクタと呼ばれます。

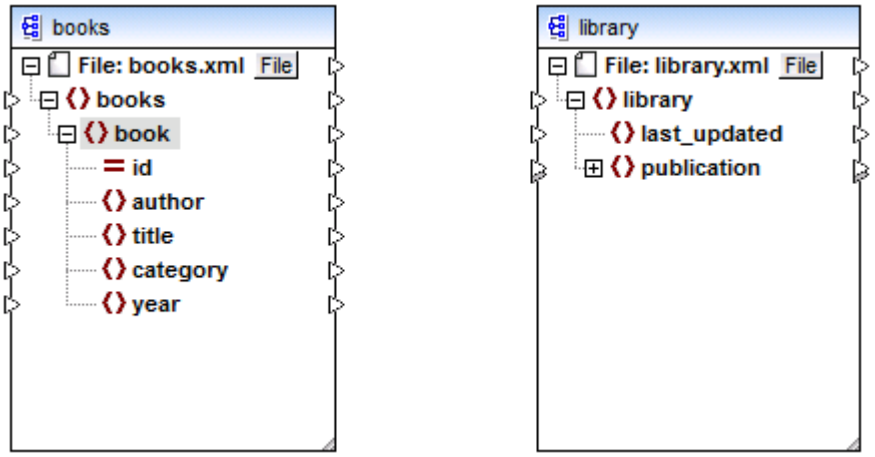
ステップ 3: ターゲット XML スキーマをマッピングに追加する

ターゲット XML を生成するには、既存のXML スキーマファイルを使用します。実生活のシナリオでは、このファイルは第三者パーティから提供される場合があり、またはXMLSpy などのツールを使用してユーザーが作成することができます。XML データのためのスキーマファイルが存在しない場合、MapForce は伴うスキーマまたはスキーマレファレンスを持たないXML ファイルをマッピングに追加すると、スキーマを生成するようにプロンプトします。

このサンプルでは、以下で見つけることのできる既存のスキーマファイルを使用します: **<マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial\library.xsd**。マッピングに追加するには、ソースXML ファイルと同様のステップを踏んでください。(「XML スキーマ / ファイルの挿入」( ) ツールバーボタンをクリックします)。MapForce にインスタンスファイルを提供するようにプロンプトされると、「スキップ」をクリックします。



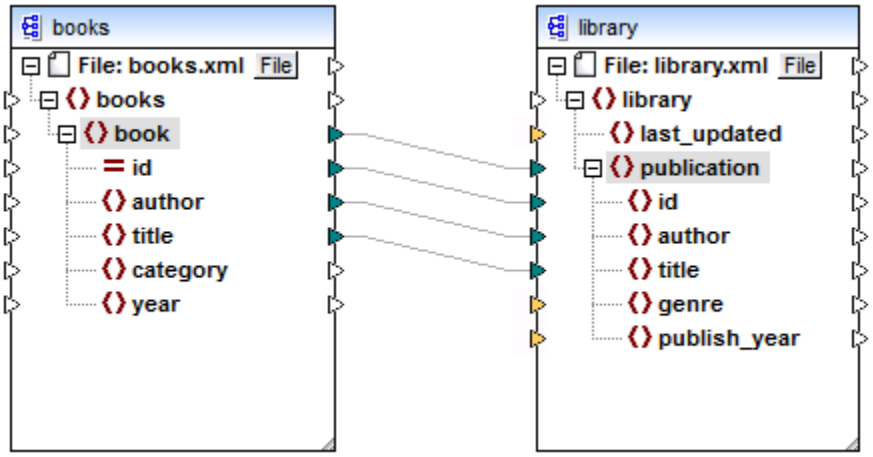
この時点では、マッピングデザインは、以下のようになります:



ステップ 4: 接続の作成

ソース XML ファイルのそれぞれの <book> 内で、ターゲット XML ファイル内に新しい <publication> を作成します。ですから、ソースコンポーネント内の <book> 要素とターゲットコンポーネント内の <publication> 要素を接続します。マッピング接続を作成するには、<book> 要素の右の出力コネクタ(小さな三角形)をクリックし、ターゲット内の <publication> 要素の入力コネクタにドラッグします。

これを行うと、MapForce は自動的に、ターゲットファイル内に同じ名前を持つソースファイル内の <book> の子の全ての要素を接続します。ですから、4 つの接続が同時に作成されます。この振る舞いは「子要素の自動接続」と呼ばれ、必要に応じて無効化または カスタム化することができます。



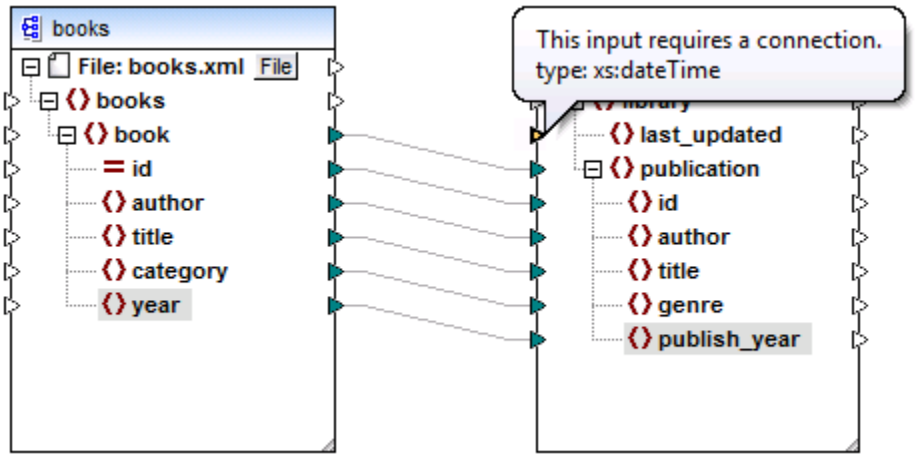
「マッチする子を自動的に接続する」の振る舞いを以下の方法で有効化または無効化することができます:

- 「子の自動接続の切り替え」( ) ツールバーボタンをクリックします。
- 「接続」メニューから「マッチする子を自動的に接続する」をクリックします。

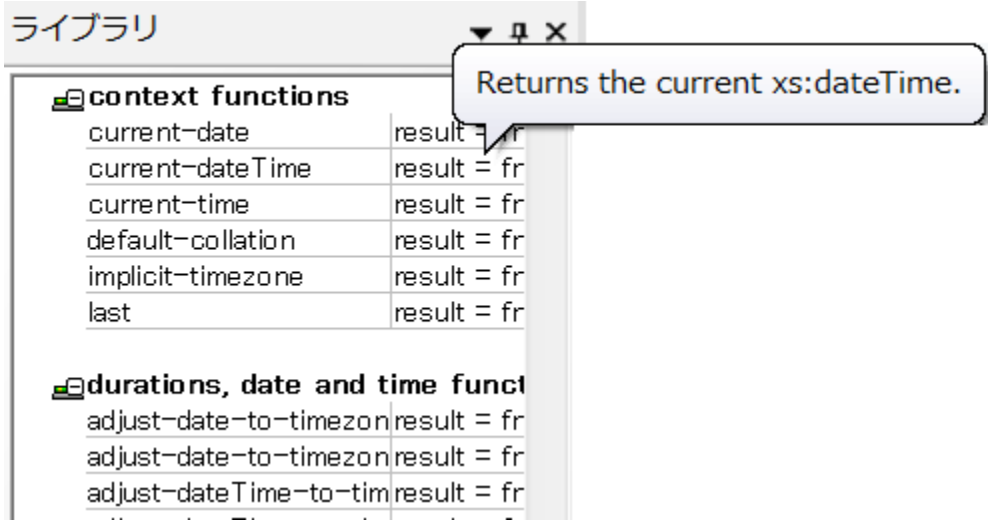
ターゲットコンポーネントの入力コネクタの一部が、MapForce により、必須アイテムを表示する、オレンジ色にハイライトされていることに注意してください。XML ファイルの妥当性を確認するために、必須アイテムの値を以下のように提供してください:

- ターゲット内の <genre> 要素とソース内の <category> 要素を接続する。
- ターゲット内の <publish_year> 要素とソース内の <year> 要素を接続する。

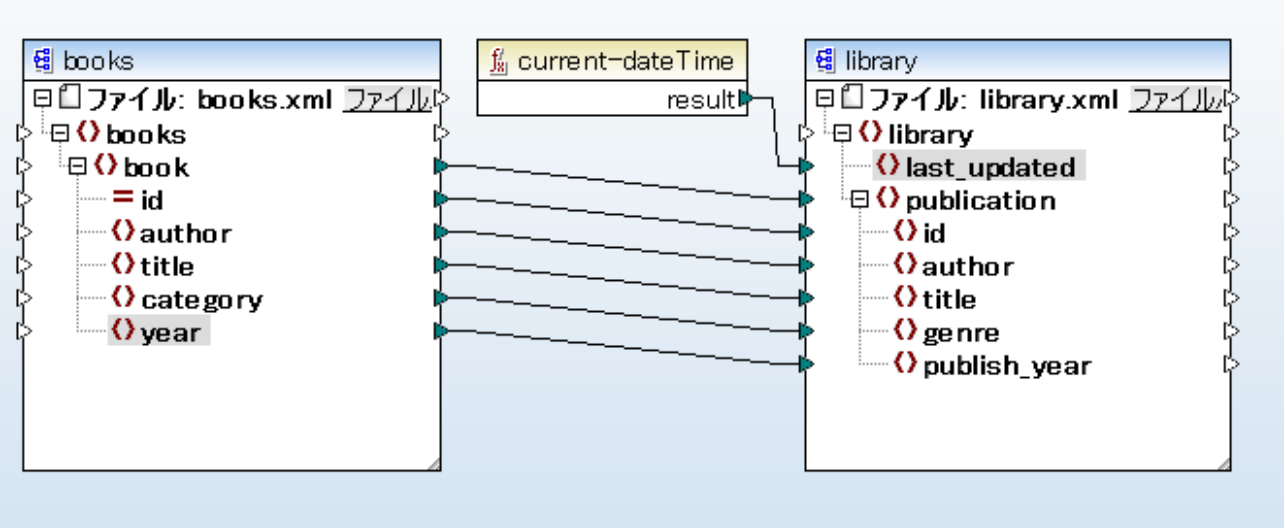
最後に、<last_updated> 要素への値を提供する必要があります。入力コネクタにマウスをポイントすると、要素は xs:dateTime 型であることが表示されます。ヒントを表示するには、「ヒントの表示」 ツールバーボタンが有効化されている必要があることに注意してください。



「**データ型の表示**」  ツールバーボタンをクリックすることにより、常に各アイテムのデータ型を表示することができます。
現在の日時（すなわち、`xs:dateTime` の値）を関 XSLT 数の日付と時刻で取得することができます。マッピングに対する XSLT 関数を検索するには、ライブラリウィンドウの下の部分にあるテキストボックスに「date」を入力します。または、マッピングの空のエリアをダブルクリックし、「current-date」を入力します。



上に表示されるように、マウスを関数の結果の部分にかざすと、詳細が表示されます。ヒントを表示するには、「**ヒントの表示**」  ツールバーボタンが有効化されていることを確認してください。
マッピングに関数を追加するには、関数をマッピングペインにドラッグし、`<last_updated>` 要素の出力を入力に接続します。



(**books.xsd** スキーマを持つ) books.xml インスタンスファイルのデータから(library.xsd スキーマを持つ) **library.xml** ファイル へデータを変換する MapForce マッピングデザイン(または「マッピング」) を作成することに成功しました。各コンポーネントのヘッダーをダブルクリックすると、コンポーネント設定ダイアログボックス内のこれらと他の設定を確認することができます。

ソースのためのコンポーネント設定

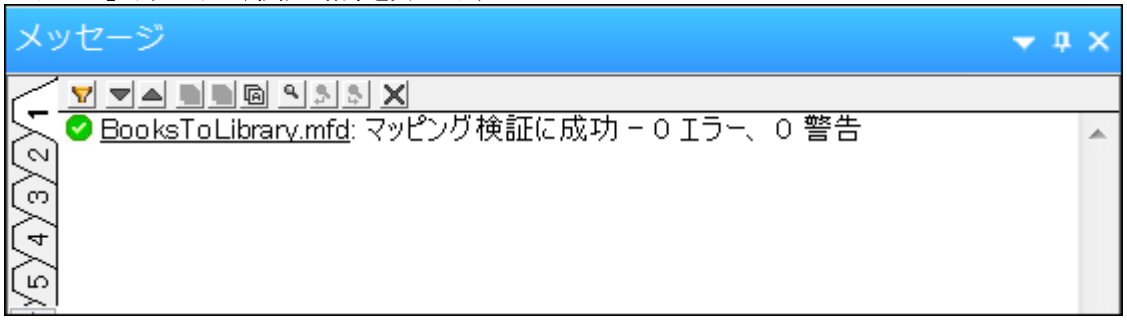
ターゲットのコンポーネント設定

ステップ 5: マッピングを検証し保存する

マッピングの検証は、マッピングを実行する前に、マッピングエラーの可能性を確認し修正するための任意のステップです。マッピングが妥当であるか確認するため、以下を行います:

- 「ファイル」メニューから「マッピングの検証」をクリックします。
- 「検証」  ツールバーボタンをクリックします。

「メッセージ」ウィンドウは、検証の結果を表示します:



メッセージウィンドウ

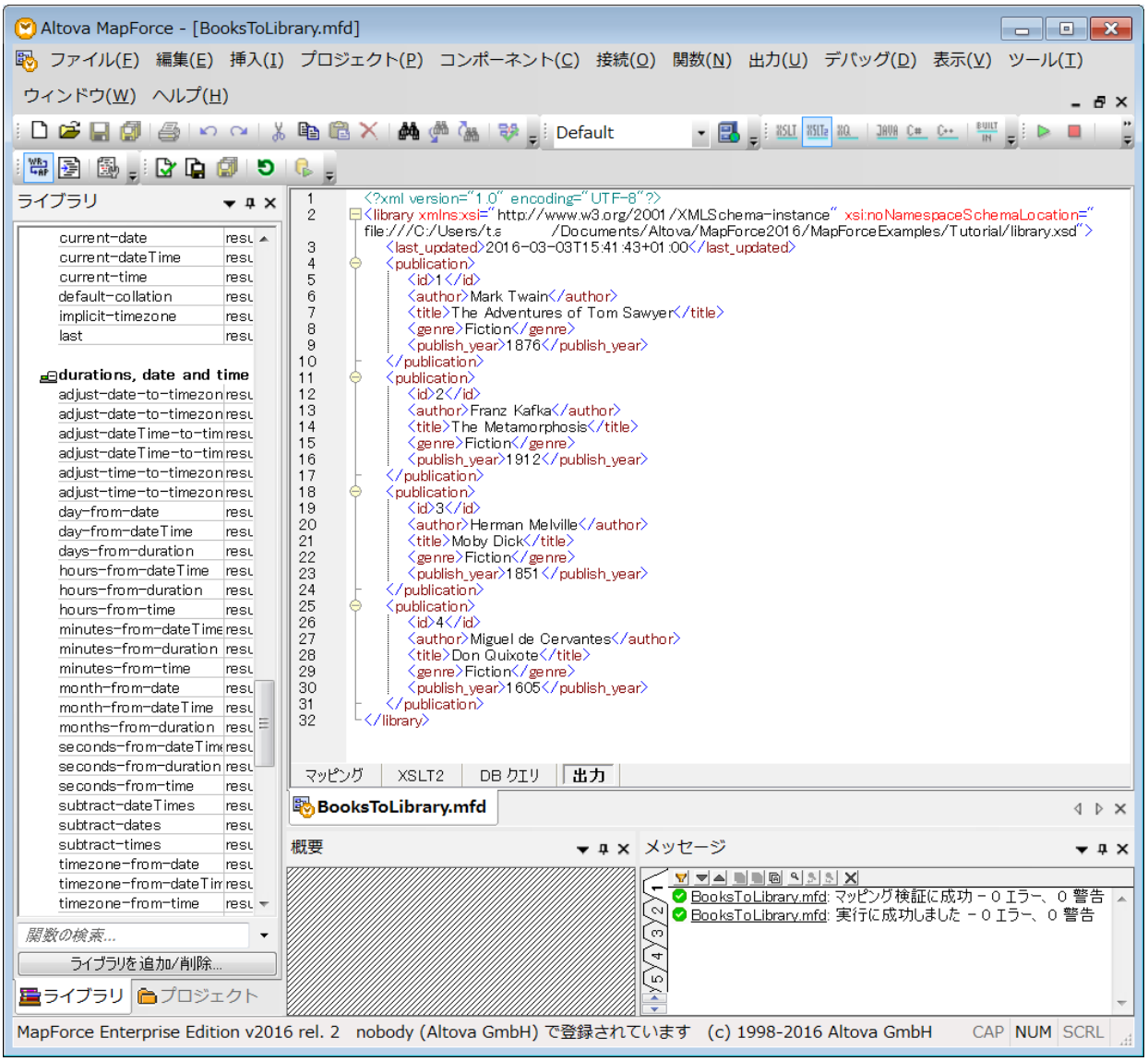
この時点で、ファイルにマッピングを保存します。マッピングを保存するには、以下を行います:

- 「ファイル」メニューから「保存」をクリックします。
- 「保存」  ツールバーボタンをクリックします。

このチュートリアルで作成されたマッピングは以下で検索することができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\\BooksToLibrary.mfd**。ですから、これから先、自分で作成したマッピングを継続して使用するか、または **BooksToLibrary.mfd** ファイルを使用することができます。

ステップ 6: マッピングの結果のプレビュー

マッピングの結果の結果を直接で MapForce でプレビューすることができます。これを行うには、マッピングペインの下にある、「出力」 ボタンをクリックします。MapForce は、変換を実行し、マッピングの結果を「出力」 ペインに表示します。



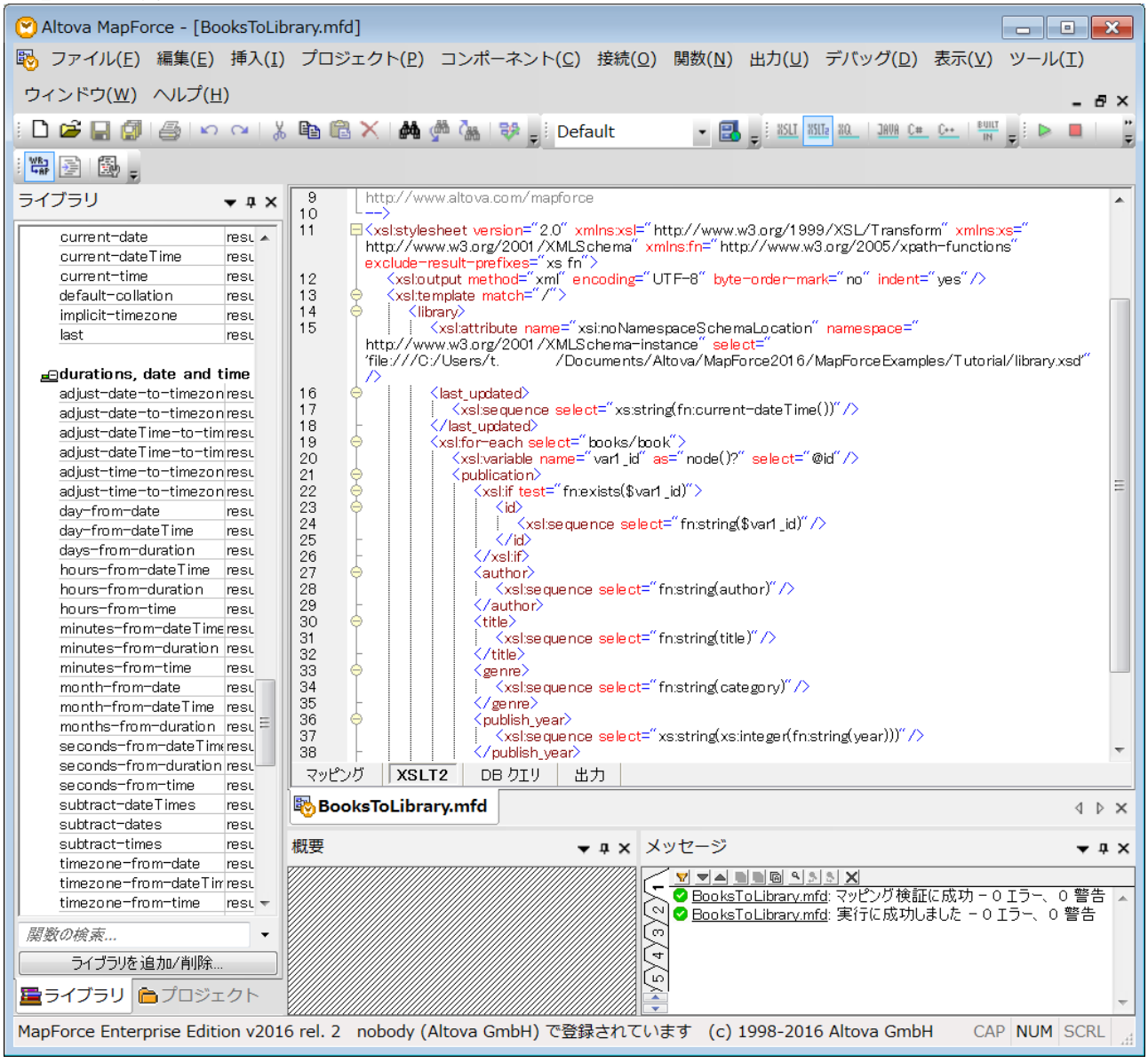
出力ペイン

MapForce 内で変換の結果を確認することができます。

デフォルトでは、「出力」 ペイン内にプレビューのために表示されたファイルはディスクに書き込まれません。その代わり、MapForce は、一時ファイルを作成します。「出力」 ペインに表示されたファイルをディスクに保存するには、メニューコマンド「出力 | 出力ファイルの保存」を選択するか、または「生成された出力を保存 」( ) ツールバーボタンをクリックします。

MapForce が一時ファイルではなく最終のファイルに直接的に出力を書き込むように構成するには、「ツール | オプション | 全般」に移動し、「最終出力ファイルにファイルを書き込む」チェックボックスを選択します。チュートリアルを実行中は、故意にではなくもとのチュートリアルファイルを上書きする可能性があるため、このオプションを有効化することは奨励されません。

変換を行う 生成された XSLT コードもプレビューすることができます。コードをプレビューするには、マッピングペインの下にある、「XSLT2」 ボタンをクリックします。



XSLT2 ペイン

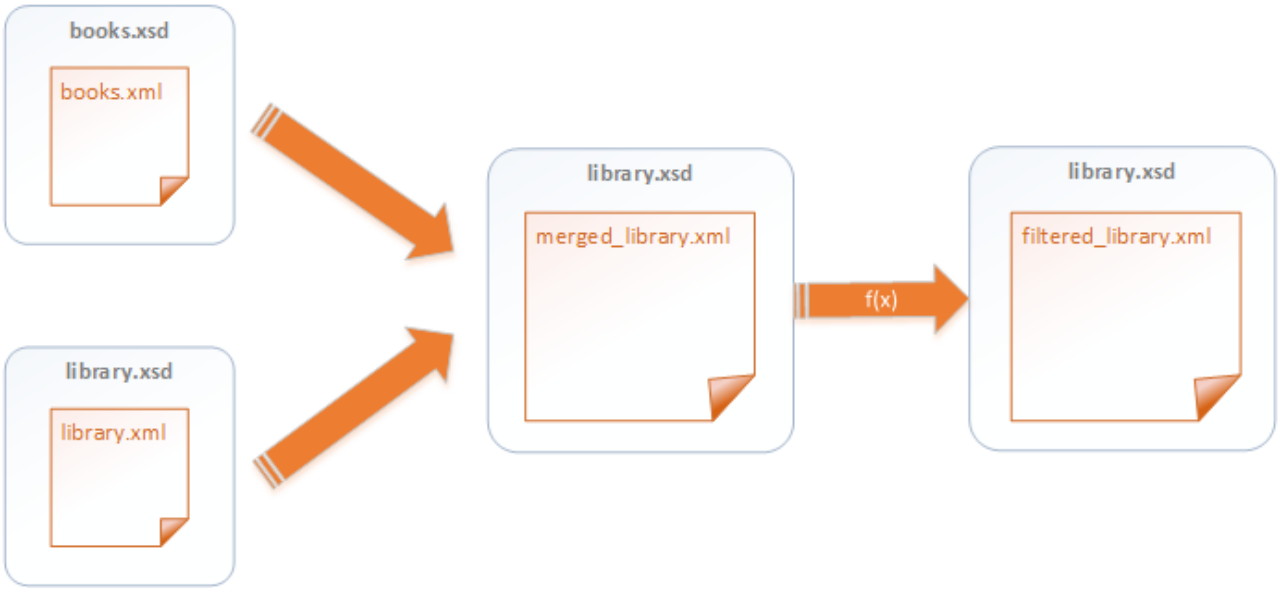
XSLT2 コードを生成しファイルに保存するには、メニューアイテム「**ファイル | コード生成 | XSLT 2.0**」を選択します。プロンプトされると、生成されたコードが保存されるフォルダーを選択してください。コード生成が完了すると、デスティネーションファイルは、以下の2つのファイルを含みます:

1. ターゲットスキーマにちなんで名づけられた XSLT 変換ファイル(このサンプルでは、**MappingMaptolibrary.xslt**)。
2. **DoTransform.bat** ファイル。**DoTransform.bat** ファイルにより RaptorXML Server 内での XSLT 変換の実行を有効化することができます。(詳細に関しては、http://www.altova.com/ja/raptorxml.html を参照してください)。

2.2　複数のソースから1つのターゲットにマップ

前のチュートリアルでは、ソースファイル（**books.xml**）からデータをターゲットファイル（**library.xml**）に変換しました。ターゲットファイル（**library.xml**）はマッピングの実行前は存在しておらず、マッピングの変換により生成されました。 **library.xml** ファイル内にすでにデータが存在すると仮定し、このデータを **books.xml** から変換されたデータとマージするとします。**library.xml** 空のデータを **books.xml** からのデータとマージすると仮定します。

このチュートリアルの目的は、**merged_library.xml** という名前のファイルを生成するマッピングをデザインすることです。生成されたファイルは、以下の2つのソースからのデータを含みます: **books.xml** と **library.xml** 。ソースとして使用される両方のファイルは異なるスキーマを持つことに注意してください。ファイルを動的に処理と生成するで説明されているとおり、ソースファイルと同じスキーマを持つ場合、異なるアプローチを使用してデータをマージすることができます



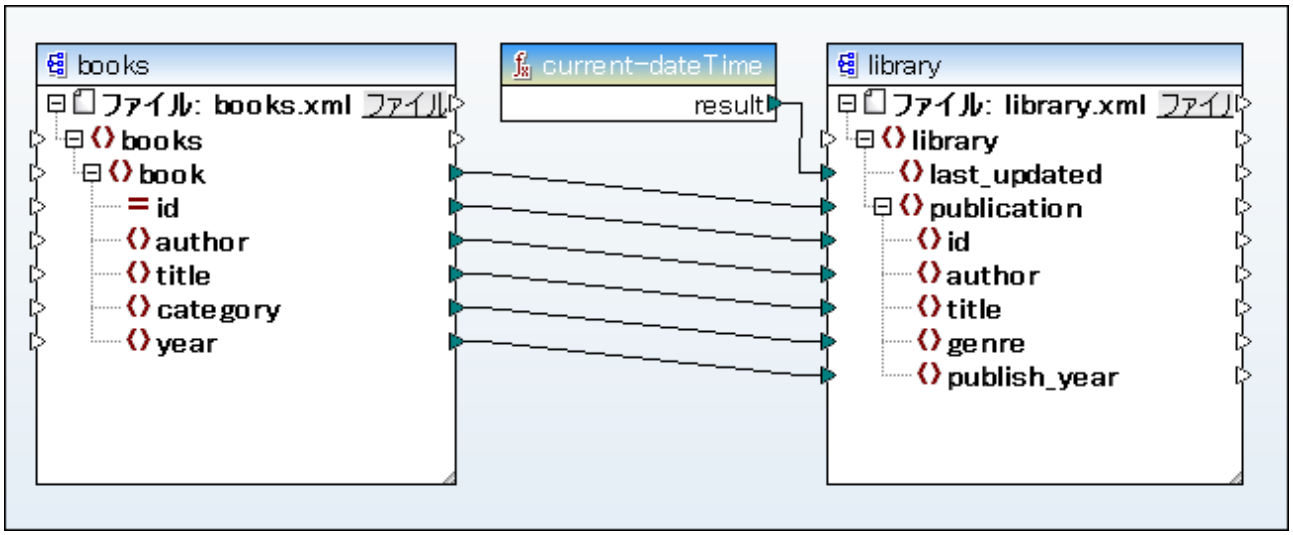
データ変換の抽象的なモデル

チュートリアルの目的を達成するためには、次のステップを踏みます。

ステップ 1: マッピングデザインファイルの準備

このチュートリアルは、＜**マイドキュメント**＞¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダーからの **BooksToLibrary.mfd** マッピングを開始点として使用します。このマッピングはXML を新しいスキーマに変換するチュートリアル内で作成済みです。チュートリアルを開始するには、MapForce で **BooksToLibrary.mfd** ファイルを開き、新しい名前で保存します。

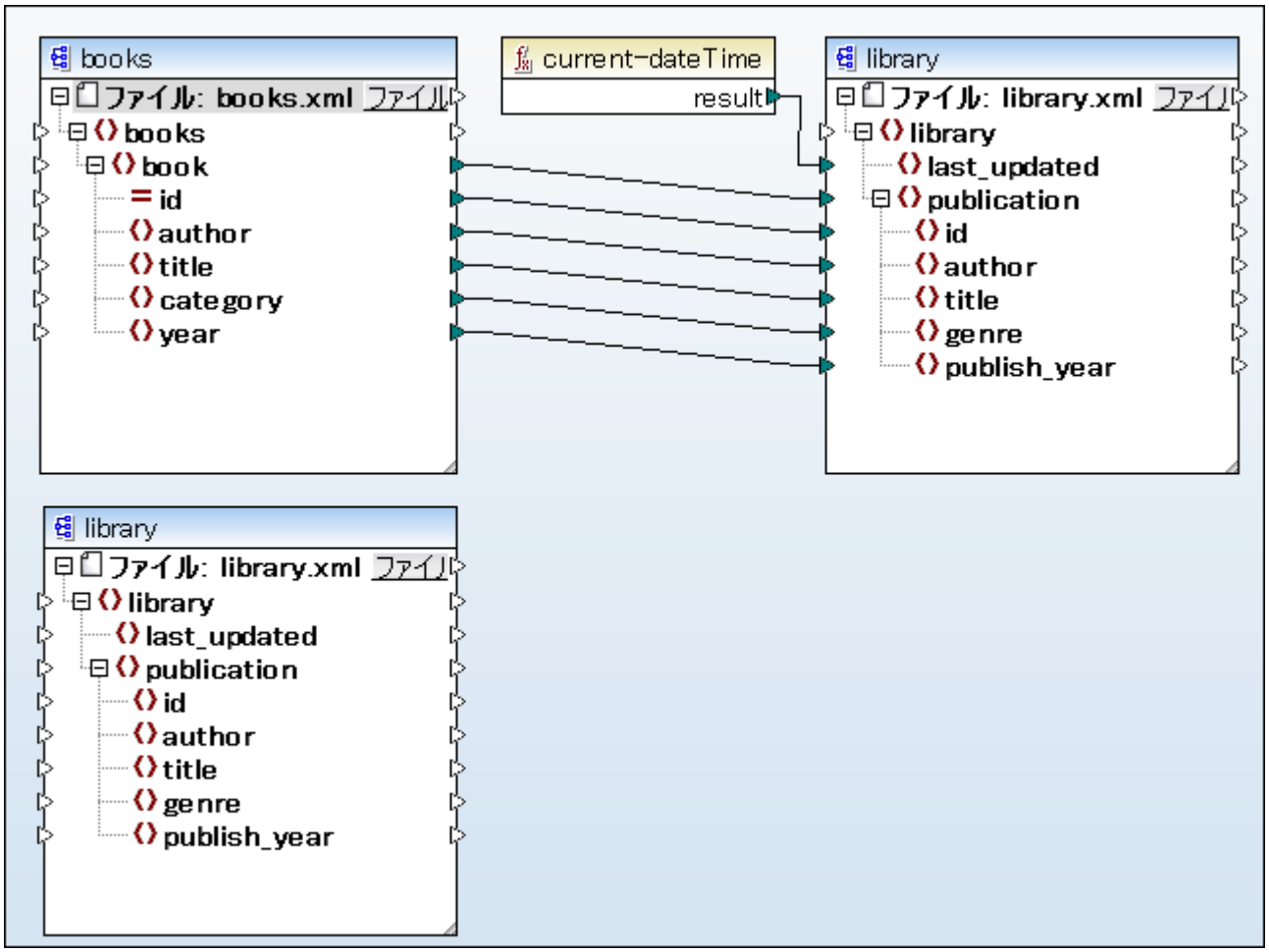
> 複数のファイルを参照するため、＜**マイドキュメント**＞¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダー内に新しいマッピングが保存されることを確認してください。



BooksToLibrary.mfd (MapForce Basic Edition)

ステップ 2: 2番目のソースコンポーネントの作成

最初に、ターゲットコンポーネントを選択して、(「Ctrl + C」を押してコピーし、同じマッピング内に「Ctrl + V」を押して貼り付けます。新しいコンポーネントのヘッダーを **books** コンポーネントの下にドラッグします。



マッピングには2つのソースコンポーネントと **books** と**library**、と1つのターゲットコンポーネントがあります: library。

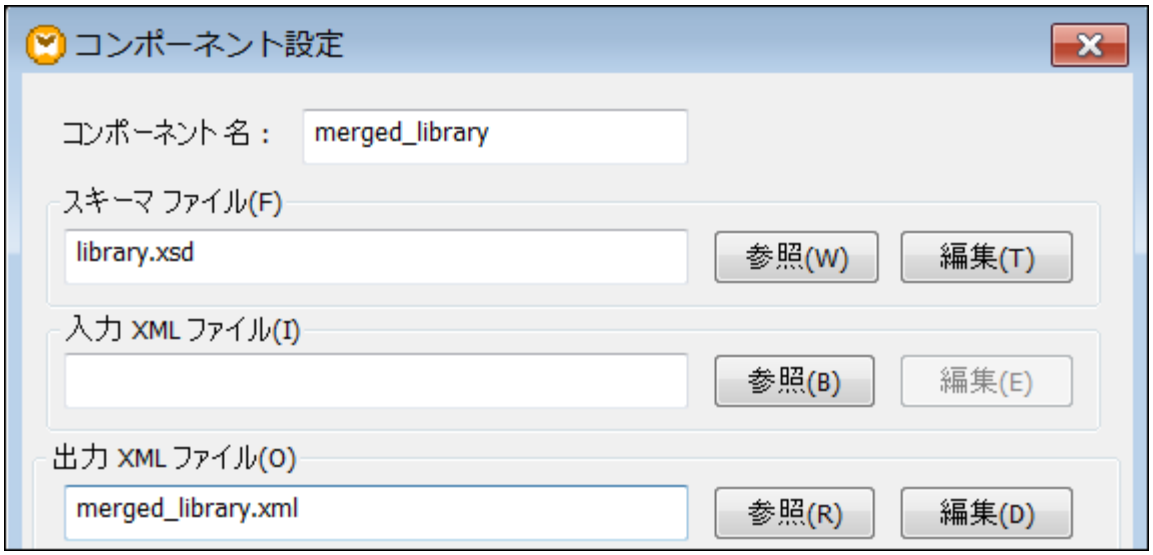
マッピングコンポーネントを4つの(左、右、上、下)方向に移動することができます。しかしながら、ソースコンポーネントをターゲットコンポーネントの左に配置することにより、マッピングの作成を簡単にし、また、他のユーザーにより読み込みやすく理解しやすくなります。このドキュメントで説明されている全てのマッピングにもこの規則は適応されており、また、MapForce インストールに伴うサンプルマッピングファイルにも同様の規則が適用されています。

ステップ 3: 入力/出力ファイルの検証とセット

前のステップでは、新しいソースコンポーネントはターゲットコンポーネントからコピーし、貼り付けられましたので同じ設定を継承します。名前入力/出力インスタンスファイルが正確に設定されるように、各コンポーネントのヘッダーをダブルクリックし、コンポーネント設定ダイアログボックス内で、下に表示されるように各コンポーネントの入力/出力ファイルを検証し、名前を変更します。

*最初のソースのコンポーネント設定 (**books**)*

*2番目のソースのコンポーネント設定 (**library**)*

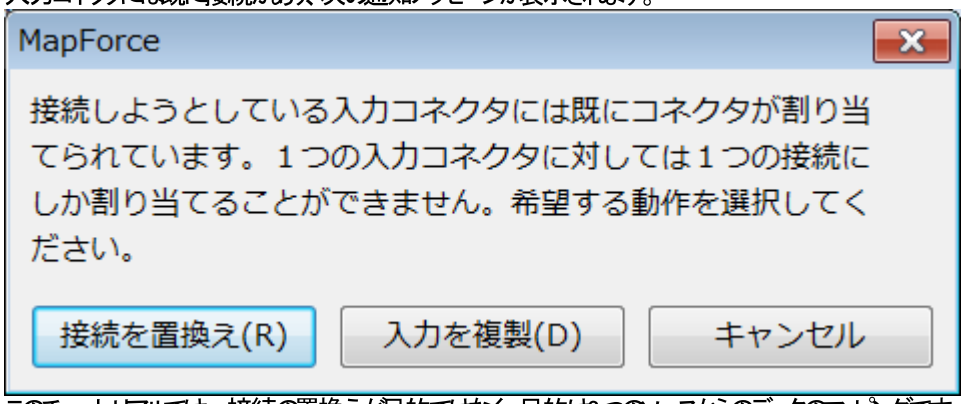


*ターゲットのコンポーネント設定 (**merged_library**)*

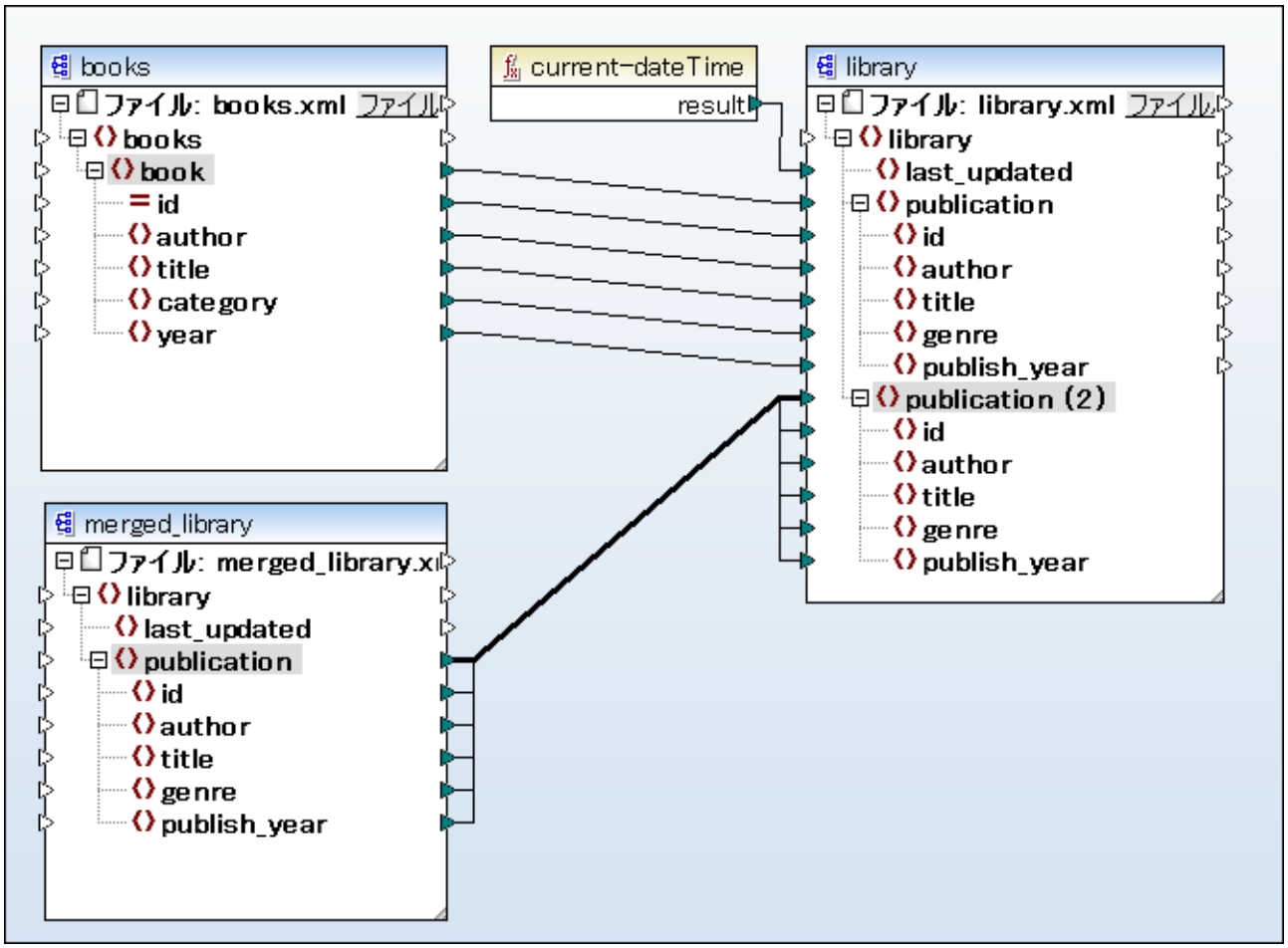
上に表示されるように、最初のソースコンポーネントは**books.xml** からデータを読み込みます。2番目のソースコンポーネントは **library.xml** からデータを読み込みます。2番目のソースコンポーネントは**merged_library.xml** というファイルのターゲットコンポーネント出力データを最後に読み込みます。

ステップ 4: 接続の作成

MapForce に2番目のソースからターゲットにデータを書き込むように命令するには、ソース **library** 内コンポーネントの `publications` アイテムの出力コネクタ（小さな三角形）をクリックし、ターゲット **library** コンポーネント内のアイテムの入力コネクタにドラッグします。ターゲット入力コネクタには既に接続があり、次の通知メッセージが表示されます。



このチュートリアルでは、接続の置換えが目的ではなく、目的は2つのソースからのデータのマッピングです。ですから、「**入力の複製**」をクリックします。これを行うことにより、ターゲットコンポーネントに新しいソースからのデータを受け入れるように構成することができます。マッピングは、以下のようになります:



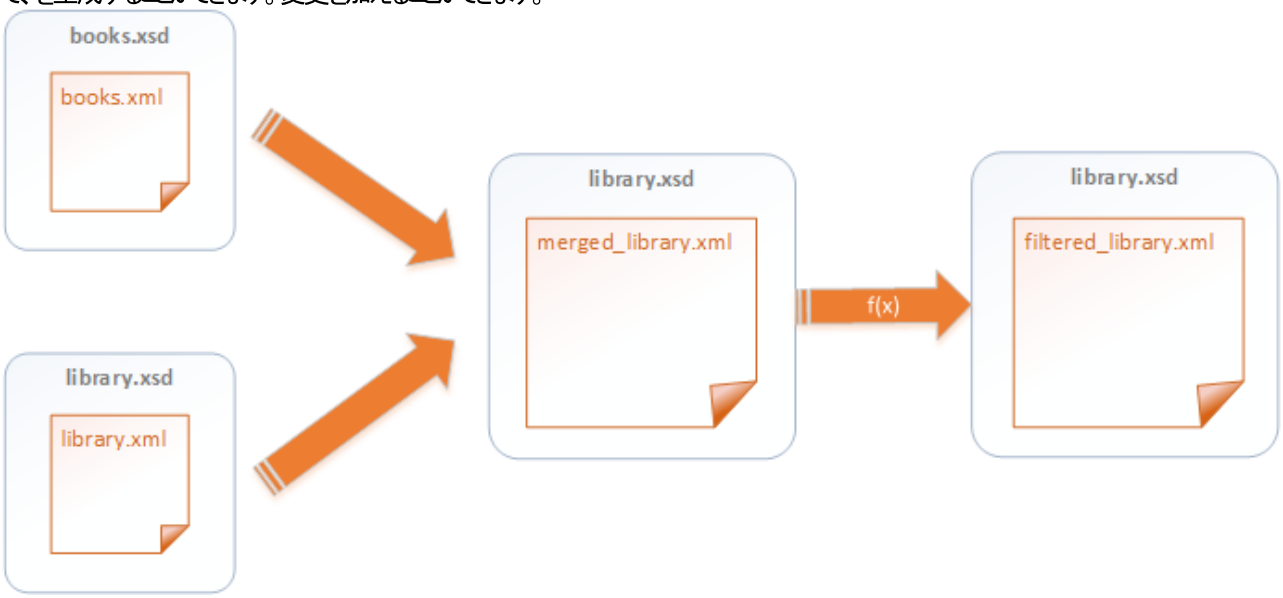
ターゲットコンポーネント内の`publication`アイテムが複製されました。新規`publication(2)`ノードはソース**library**コンポーネントからのデータを受け入れます。更に重要な点は、マッピング内でこのノードは`publication(2)`と表示されますが、結果XMLファイル内での名前は、想定された目的である`publication`となります。

マッピングペインの下の「出力」ボタンをクリックして、マッピングの結果を確認します。**library.xml**と**books.xml**ファイルのデータが、新規ファイル**merged_library.xml**にマージされたことを確認してください。

2.3 複数のターゲットスキーマとの作業

前のチュートリアルの、複数のソースから1つのターゲットにマップ では、複数のソーススキーマから単一のターゲットスキーマへのデータのマッピングについて作成方法について説明しました。2 つのソースからのブックのレコードを保管する **merged_library.xml** と呼ばれるファイルを作成しました。他の部署からこのXML ファイルのサブセットを提供するように要請があったと仮定します。特に、1900 年以降に出版された書籍を含むXML ファイルを提出するとします。

既存の**MultipleSourcesToOneTarget.mfd** マッピングを使用し、完全な XML ライブラリとフィルターされたライブラリ必要に応じて、を生成することができます。変更を加えることができます。



データ変換の抽象的なモデル

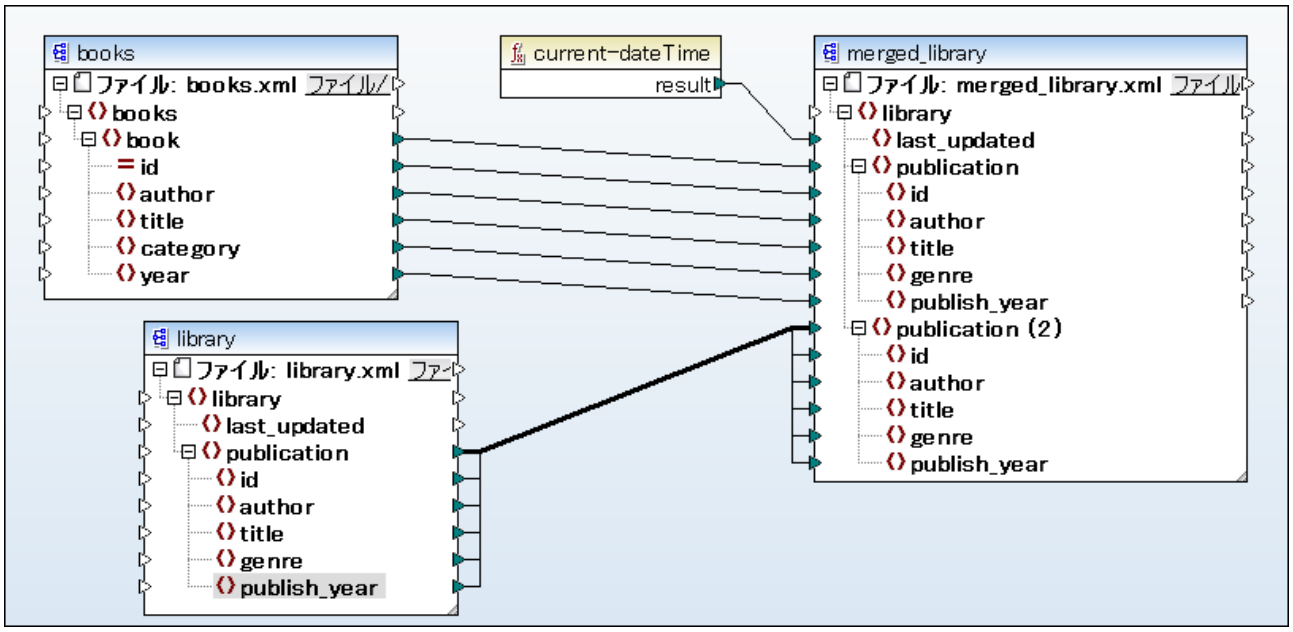
上の図では、データは2 つの異なるスキーマ（**books.xsd** と**library.xsd**）が**merged_library.xml** と呼ばれる単一のXML ファイルにマージされます。第2に、データはフィルター関数を使用して変換され、**filtered_library.xml** と呼ばれるXML ファイルを作成する、次のコンポーネントに渡されます。「中間」コンポーネントは、データターゲットとソースとしての役割を果たします。MapForce では、このテクニックは「チェーンマッピング」と呼ばれ、このチュートリアルの主な内容です。ます。

このチュートリアルのゴールは、**merged_library.xml** と**filtered_library.xml** を必要に応じて生成することです。チュートリアルの目的を達成するためには、次のステップを踏みます。

ステップ 1: マッピングデザインファイルの準備

このチュートリアルは、＜マイドキュメント＞¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダーからの**MultipleSourcesToOneTarget.mfd** マッピングを開始点として使用します。このマッピングは複数のソースから1つのターゲットにマップ チュートリアル内で作成済みです。チュートリアルを開始するには、MapForce で**MultipleSourcesToOneTarget.mfd** ファイルを開き、新しい名前で保存します。

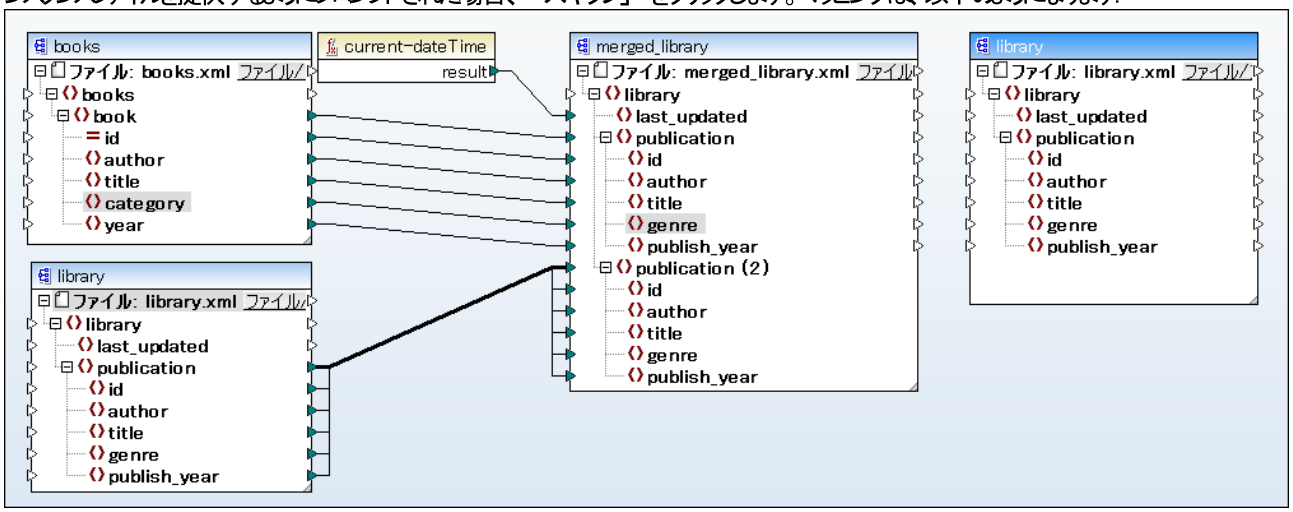
> 複数のファイルを参照するため、＜マイドキュメント＞¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダー内に新しいマッピングが保存されることを確認してください。



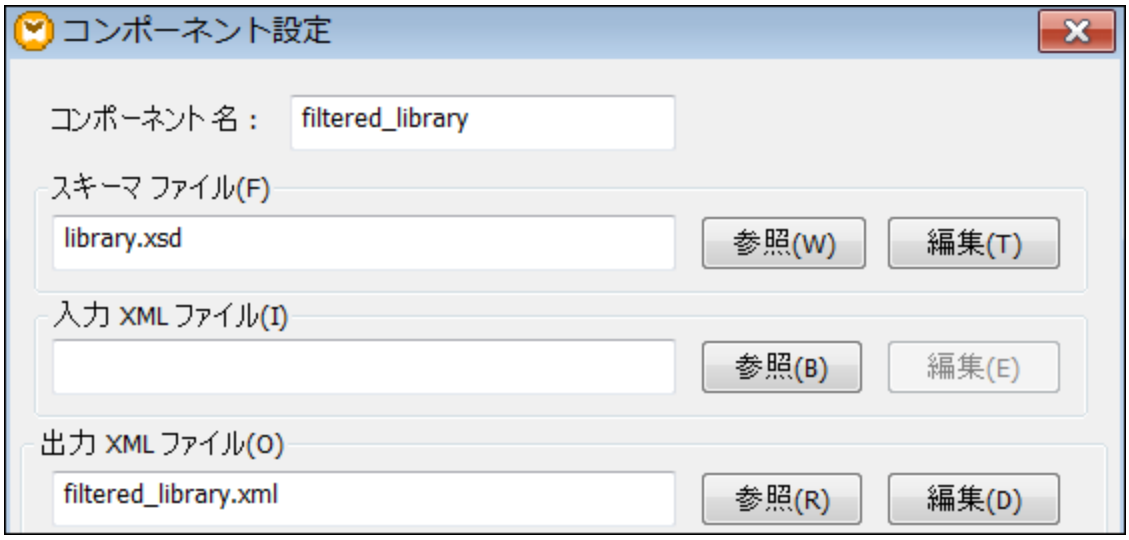
MultipleSourcesToOneTarget.mfd (MapForce Basic Edition)

ステップ 2: 2番目のターゲットコンポーネントの追加と構成

2番目のターゲットコンポーネントを追加するには、「XML スキーマ / ファイルの挿入」( ) ツールバーボタンをクリックし、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダーにある **library.xsd** ファイルを開きます。サンプルインスタンスファイルを提供するようにプロンプトされた場合、「スキップ」 をクリックします。マッピングは、以下のようになります:



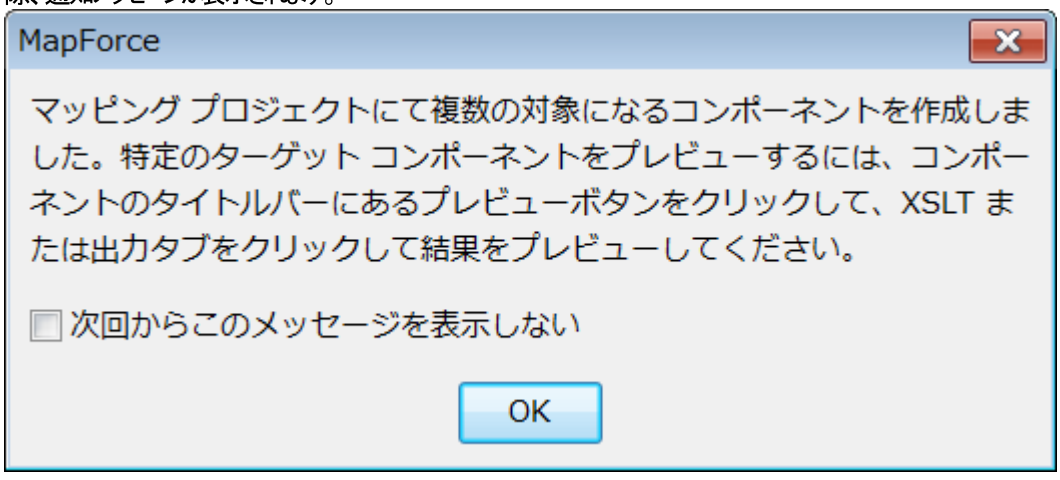
上に表示されるように、マッピングには2つのソースコンポーネントと **books** と**library**、と2つのターゲットコンポーネントがあります。ターゲットコンポーネントを区別するために、2番目のターゲットコンポーネントを**filtered_library** に名前を変更し、生成される XML ファイルの名前も変更します。これを行うには、右端にあるコンポーネントをダブルクリックして、コンポーネントの設定を次のように編集します:



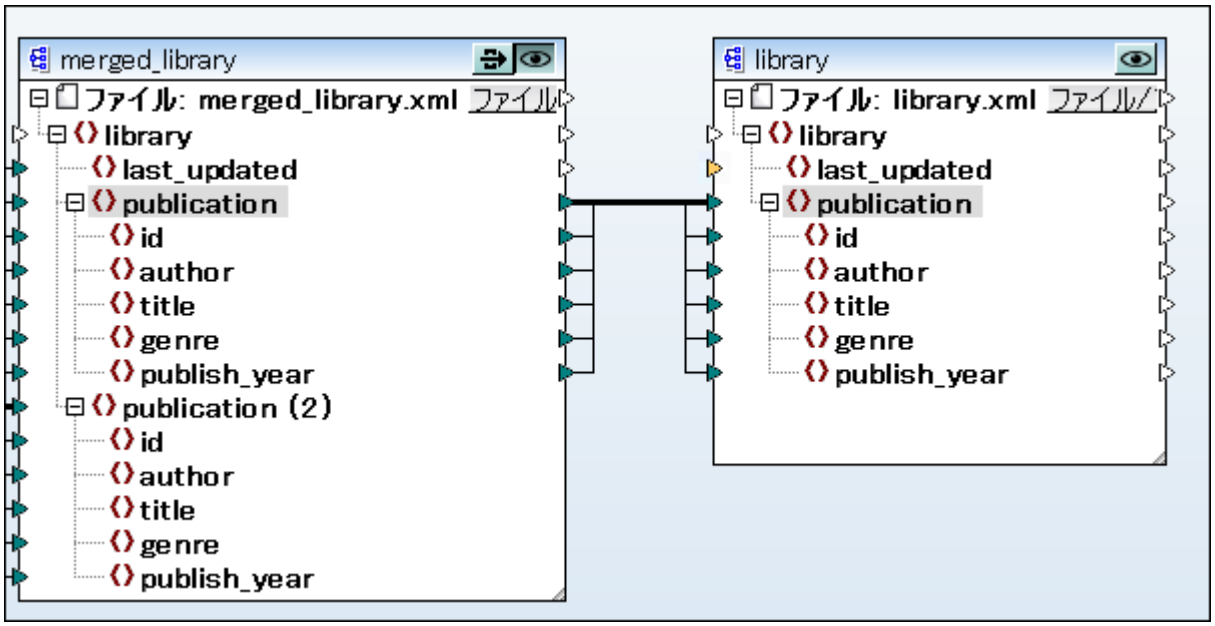
コンポーネントの新しい名前は**filtered_librar** となり、出力 XML ファイルには**filtered_library.xml** という名前が付けられます。

ステップ 3: 接続の作成

merged_library 内のアイテム**publication** から**filtered_library** 内のアイテム**publication** への接続を作成します。これを行う際、通知メッセージが表示されます。

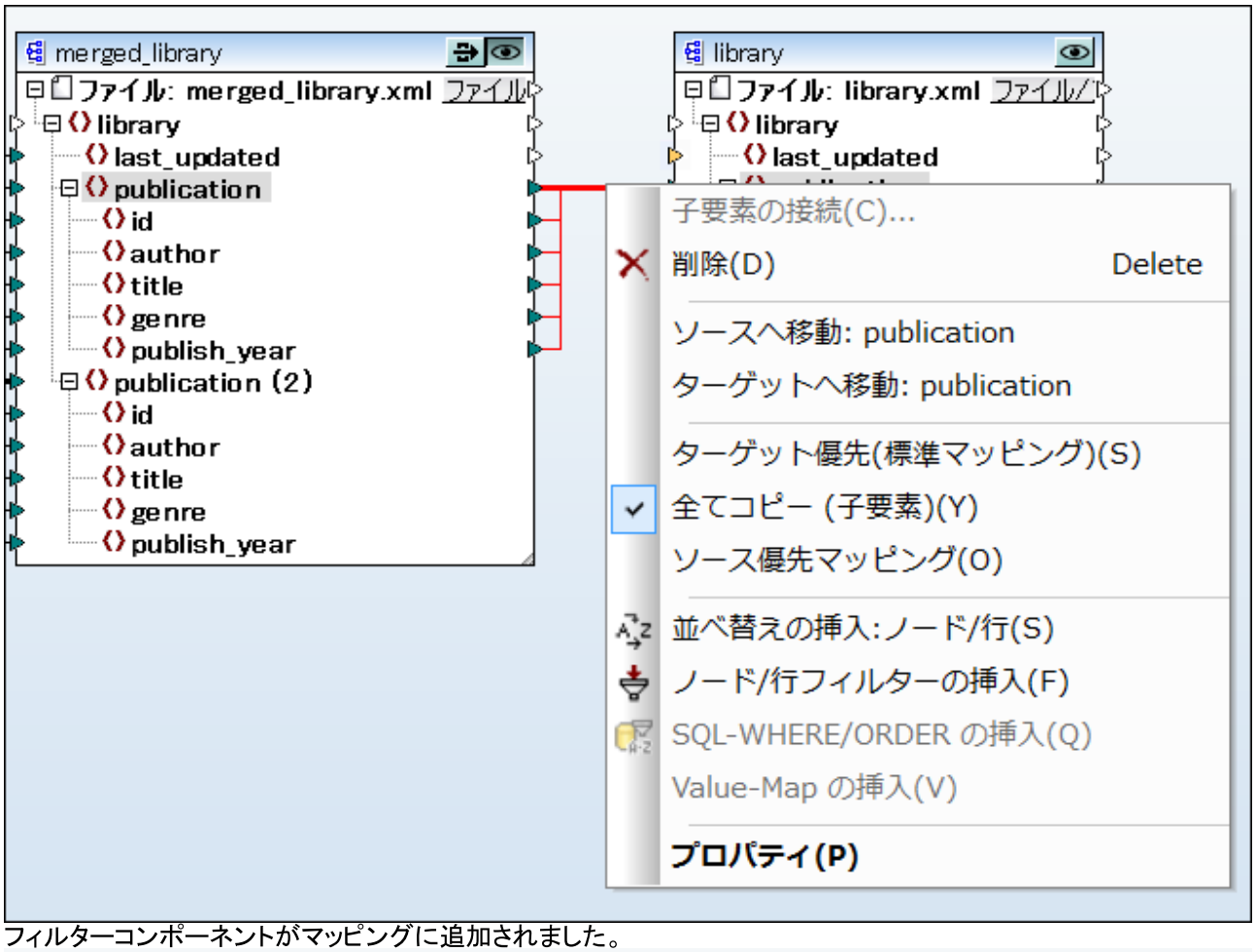


「OK」をクリックします。次の新規のボタンがターゲットコンポーネントの右上に表示されます:「**プレビュー**」() と「**パススルー**」()。次のステップでこれらのボタンは使用され、説明されます。

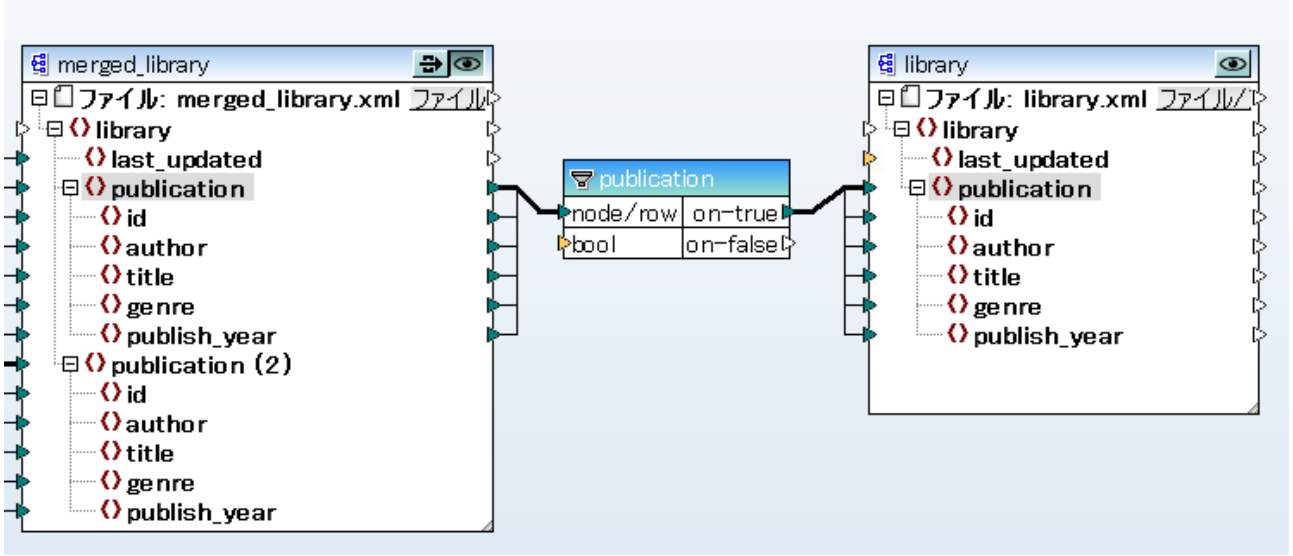


ステップ 4: データのフィルター

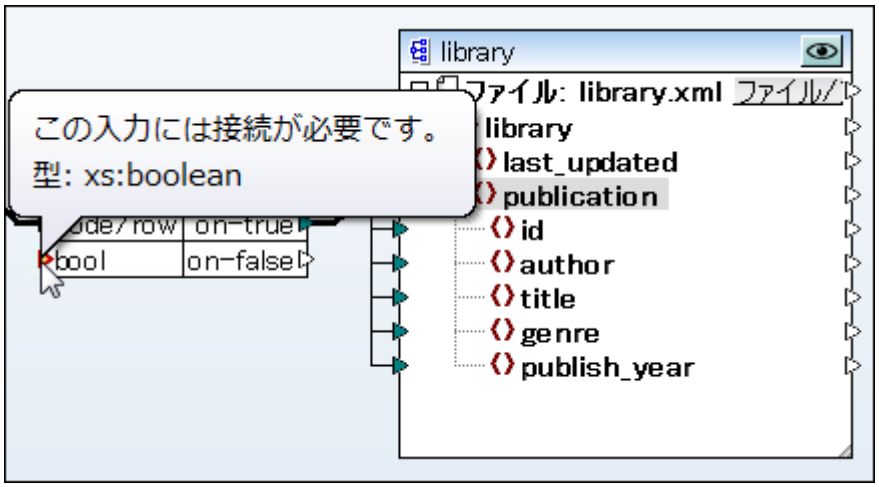
filtered_library に提供する前にデータをフィルターするには、「**フィルター**」コンポーネントを使用します。フィルターコンポーネントを追加するには、**merged_library** と**filtered_library**、間の接続を右クリックして、コンテキストメニューから、「node/row **フィルターの挿入**」を選択します。



フィルターコンポーネントがマッピングに追加されました。



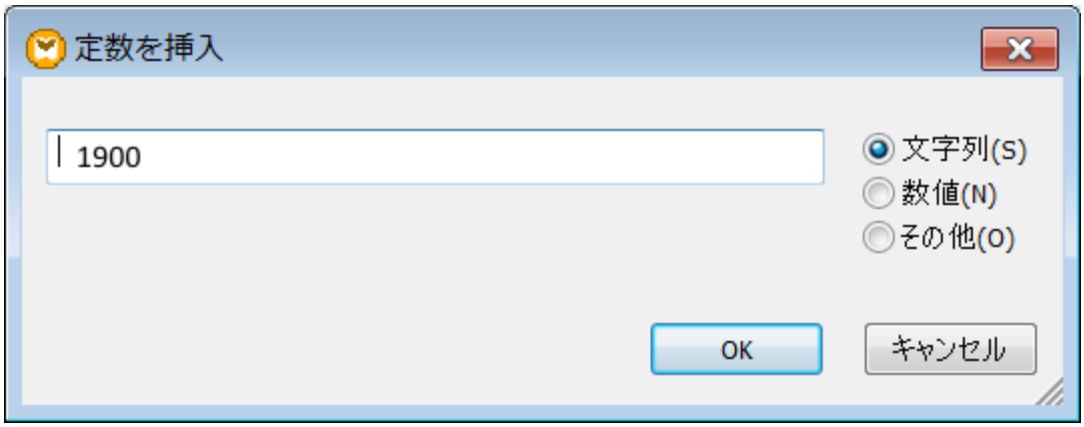
上に表示されるように、ブール入力コネクタは、入力が必要なことを示すオレンジ色にハイライトされます。コネクタ上にマウスをポイントすると、入力の型 `xs:boolean` が必要なことがわかります。ヒントを表示するには、「**ヒントの表示**」( ) ツールバーボタンが有効化されている必要があることにご注意してください。



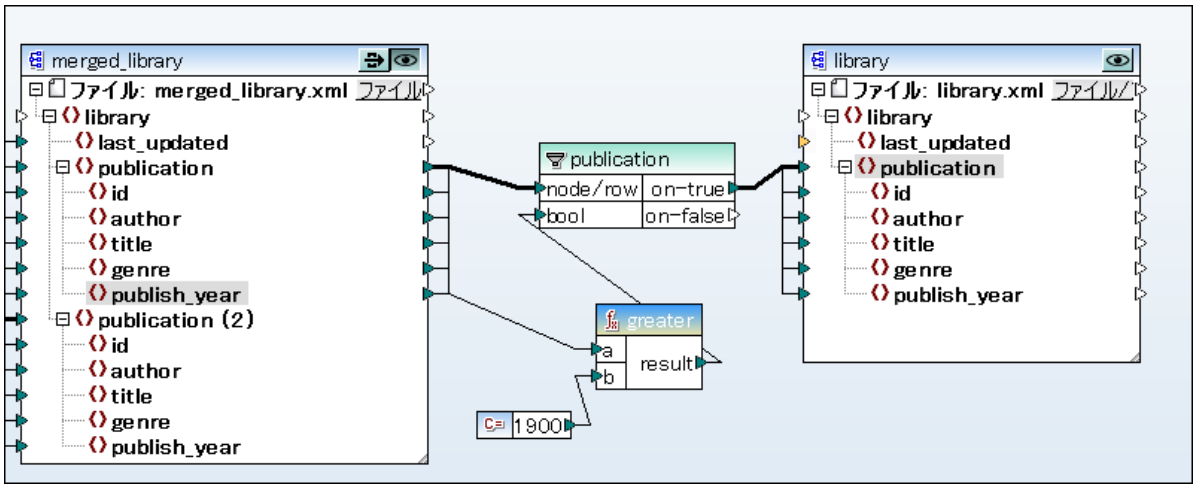
フィルターコンポーネントは `true` または `false` を返す条件を必要とします。ブール型の条件は、現在の**publication** シーケンスのデータがターゲットにコピーされると `true` を返します。条件が、 `false` を返すと、データはコピーされません。

このチュートリアルで使用される条件は1900 年以降に出版された書籍をフィルターします。条件を整えるために、、以下を行います:

1. 1900 の値を持つ定数の数値の型を追加します。(「**挿入**」メニューから「**定数**」をクリックします)。「**数値**」を型として選択します。



2. ライブラリウィンドウ内で、関数 `greater` を検索し マッピングペインにドラッグします。
3. `greater` 関数へと`greater` 関数から以下に表示されるように、マッピングを接続します。これを行うことにより、MapForce に以下を行うように命令します:「`publish_year`(出版年) が1900 以降の場合は、現在の`publication` ソースアイテムを`publication` ターゲットアイテム」にコピーします。



ステップ 5: 各ターゲットコンポーネントの出力をプレビューし保存する

ターゲットコンポーネントの出力をプレビューし保存することができます。複数のターゲットコンポーネントが同じマッピングに存在する場合、「プレビュー」( ) ボタンをクリックすることにより、プレビューの対象を選択することができます。「プレビュー」ボタンが押された状態 ( ) の場合、特定のコンポーネントが現在プレビューのために有効化されていることを示します (そして、この特定のコンポーネントが「プレビュー」ペイン内に出力を生成します)。一度に 1 つのコンポーネントのみがプレビューのために有効化することができます。

ですから、**merged_library** (すなわち、「中間」) コンポーネントの出力を確認し保存する場合は、以下を行います:

1. **merged_library** コンポーネントの「プレビュー」ボタン ( ) をクリックします。
2. 「出力」ボタン マッピングペインの下のをクリックします。
3. 出力をファイルに保存するには、「出力」メニューから「出力ファイルの保存」をクリックします。

filtered_library コンポーネントの出力を表示して保存するには、以下を行います:

1. **merged_lilbrary** コンポーネント上の「パススルー」ボタン ( ) をクリックします。
2. **filtered_library** コンポーネント上の「プレビュー」ボタン ( ) をクリックします。
3. マッピングペインの下の「出力」ボタンをクリックします。
4. 出力をファイルに保存するには、「出力」メニューから「出力ファイルを保存」をクリックします。

「パススルー」( ) ボタンー をクリックまたはクリックしないことにより、複数のターゲットコンポーネントを持つマッピングに大きな違いをもたらします。このボタンが押された状態 ( ) である場合、MapForce は、データに中間コンポーネントをパススルーさせ、マッピング全体の残りをプレビューすることができます。

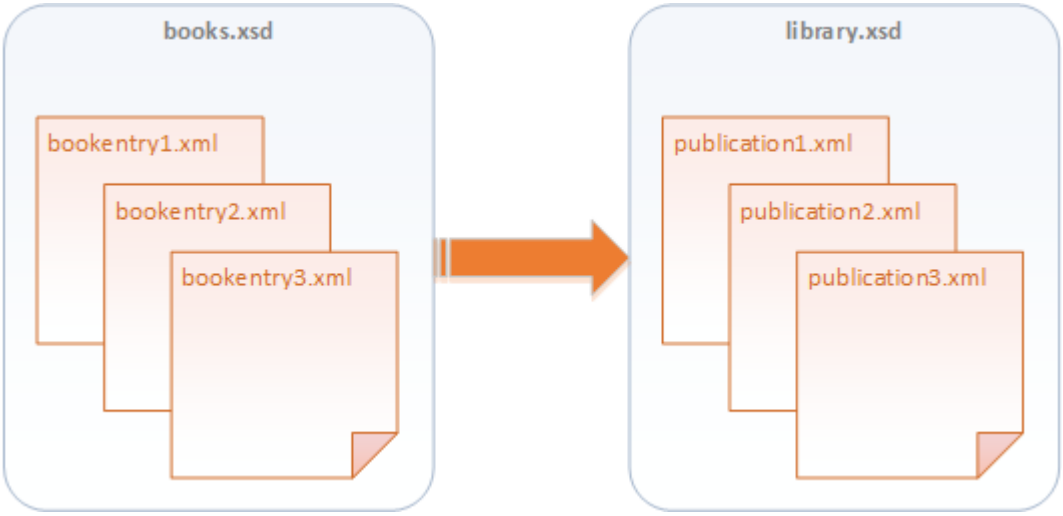
merged_library と **filtered_library** 間のマッピングの部分のみをプレビューするには、ボタン ( ) をリリースします。後者の場合、エラーが生成されます。中間コンポーネントはデータを読み込むべき妥当な入力 XML ファイルを持たないため、この振る舞いは予期されます。問題を解決するには、 コンポーネントのヘッダーをダブルクリックし、編集して、妥当な入力 XML ファイルを提供します。

複数のターゲット コンポーネントを持つマッピングのデザインが終わると、このチュートリアルの目的である、各ターゲットの出力を確認して保存することができます。パススルーコンポーネントに関しての詳細は、 チェーンマッピング/ パススルーコンポーネントを参照してください。

2.4 ファイルを動的に処理と生成する

このチュートリアルは、複数のソースXMLファイルからデータを読み込み、同じ変換内で複数のターゲットファイルに書き込む方法を説明します。このテクニックを説明するために、次の目的を持つマッピングを作成します:

1. 同じディレクトリ内の複数のXMLファイルからのデータを読み込む。
2. 新規のXMLスキーマに各ファイルを変換する。
3. 各ソースXMLファイルのために、新しいスキーマの下で新規ターゲットファイルを生成する。
4. 生成されたファイルからXMLと名前空間宣言を削除する。



データ変換の抽象的なモデル

3つのソースXMLファイルをサンプルとして使用します。ファイルの場所は**マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial** フォルダーで確認することができ**bookentry1.xml**、**bookentry2.xml**, と**bookentry3.xml** という名前が付けられています。これらのファイルはそれぞれ単一のブックを保管します。

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
   <book id="1">
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <category>Fiction</category>
      <year>1876</year>
   </book>
</books>
```

bookentry1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
   <book id="2">
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <category>Fiction</category>
      <year>1912</year>
   </book>
</books>
```

bookentry2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="books.xsd">
   <book id="3">
      <author>Herman Melville</author>
      <title>Moby Dick</title>
      <category>Fiction</category>
      <year>1851</year>
   </book>
</books>
```

bookentry3.xml

ソースXML ファイルは次のフォルダーで使用することのできる **books.xsd** スキーマを使用します: **<マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial**。ソースファイルを新規 XML スキーマに変換するには、(同じフォルダーで見つけることのできる) **library.xsd** スキーマを使用します。変換の後、マッピングは3つのファイルを新しいスキーマに従い生成します (下のコードリストを参照)。生成されたファイルが以下のようになるよう、マッピングを構成します: **publication1.xml**、**publication2.xml**、と**publication3.xml**。XML 宣言と名前空間が削除されなければならない点に注意してください。

```
<library>
   <publication>
      <id>1</id>
      <author>Mark Twain</author>
      <title>The Adventures of Tom Sawyer</title>
      <genre>Fiction</genre>
      <publish_year>1876</publish_year>
   </publication>
</library>
```

publication1.xml

```
<library>
   <publication>
      <id>2</id>
      <author>Franz Kafka</author>
      <title>The Metamorphosis</title>
      <genre>Fiction</genre>
      <publish_year>1912</publish_year>
   </publication>
</library>
```

publication2.xml

```
<library>
   <publication>
      <id>3</id>
      <author>Herman Melville</author>
      <title>Moby Dick</title>
      <genre>Fiction</genre>
      <publish_year>1851</publish_year>
   </publication>
</library>
```

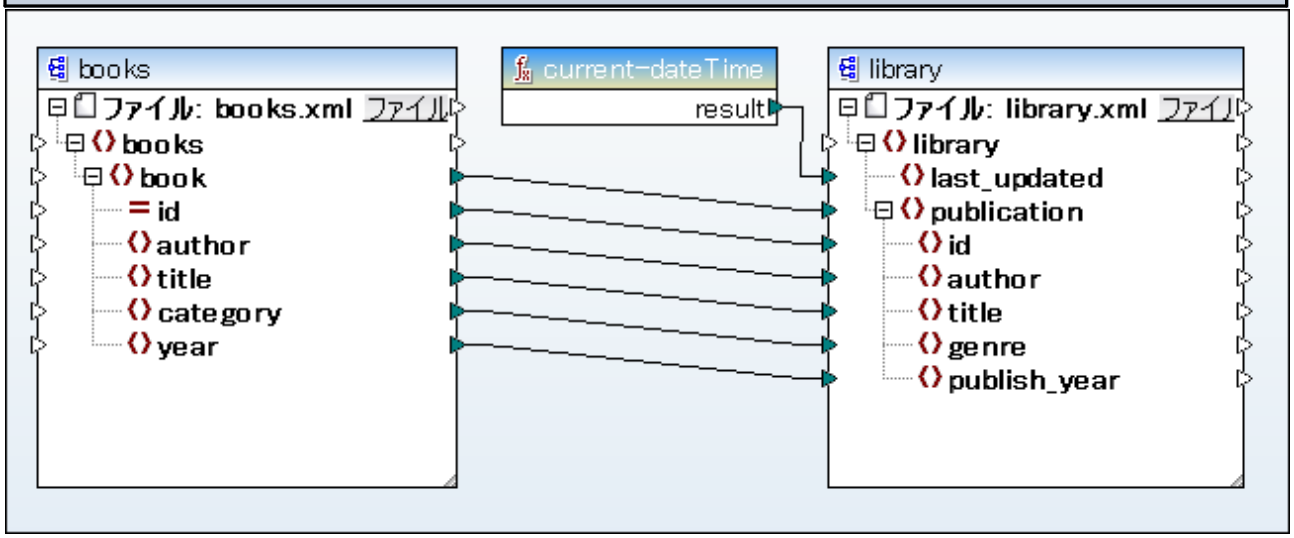
publication3.xml

チュートリアルの目的を達成するために、次のステップを踏みます.

ステップ 1: マッピングデザインファイルの準備

このチュートリアルは、開始のポイントとして<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダーから**BooksToLibrary.mfd** マッピングを使用しています。すでにXML を新しいスキーマに変換するチュートリアル内でこのマッピングをデザインしています。開始するには、MapForce で**BooksToLibrary.mfd** ファイルを開き、新しい名前で同じフォルダーに保存します。

複数のファイルを参照するため、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダー内に新しいマッピングが保存されることを確認してください。



BooksToLibrary.mfd (MapForce Basic Edition)

ステップ 2: 入力を構成する

MapForce に複数のXML インスタンスファイルを処理するように命令するには、ソースコンポーネントのヘッダーをダブルクリックします。コンポーネント設定ダイアログボックス内に入力ファイルとして**bookentry*.xml** を入力します。

コンポーネント設定 ダイアログボックス

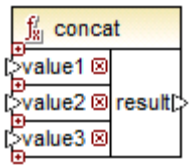
ファイル名内のアスタリスク（＊）ワイルドカード文字は、マッピング入力に**bookentry-** プレフィックスを伴う全てのファイルを使用するように命令します。パスは相対的であるため、MapForce はマッピングファイルと同じディレクトリ内の全ての**bookentry-** ファイルを検索します。マッピングファイル ＊ ワイルドカード文字を使用しているため、必要であれば、絶対パスを入力することもできます。

ステップ 3: 出力を構成する

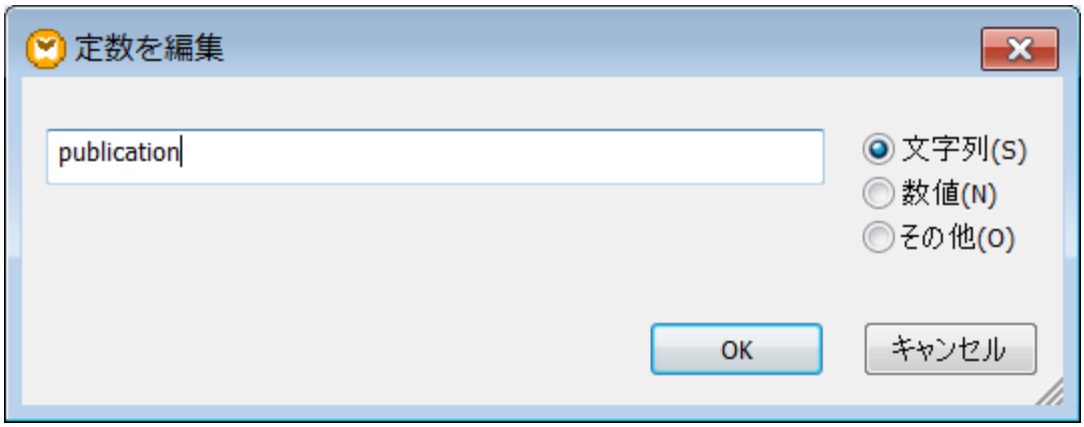
各出力ファイルのファイル名を作成するには、`concat` 関数 を使用します。この関数は引数に与えられた全ての値を連結します。

`concat` 関数を使用してファイル名を作成する

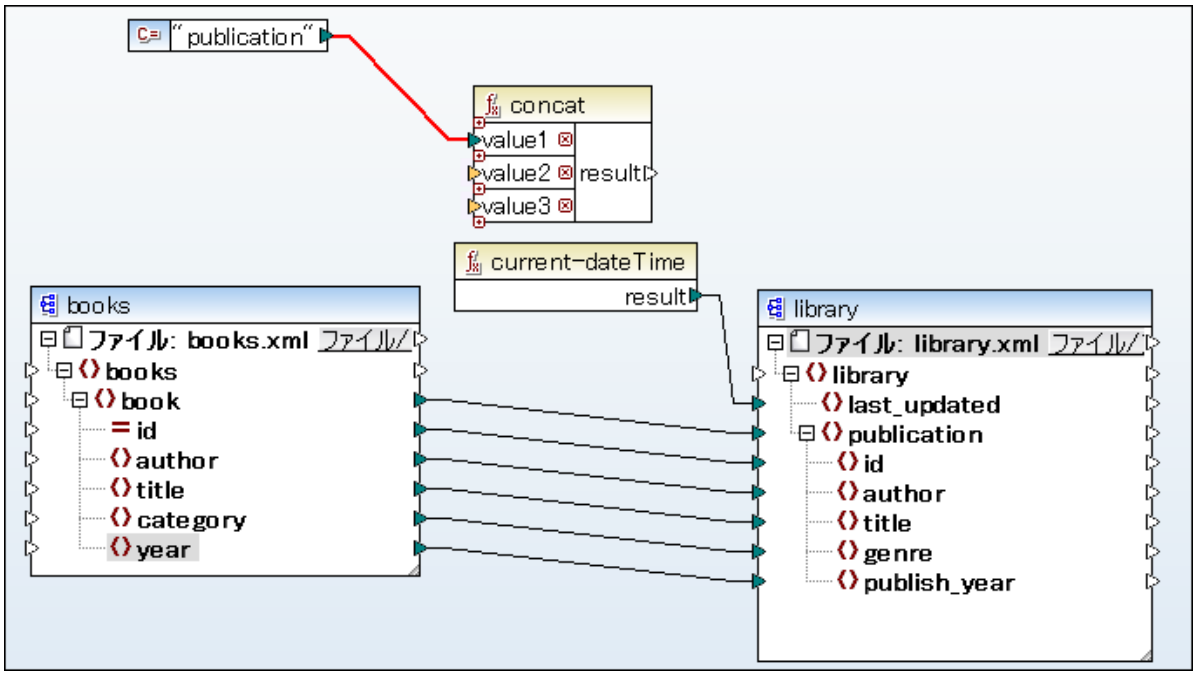
1. `concat` 関数 をライブラリウィンドウで検索し、マッピングエリアにドラッグします。デフォルトでは、この関数は パラメータが2つある マッピングに追加されます。しかし、必要であれば新しい パラメーターを追加することができます。関数コンポーネント内の「**パラメーターの追加**」（⊡）シンボルをクリックして、3番目の パラメーターを追加します。「**パラメーターの削除**」（⊠）シンボルをクリックすると パラメーターは削除されることに注意してください。



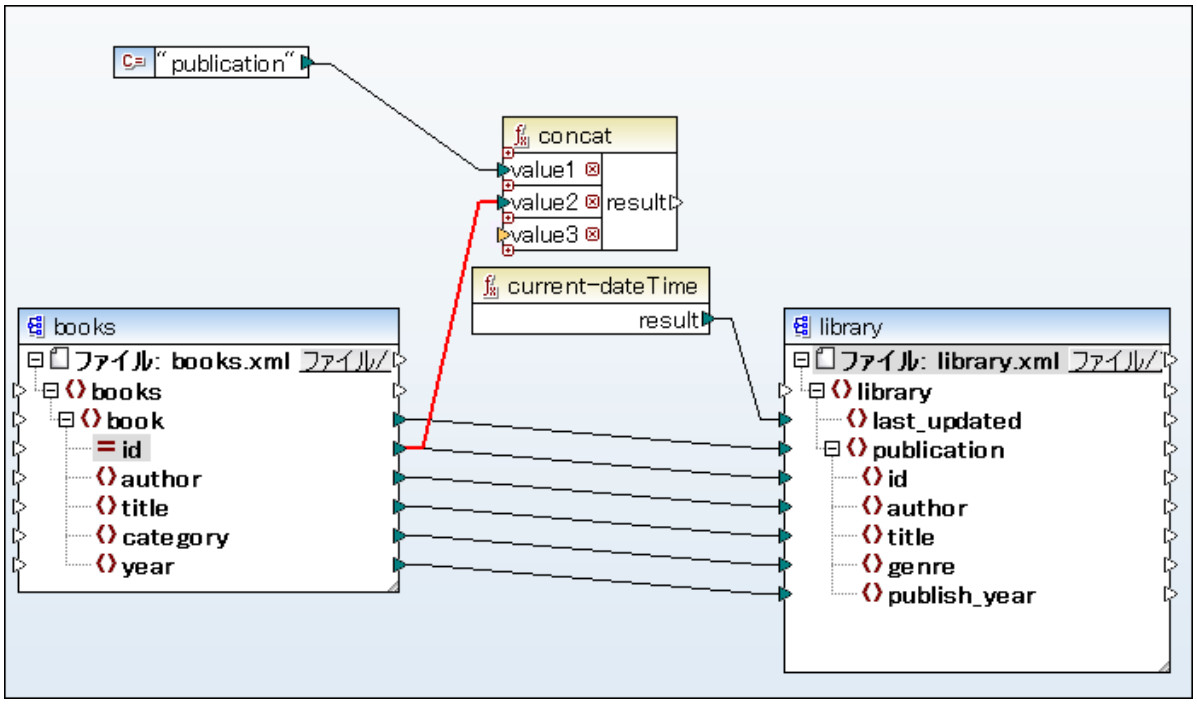
2. 定数を入力するには、（「**挿入**」メニューから「**定数**」をクリックします）。値を提供するようにプロンプトされると、「publication」を入力して、**文字列 オプション** を変更しないでそのままにしておきます。



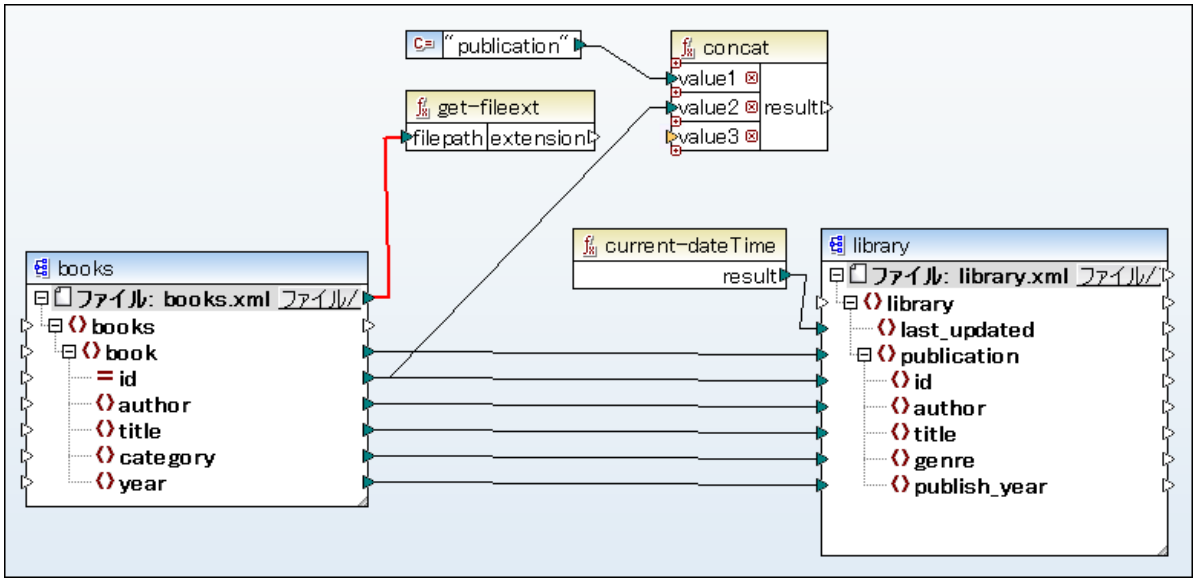
3. `concat` 関数の**value1** と定数をと接続します。



4. ソースコンポーネントの`id` 属性を`concat` 関数の**value2** と接続します。



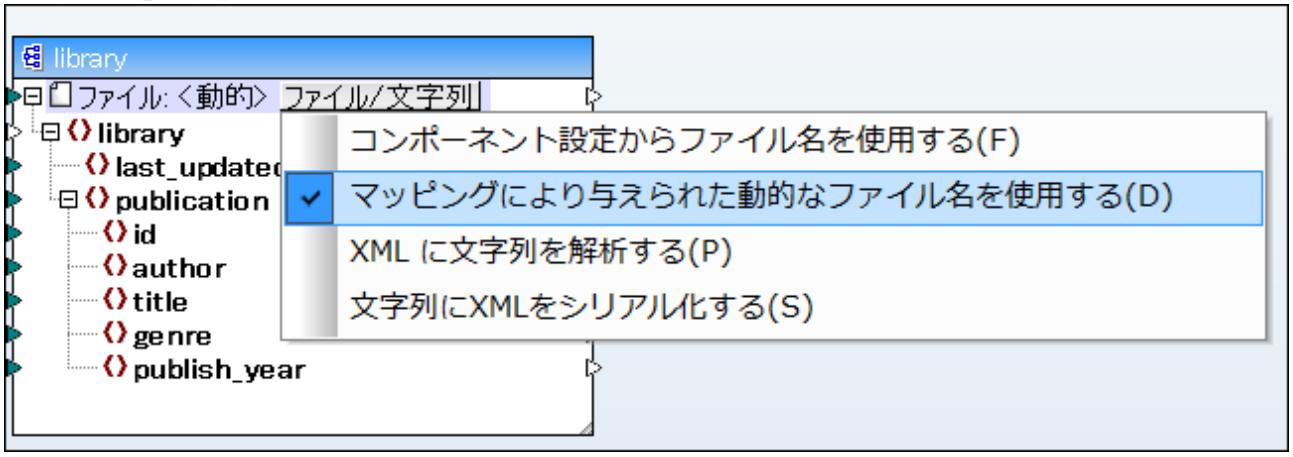
5. ライブラリウィンドウ内で `get-fileext` 関数を検索し、マッピングエリアにドラッグします。ソースコンポーネント(**File: books.xml**) のトップノードからこの関数の**filepath** パラメータへ接続を作成します。`get-fileext` 関数の結果から `concat` 関数の**value3** への接続を作成します。これを行うことにより、ソースファイル名から拡張子の部分(この場合、.xml)を抽出することができます。



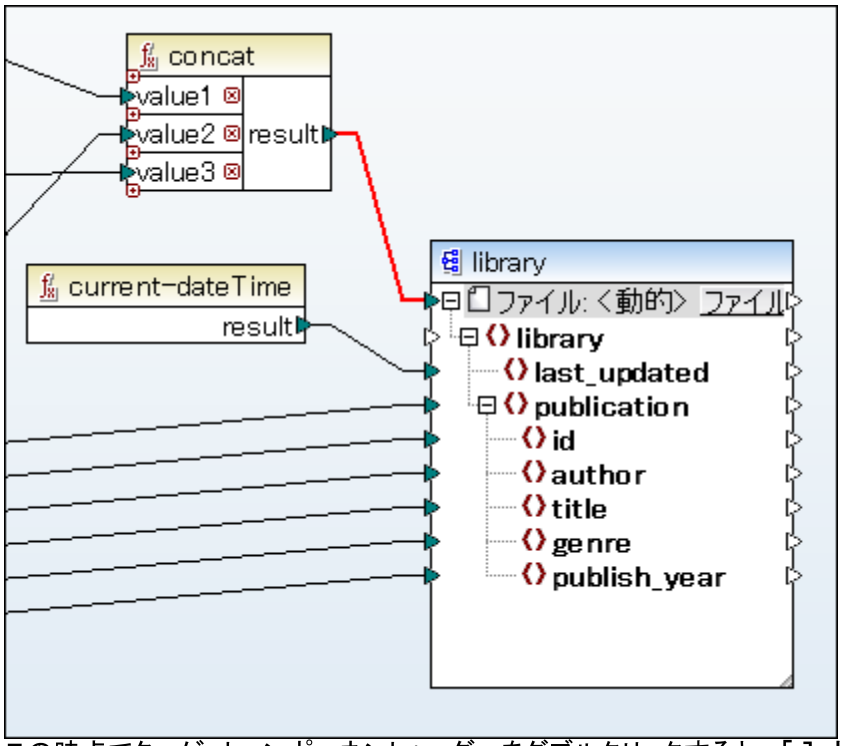
`concat` 関数のパラメーターに、連結されると生成されるファイル名 (例: **publication1.xml**) を作成する3つの値を提供しました

パート	サンプル
定数「publication」は定数文字列の値「publication」を与えます。	publication
ソースXMLファイルの属性 `id` は、各ファイルのための一意の識別子の値を与えます。これは全てのファイルが同じ名前で生成されないようにするためです。	1
`get-fileext` 関数は、生成されるファイルの拡張子を返します。	.xml

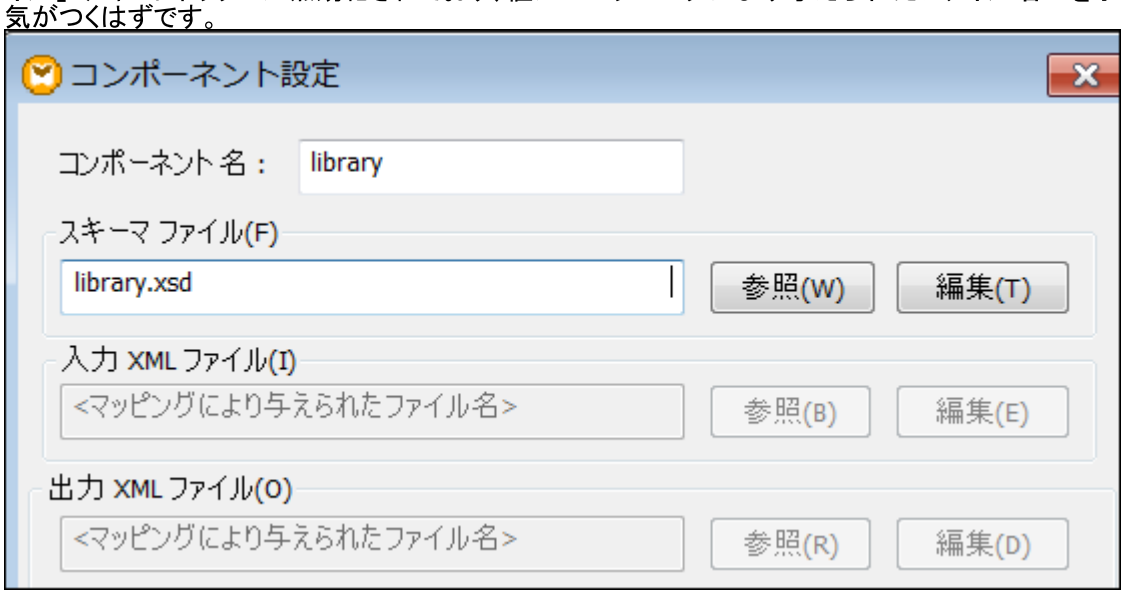
MapForceにマッピングが実行される際に実際にファイル名を作成するように命令することができます。これを行うには、「ファイル」（File）をクリックする、または ターゲットコンポーネントの「ファイル/文字列」（File/String）ボタンから「マッピングから与えられた動的なファイル名を使用する」 を選択します



マッピングにより与えられた名前を伴うMapForceにインスタンスファイルを動的に生成するように命令しました。この特別な例では、`concat` 関数により名前が作成されました。ですから、`concat` 関数の結果とターゲットコンポーネントのファイル: <動的> ノードを接続します。

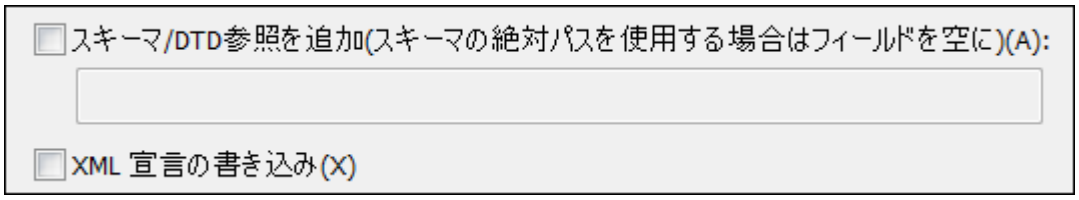


この時点でターゲットコンポーネントヘッダーをダブルクリックすると、「入力 XML ファイル」と「出力 XML ファイル」テキストボックスが無効化されており、値が <マッピングにより与えられたファイル名> を示していることに気がつくはずです。

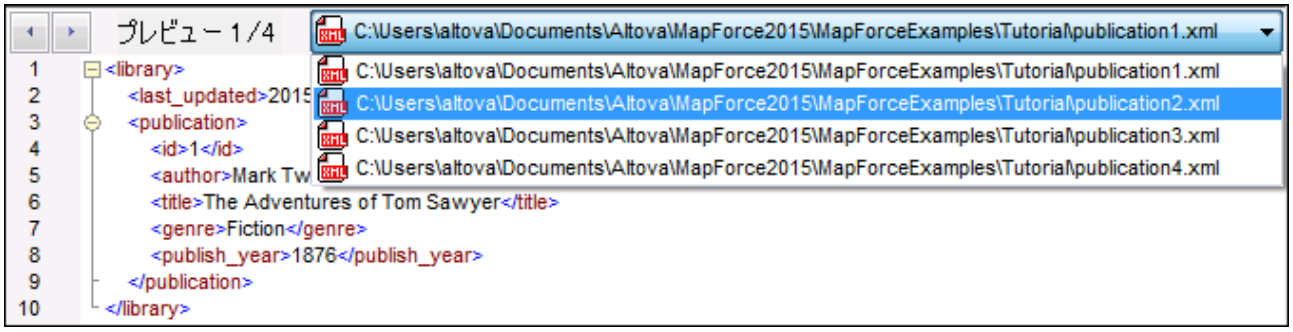


これは、インスタンスファイル名を動的にマッピングから提供したことを示し、コンポーネント設定内で定義することは関連性がありません。

最後に、ターゲットから XML 名前空間とスキーマ宣言を削除する必要があります。これを達成するには、コンポーネント設定ダイアログボックスの「スキーマ /DTD レファレンスの追加...」と「XML 宣言の書き込み」チェックボックスから選択を解除します。



結果と生成されたファイル名を確認するために、マッピングを実行します。このマッピングは複数の出力ファイルを生成します。出力ペインの左上の左と右ボタンを使用して出力ファイル全体を移動することができます。または 横にあるドロップダウンリスト からファイルを選択します。



3 一般的なタスク

このチャプターは、マッピングとの作業、コンポーネント、接続などのMapForce の全般的なタスクとコンセプトを説明します。

3.1 マッピングとの作業

MapForce マッピングデザイン（または、略して「マッピング」）は、データがどのように1つのフォーマットから他のフォーマットに変換されるかの視覚的な表現です。マッピングは、データ変換を行うためのMapForce マッピングエリア内で1つのスキーマから他のスキーマに追加するコンポーネントから構成されています。（例: XMLドキュメントを1つのスキーマから他のスキーマに変換）。有効なマッピングは、1つまたは複数のターゲットコンポーネントに接続されている、1つまたは複数の複数のソースコンポーネントから構成されています。マッピングを実行して、結果を直接 MapForce 内で確認することができます。コードを生成し、外部で実行することも可能です。MapForce 実行可能ファイルにマッピングをコンパイルし、MapForce Server またはFlowForce Server を使用してマッピングの実行を自動化することもできます。MapForce は、マッピングを.mfd の拡張子を持つファイルとして保存します。

新規マッピングの作成方法:

1. 以下の内の1つを行ってください：
   - 「ファイル」メニューから「新規作成」をクリックします。
   - 「新規作成」（  ）ツールバーボタンをクリックします。

マッピングが作成されました、しかしながら、空のため何も起こりません。最低でも妥当なマッピングは2つの接続されたコンポーネント を必要とし次のステップはマッピングにコンポーネントを追加し 接続線を描きます。

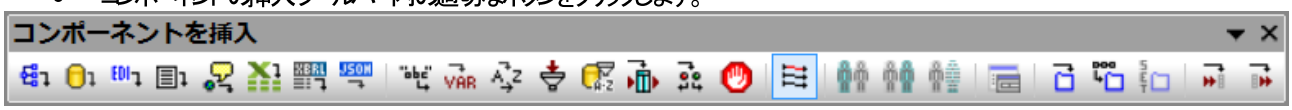
3.1.1 マッピングにコンポーネントを追加する

MapForce 内では、「コンポーネント」という用語はデータの構造（スキーマ）またはデータがどのように変換（関数）されるかを視覚的に表します。コンポーネントはすべてのマッピングを構築する中心的な役割を果たします。マッピングエリアでは、コンポーネントは正方形の枠で表示されます。以下は、MapForce コンポーネントの例です:

- 定数
- フィルター
- 条件
- 関数コンポーネント
- EDI ドキュメント（UN/EDIFACT、ANSI X12、HL7）
- Excel 2007+ ファイル
- 単純型 入力コンポーネント
- 単純型 出力コンポーネント
- スキーマおよびDTD

マッピングにコンポーネントを追加するには以下を行います:

- 「挿入」メニューから追加するコンポーネントの型（例: 「XML スキーマ/ファイル」をクリックします。
- Windows ファイルエクスプローラーからファイルをドラッグしてマッピングエリアにドロップします。このオペレーションは互換性のあるファイルベースのコンポーネントのみに対して使用することができます。
- コンポーネントの挿入ツールバー内の適切なボタンをクリックします。



コンポーネントの挿入ツールバー（MapForce Enterprise Edition）

それぞれのコンポーネントの型には、特定の目的と振る舞いがあります。コンポーネントの型が必要な箇所では、MapForce は、コンテキストウィザードステップまたはダイアログボックスで手順を踏んで、説明を行います。例: XML スキーマを追加する場合、通知ダイアログボックスがインスタンスファイルをオプションで選択するように促します。

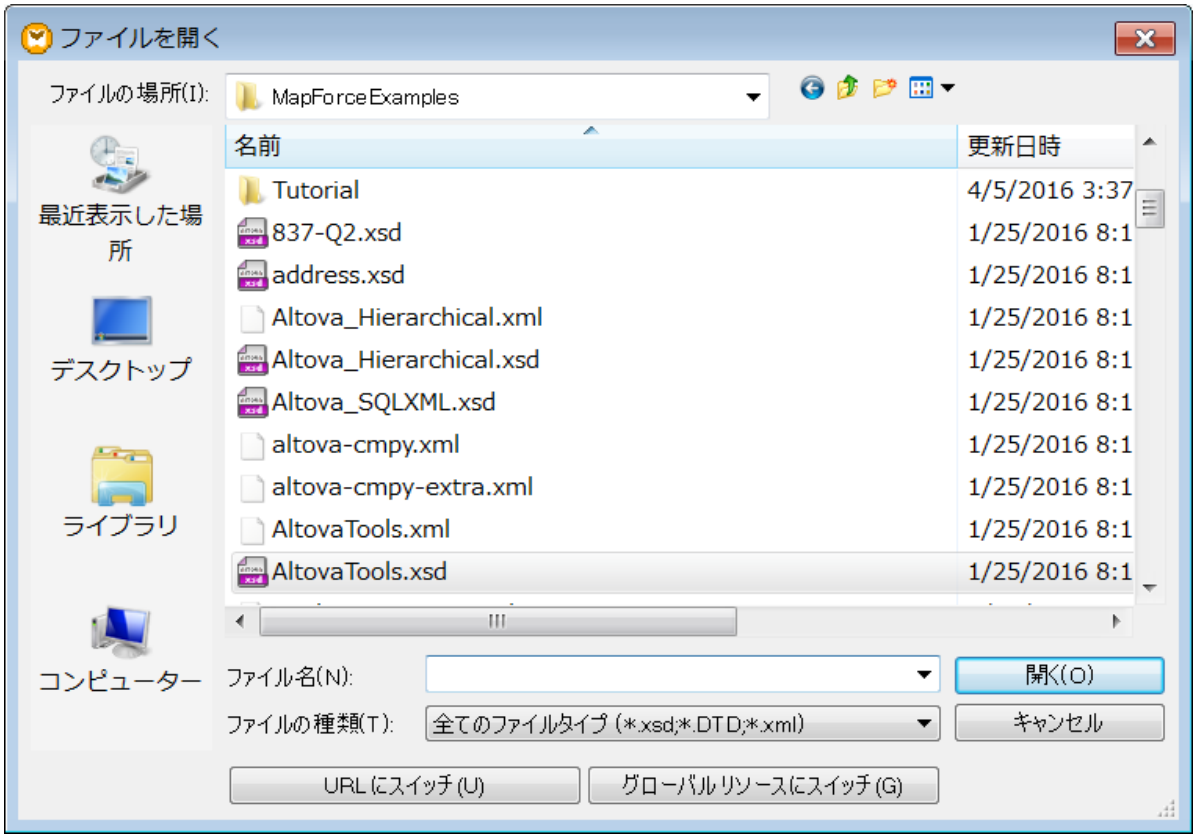
コンポーネントについての説明は、 コンポーネントとの作業を参照してください。マッピングソースまたはターゲットとしてサポートされる個々の技術に関しては、データソースとターゲットを参照してください。データを一時的に保管または変換する MapForce ビルドインコンポーネントに関しては、または(フィルタリングまたは並べ替えなどの) マッピングのデザインを参照してください。

3.1.2 URL からコンポーネントを追加する

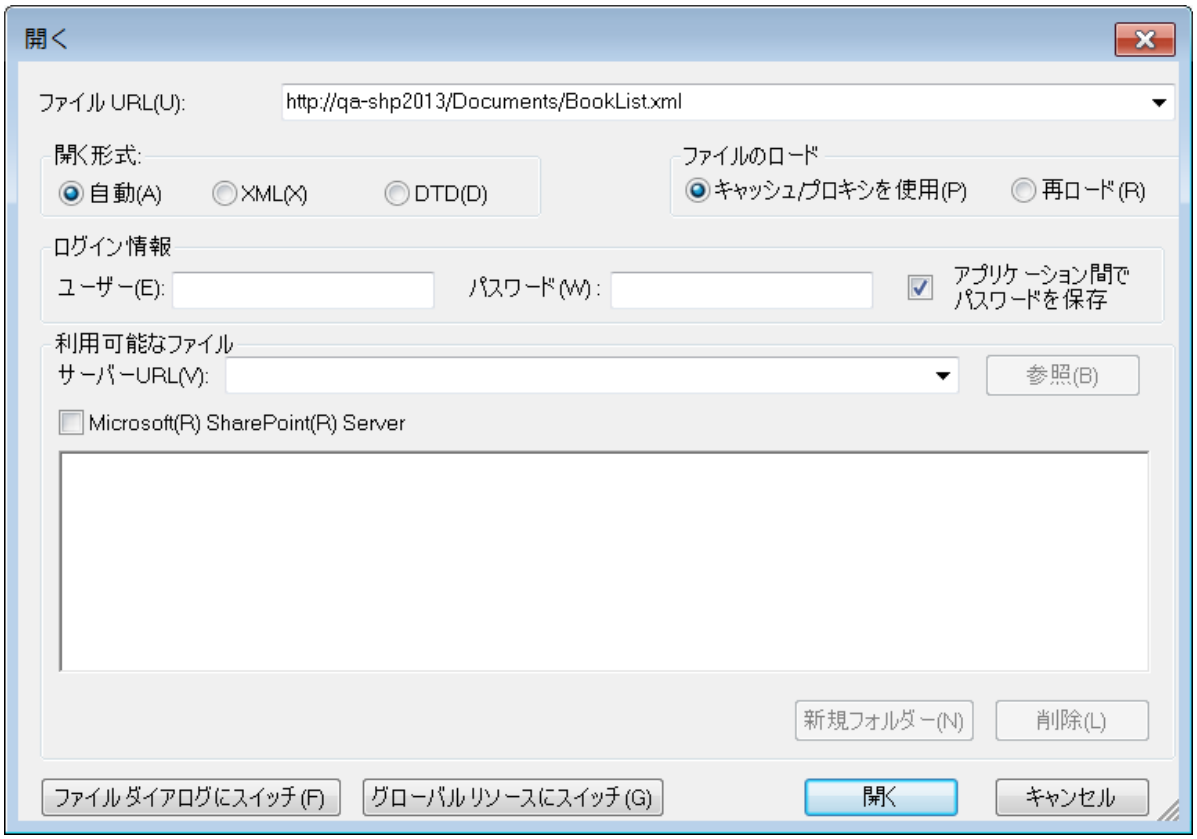
マッピングコンポーネントとしてローカルファイルを追加するだけでなく、URL からファイルを追加することができます。この操作はコンポーネントをソースコンポーネントとして追加した場合のみサポートされます。(すなわち、マッピングは、リモートファイルからデータを読み込みます)。サポートされるプロトコルは HTTP、HTTPS、と FTP です。

URL からコンポーネントを追加する:

1. 「挿入」メニューから追加するコンポーネントの型を選択します(例: XML スキーマ/ファイル)。
2. 「開く」ダイアログボックスから「URL にスイッチ」 をクリックします。



3. 「ファイル URL」テキストボックスにファイルの URL を入力して、「開く」をクリックします。



「ファイル URL」テキストボックス内のファイル型がステップ 1 で指定されたファイル型と同じであることを確認してください。

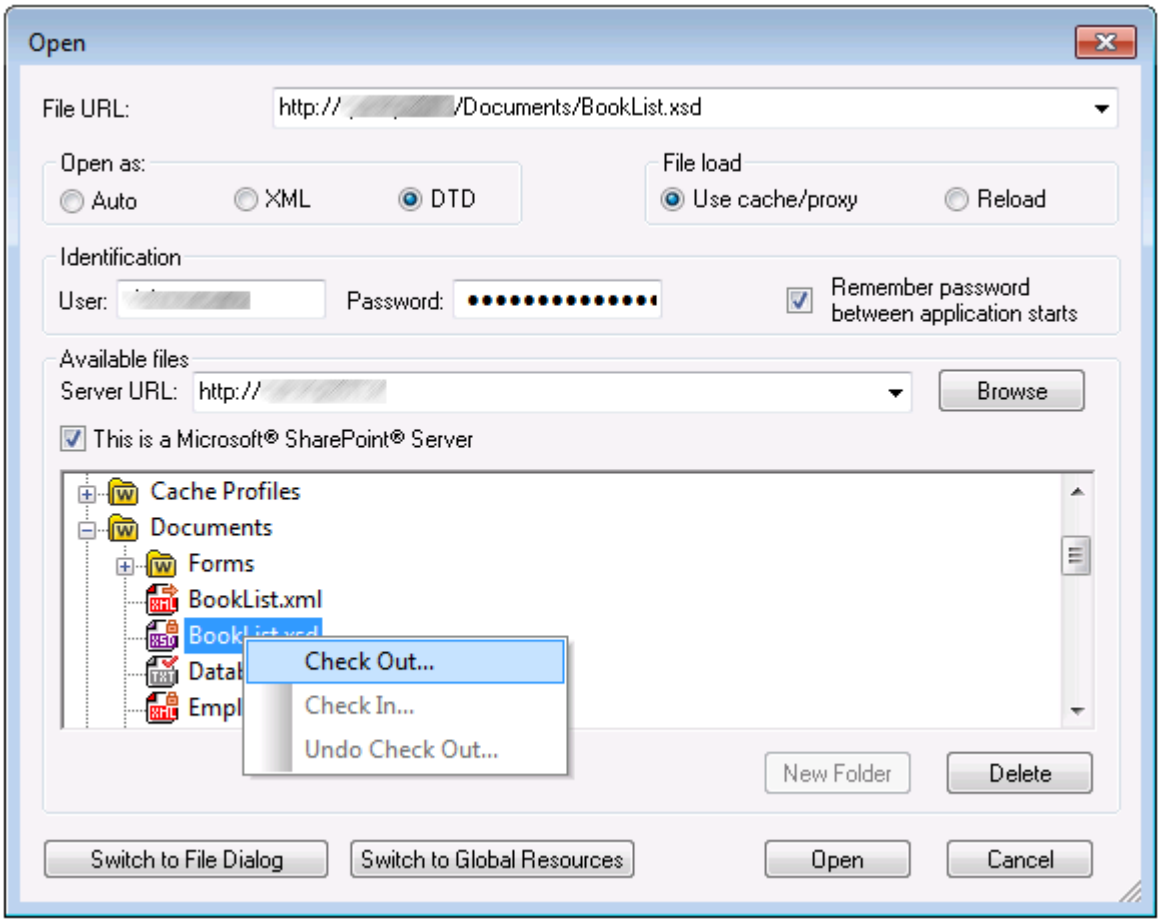
サーバーがパスワード認証を必要とする場合、ユーザー名とパスワードを入力するようにプロンプトされます。次回 MapForce を起動する際に使用できるように、ユーザー名とパスワードを保存するには、「開く」ダイアログボックス内の**「アプリケーション間でパスワードを保存」**チェックボックスを選択します。

「開く形式」設定は、ファイルを開く際のパーサーのための文法を定義します。デフォルトと奨励されるオプションは**「自動」**です。

アップロードするファイルに変更が加えられない場合、**「キャッシュ/プロキシの使用」**オプションを選択して、データをキャッシュし、ロードをスピードアップします。それ以外の場合、マッピングを開くたびにファイルを再ロードする場合は、**「再ロード」**を選択します。

Web 分散オーサリングとバージョン管理（WebDAV）をサポートするサーバーに関しては、**「サーバー URL」**テキストボックスにサーバー URL を入力した後にファイルを**「参照」**をクリックして、参照することができます。参照は全てのファイルの型を表示しますが、上記のステップ 1 で指定されたファイル型と同じ型が選択されていることを確認してください。それ以外の場合、エラーが発生します。

サーバーがMicrosoft SharePoint Server の場合、**「これは Microsoft SharePoint Server です」**チェックボックスをチェックしてください。これを行うことにより、プレビューエリア内のファイルのチェックインとチェックアウトの状態が表示されます。MapForce を使用中に他のユーザーがファイルを編集できないようにするためには、ファイルを右クリックして、**「チェックアウト」**を選択します。ユーザーにより以前チェックアウトされたファイルをチェックするには、ファイルを右クリックして、**「チェックイン」**を選択します。



開くダイアログボックス（URL に切り替える モード）

3.1.3 変換言語の選択

データ変換言語として次を選択することができます:

- XSLT 1.0
- XSLT 2.0
- XSLT 3.0

変換言語を選択するには、以下を行います:

- 「出力」 メニューから変換に使用する言語をクリックします。
- 言語選択ツールバー内の言語名をクリックします。

マッピングの変換言語を変更する場合、MapForce 機能の一部はその言語のためにサポートされない場合があります。詳細に関し

では、サポートメモを参照してください。

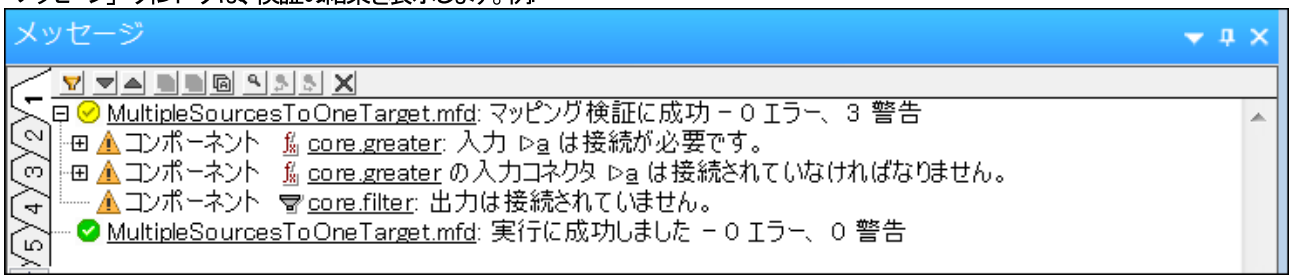
3.1.4 マッピングの検証

MapForce は変換結果をプレビューするために「出力」 タブをクリックすると、マッピングを自動的に検証します。結果をプレビューする前にマッピングのみを検証することもできます。この機能は、マッピングのエラーと警告のマッピングを実行する前に認識し修正するために役にたちます。マッピングの実行は、処理されたデータにより更なるランタイムエラーまたは警告を生成する場合があります。例: 属性にマップされた値の上書きなど。

マッピングを正確に検証するには、以下を行います:

- 「ファイル」メニューから「マッピングの検証」をクリックします。
- 「検証」( ) ツールバーボタンをクリックします。

「メッセージ」ウィンドウは、検証の結果を表示します。例:



メッセージウィンドウ

マッピングを検証するには、MapForce は、マッピングの妥当性をチェックし、(無効または見つからない接続、サポートされないコンポーネントの種類などの) 検証結果は、以下のステータスアイコンと共にメッセージウィンドウに表示されます:

アイコン	意味
	変換は成功しました。
	変換は警告と伴い完了しました。
	変換は失敗しました。

「メッセージ」ウィンドウは以下のメッセージの型を表示する場合があります: 情報メッセージ、警告、とエラー。

アイコン	意味
	情報メッセージを表示します。情報メッセージはマッピングの実行を停止しません。
	警告メッセージを表示します。警告はマッピングの実行を停止しません。警告メッセージは以下の場合に生成される可能性があります。例: 必須の入力コネクタへの接続が作成されなかった場合。このような場合、出力は、有効な接続が存在する場所で、これらのコンポーネントのために生成されます。
	エラーを表示します。エラーが発生すると、マッピングの実行に失敗すると、出力は生成されません。XSLT または XQuery コードのプレビューを行うこともできません。

マッピングエリア内の情報、警告または、エラーメッセージをトリガーしたコンポーネントまたは構造をハイライトするには、メッセージウィンドウ内の下線のついたテキストをクリックします。

データを変換するコンポーネントに関しては、(関数または変数など) MapForce 検証は以下のように動作します:

- 必須の**入力コネクタ**が接続されていない場合、エラーメッセージが生成され、変換は停止されます。
- **出力コネクタ**が接続されていない場合、警告が生成され、変換プロセスは続行されます。問題のあるコンポーネントとそのデータは無視され、ターゲットドキュメントにマップされません。

個別のタブ内にそれぞれの検証の結果を表示するには、メッセージウィンドウの左横にある番号の付いたタブをクリックします。これはとても役にたちます。例: 複数のマッピングファイルと同時に作業する場合。

メッセージウィンドウ内の他のボタンにより以下のアクションを実行することができます:

- メッセージを型別にフィルターします(例: エラーまたは警告のみを表示)。
- エントリを上下に移動します。
- メッセージテキストをクリップボードにコピーします
- ウィンドウ内で特定のテキストを検索します
- メッセージウィンドウをクリアします

メッセージウィンドウの全般的な情報に関しては、ユーザーインターフェイスの概要を参照してください。

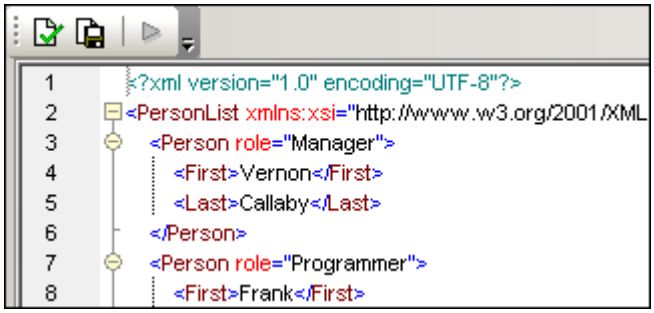
3.1.5 マッピング出力の検証

「**出力**」タブをクリックした後、マッピングをプレビューする場合は、「出力」ペインで出力の結果を確認することができます。この出力を関連するスキーマに対して検証することができます。例: マッピング変換がXML ファイルを生成する場合、結果 XML ドキュメントはXML スキーマに対して検証されます。

XML ファイルに関しては、コンポーネント設定ダイアログボックスの「**スキーマ/DTD レファレンスの追加**」フィールド内 のインスタンスファイルに関連するスキーマを指定することができます(XML コンポーネント設定 を参照)。生成されたXML 出力により参照されるスキーマファイルをパスは指定します。これにより、マッピングの実行時、出力インスタンスが検証されることができます。このフィールドに `http://` 、絶対または相対パスを入力することができます。「**スキーマ/DTD レファレンスの追加**」フィールドを選択しない場合、スキーマに対する、出力ファイルの検証は不可能です。このチェックボックスを選択し、空白のままにしておくと、検証が終わると、コンポーネント設定ダイアログボックスのスキーマファイル名が出力に生成されます。

コンポーネント設定を開くには、以下を行います:

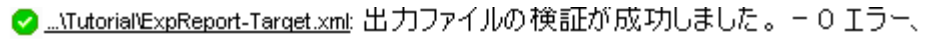
- 「**出力の検証**」  ツールバーボタンをクリックします。



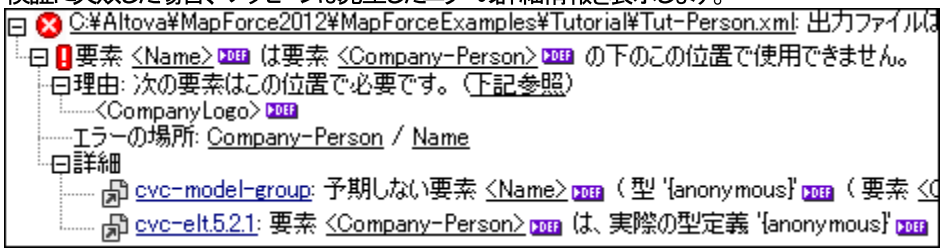
- 「出力」 メニューから「出力ファイルの検証」をクリックします。

メモ　「出力の検証」 ボタンと対応する メニューコマンド（「出力 | 出力ファイルの検証」）は、出力ファイルがスキーマに対する検証をサポートする場合のみ有効化されます。

検証の結果はメッセージウィンドウ内に表示されます。例:



検証に失敗した場合、メッセージは発生したエラーの詳細情報を表示します。



検証メッセージは更に詳しい詳細情報を含む多くのハイパーリンクを含みます。:

- ファイルパスをクリックすると、MapForce の「出力」 タブの結果の出力が開かれます。
- <ElementName> リンクをクリックすると、「出力」 タブ内の要素がハイライトされます。
- PDEF アイコンをクリックすると、XMLSpy（インストールされている場合）内の要素の定義が開かれます。
- 詳細サブセクションのハイパーリンク（例: cvc-model-group）をクリックすると、http://www.w3.org/ Web サイト上の対応する検証のルールについての詳細が開かれます。

3.1.6　出力のプレビュー

MapForce マッピングと作業する場合、結果出力を、外部のプロセッサーまたはコンパイラーを使用して生成されたコードを実行またはコンパイルすることなく、プレビューすることができます。全般的には、生成されたコードを外部で処理する前に、MapForce 内で変換出力をプレビューすることはよい考えです。

マッピングの結果をプレビューする場合、MapForce は、マッピングを実行して、結果出力を出力ペインに表示します。

出力ペインにデータが準備されると、必要に応じて検証し保存します（マッピング出力の検証を参照）。また、「検索」 コマンドを使用して、(**Ctrl + F** キーの組み合わせ) 簡単に特定のテキストのパターンを出力ファイル内で検索することができます。

マッピングの実行に関するエラー、警告、情報メッセージはメッセージウィンドウに表示されます（ユーザーインターフェイスの概要を参照）。

変換の出力をプレビューする方法:

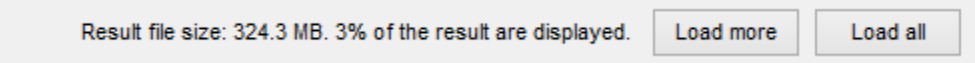
- マッピングウィンドウ下の「出力」 タブをクリックします。MapForce は、言語ツールバー内で選択された変換言語を使用してマッピングを実行し、結果の出力を使用して「出力」 ペインを作成します。

変換の出力を保存するには、以下を行います:

- 「出力」 メニューから「出力ファイルの保存」をクリックします。
- 「生成された出力の保存」 ツールバーボタンをクリックします。

部分的な出力のプレビュー

大きな出力ファイルをプレビューする場合、MapForce は、出力ペインに表示されるデータの量を制限します。具体的には、MapForce は、出力ペイン内でファイルの一部を表示します。そして、「更にロード...」ボタンがペインの下の部分に表示されます。「更にロード...」ボタンは現在表示されているデータのファイルの部分の横に表示されます。



メモ 出力ペイン内にファイル全体がロードされると、「整形出力」ボタンが有効化されます。

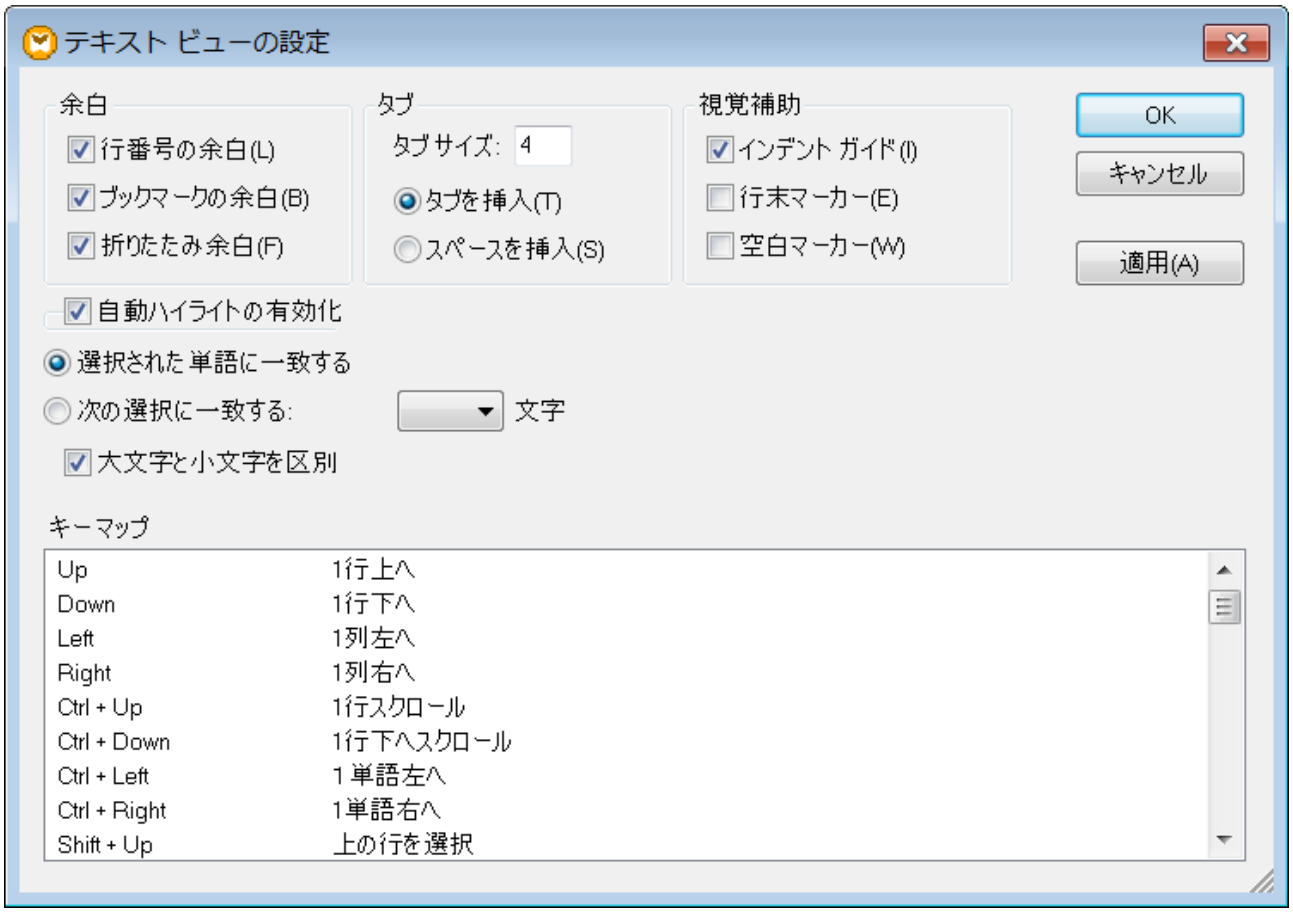
オプションダイアログボックスの**全般**タブからプレビューの設定を構成することができます（ MapForce オプションの変更を参照）。

3.1.7 テキストビューの機能

「出力」ペインと「XSLT」ペインには、テキストをより簡単に表示するための複数の視覚的な補助が搭載されています。以下のような機能が含まれます:

- 行番号
- 構文の色
- ブックマーク
- ソースの折りたたみ
- インデントのガイド
- 行末と空白文字のマーカー
- ズーム
- 整形出力
- ワード ラップ
- テキストのハイライト

適用できる箇所では、上記の機能を「**テキストビュー設定**」ダイアログボックスから切り替え、またはカスタム化することができます。「**テキストビュー設定**」ダイアログボックス内の設定は、アクティブなドキュメントだけではなく、アプリケーション全体に適用されます。

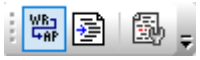


テキストビュー設定 ダイアログボックス

「テキストビュー設定」ダイアログボックスを開くには、以下を行います:

- **「出力」**メニューから**「テキストビュー設定」**を選択します。
- **「テキストビュー設定」**  ツールバーボタンをクリックします。
- 出力 ペインを右クリックして、コンテキストメニューから、**「テキストビュー設定」**を選択します。

ナビゲーションの補助の一部は、テキストビューツールバー、アプリケーションメニュー、キーボードショートカットから切り替えることができます。



テキストビューツールバー

適用することのできるショートカットへの参照は、上記の**「テキストビュー設定」**ダイアログボックスの「キーマップ」セクションを確認してください。

行番号

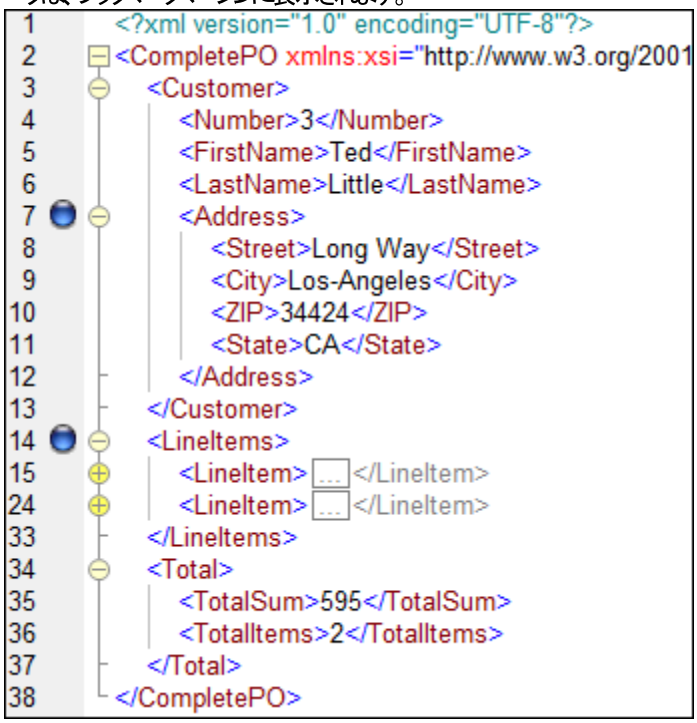
行番号は、**「テキストビュー設定」**ダイアログボックス内で切り替えることのできる行番号 マージンで表示されます。テキストのセクションが折りたたまれると、折りたたまれたテキストの行番号は表示されません。

構文の色

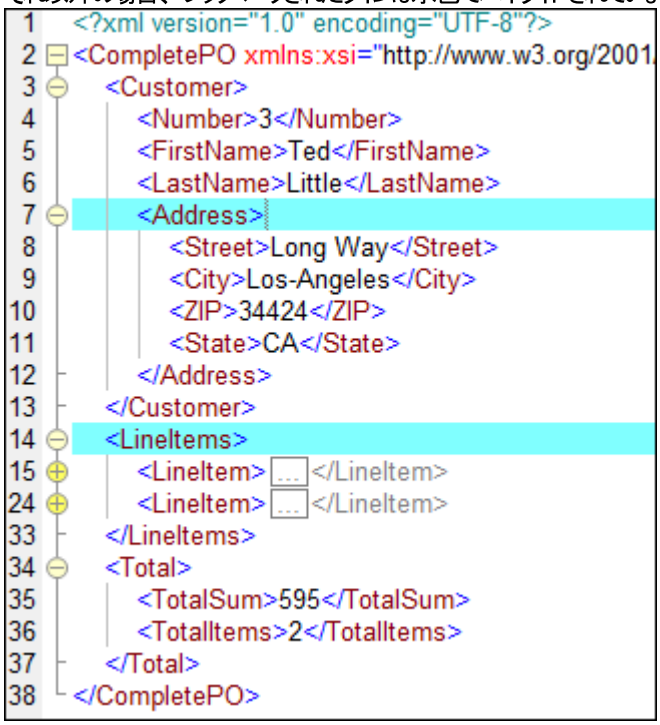
構文の色分けはテキストの構文の値に従い適用されます。例えば、XML ドキュメント内では、XML ノードが要素、属性、コンテンツ、CDATA セクション、コメント、処理命令、ノード名（そして一部の場合ではノードのコンテンツ）により異なる色分けが使用されています。

ブックマーク

ドキュメント内のラインは参照とアクセスのためにブックマークとして使用することができます。ブックマークマージンオンに設定されていると、ブックマークは、ブックマークマージンに表示されます。



それ以外の場合、ブックマークされたラインは水色でハイライトされています。



ブックマークマージンは、テキストビュー設定 ダイアログボックス内で切り替えることができます。

ブックマークを以下のコマンドを使用して編集、および、移動することができます:

ブックマークの挿入/削除 (Ctrl + F2)

次のブックマークへ移動 (F2)

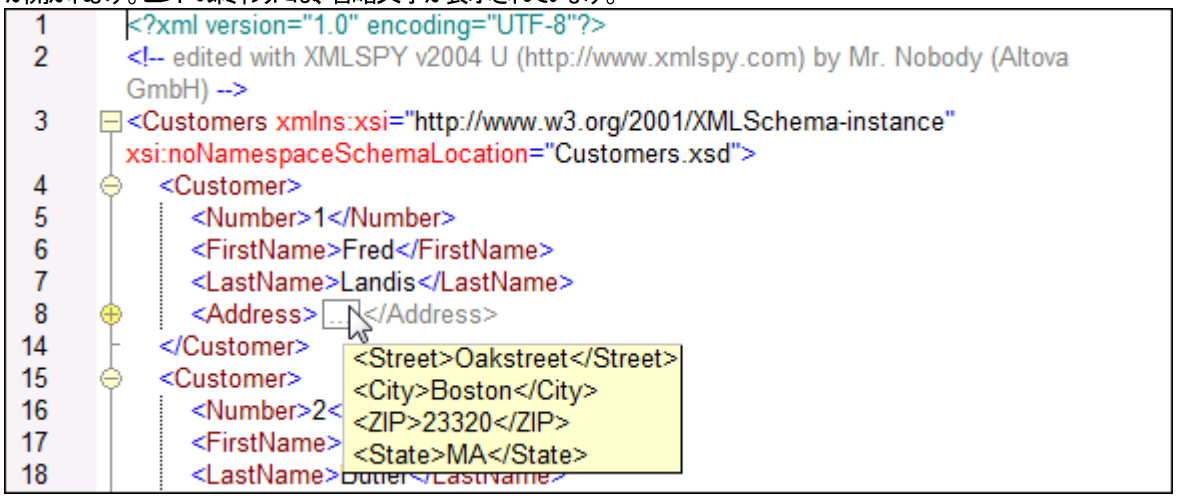
前のブックマークへ移動 (Shift + F2)

全てのブックマークを削除 (Ctrl + Shift + F2)

上記の「出力」メニュー内で使用することができます。「出力」(または、「XSLT」または「XQuery」) ペインを右クリックすると、コンテキストメニューを使用してブックマークコマンドを使用することができます。

ソースの折りたたみ

ソースの折りたたみは、ノードの展開と折りたたみ、およびソースの折りたたみマージン内に表示されるノードを表しています。テキストビュー設定ダイアログボックス内でマージンを切り替えることができます。展開するには、またはテキストの一部を折りたたむには、ウィンドウの左側の「+」と「-」ノードをクリックします。折りたたまれたコードは、省略記号と共に表示されます。これにより、プレビューされているコードを表示するヒントが開かれます。ヒントの終わりには、省略文字が表示されています。



インデントのガイド

インデントのガイドは、ラインのインデントを示す垂直な点線です。「テキストビュー設定」ダイアログボックス内で切り替えることができます。

メモ　「挿入タブ」と「スペースの挿入」オプションは、「出力 | XML テキストの整形出力」オプションを使用すると、効果が与えられます。

行末のマーカーと空白文字のマーカー

行末 (EOL) マーカーと空白文字マーカーは、テキストビュー設定 ダイアログボックス内で切り替えることができます。下のイメージは、行末と空白文字のマーカーが表示されている状態を表示しています。矢印は、タブ文字を表し、「CR」は改行を表し、点は空白文字を表しています。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <books xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="books.xsd">
3      <book id="1">
4          <author>Mark Twain</author>
5          <title>The Adventures of Tom Sawyer</title>
6          <category>Fiction</category>
7          <year>1876</year>
8      </book>
9  </books>
```

ズームインとズームアウト

「**Ctrl** キーを長く押して、スクロール(マウスのホイールをスクロール)することによりズームイン、とズームアウトを行うことができます。または、「**Ctrl** キーを押しながら「-」、または「+」キーを押してもズームイン、および、ズームアウトすることができます。

整形出力

「**XML テキストの整形出力**」コマンドは、アクティブな XML ドキュメントを構成された形式で表示するために、テキストビュー内で再フォーマットします。デフォルトでは、各子ノードが親ノードより4つの空白文字によりオフセットで表示されます。これは、「**テキストビュー設定**」ダイアログボックスによりカスタム化することができます。

XML ドキュメントを整形出力するには、「**出力 | XML テキストの整形出力**」メニューコマンドを選択、または「**整形出力**」  ツールバーボタンをクリックします。

ワード ラップ

現在アクティブなドキュメント内で、ワード ラップを切り替えるには、「**出力 | ワードラップ**」メニューコマンドを選択、または「**ワードラップ**」  ツールバーボタンをクリックします。

テキストのハイライト

テキストを選択すると、ドキュメント内で選択されたテキストに一致する箇所が自動的にハイライトされます。選択箇所は薄い青でハイライトされ、一致する箇所は薄いオレンジでハイライトされます。選択箇所と、一致する箇所は、灰色のマーカーによりスクロールバー上に表示されます。現在のカーソルの位置は、スクロールバー上で青いマーカーにより表示されています。

テキストのハイライトをオンにするにはテキストビュー設定 ダイアログボックス内の「**自動ハイライトの有効化**」を選択します。選択範囲は、文字全体、または固定された文字数に設定することができます。大文字と小文字の区別も指定することができます。

文字の選択範囲は、選択範囲の文字の最初の文字から開始する、一致する文字の最小数を指定することができます。例えば、2つ、または2文字以上の文字が一致するように選択することができます。この場合、1つの文字の選択は、一致とは考えられません。2文字以上の一致のみが一致と考えられます。ですから、この場合、`t` を選択すると、一致は表示されません。`ty` を選択すると、`ty` に一致するが表示されます。`typ` を選択すると、`typ` に一致するすべてが表示されます。

文字の検索に関しては、文字を区切るために以下が考慮されます: (角かっこのない)要素名、要素タグの角かっこ、属性名、引用符無しの属性の値。

3.1.8 テキストビュー内の検索

「出力」ペインと「XSLT」ペイン内のテキストは、広範囲のオプションの組み合わせと視覚的な補助を用いて、検索をすることができます。

検索を開始するには、「Ctrl+F」を押します(または、メニューコマンド「**編集 | 検索**」を選択します)。ダイアログ内に入力された検索用語を、ドキュメント全体で、または、選択された範囲で検索することができます。

- 検索する文字列を入力、または、コンボボックスを使用して最近使用した10件の文字列を選択します。
- 検索する文字列を入力、または、選択すると、全ての一致がハイライトされ、スクロールバー内で一致がベージュ色で表示されます。
- 現在選択されている一致は、他の一致と異なる色で表示されます。一致の位置はスクロールバー上で濃い青のカーソルマーカーで表示されます。
- 検索用語フィールド内に一致の総数が、現在選択されている一致のインデックス位置と共に表示されます。例えば、`2 の 4` は、4つ中の2番目の一致が選択されていることを示しています。
- 右下の**前へ**  **(Shift+F3)** と**次へ**  **(F3)** ボタンを選択することにより、1つの一致から次の一致へ両方向に移動することができます。

- 検索ダイアログを閉じるには、右上の**閉じる**  ボタンをクリック、または、**Esc** を押します。

以下の点に注意してください:

- 検索ダイアログに*モードは存在しません*。これは、検索ダイアログは、テキストビューを使用する場合でも、開き続けることができます。
- ダイアログボックスを開く前にテキストが選択されている場合、選択されたテキストは、自動的に検索用語フィールドに挿入されます。
- 選択範囲内で検索を行うには、以下を行います: (i) 選択範囲をマークします (ii) *選択範囲内で検索*  オプションをオンにして、検索範囲をロックします。(iii) 検索用語を入力します。他の選択範囲内を検索するには、現在の選択範囲を、*選択範囲内を検索*  オプションをオフにしてアンロックします。そして、新規の選択範囲を*選択範囲内を検索*  オプションをオンにして切り替えます。

- 検索ダイアログが閉じられると、現在の検索は **F3** を押すことにより順方向検索し、**Shift+F3** を押すことにより逆方向検索します。検索ダイアログが表示されます。

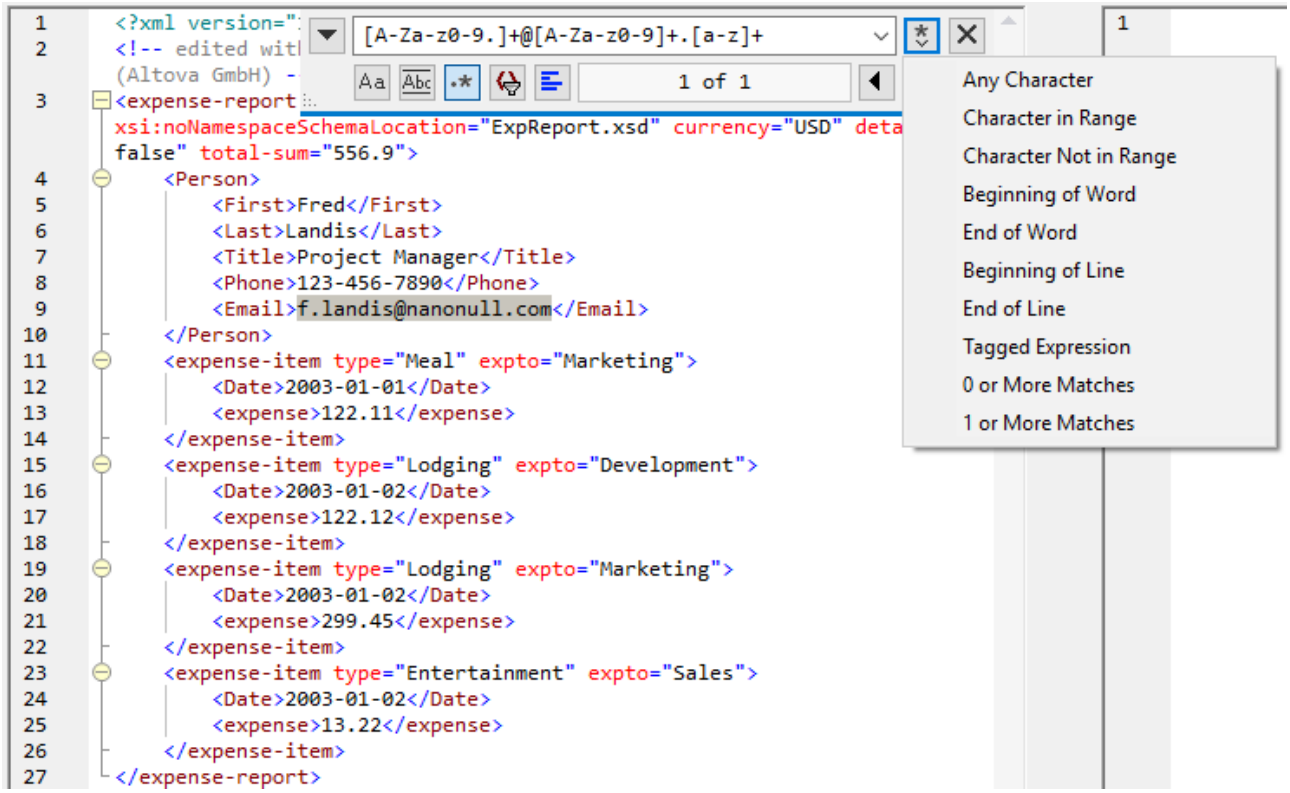
検索オプション

検索の条件は、検索フィールドの下にあるボタンにより指定することができます。オプションがオンになっている場合、ボタンの色は青に変更されます。以下のオプションから選択することができます:

オプション	アイコン	説明
大文字と小文字を区別する		切り替えられると、大文字と小文字を区別する検索が行われます（「Address」は「address」とは異なります）。
単語単位で検索		テキスト内の文字のみが一致されます。例えば、入力文字列 *fit* に対して、**単語単位を一致する** がオンにされていると、単語 *fit* のみが検索文字列に一致します。*fitness* 内の *fit* は一致しません。
正規表現		オンに切り替えられると、検索用語は、正規表現として読み取られます。「正規表現の使用」を参照してください。
アンカーの検索		検索用語が入力されると、ドキュメント内の一致がハイライトされ、一致の内の1つが現在の選択としてマークされます。**アンカーの検索** の切り替えは、最初の選択がカーソルの位置に対して相対的かを決定します。**アンカーの検索** がオンに切り替えられると、現在選択されている一致が選択され、現在のカーソルの場所の次の一致に一致します。**アンカーの検索** がオフに切り替えられていると、現在選択されている一致はドキュメントの最初から数えて最初の一致が一致します。
選択範囲内の検索		オンに切り替えられると、現在のテキストの選択範囲をロックし検索を選択されている範囲に制限します。それ以外の場合、ドキュメント全体が検索されます。テキストの新しい範囲を選択する前に、選択範囲内の検索 オプションをオフに切り替えて現在の選択範囲のロックを解除します。

正規表現の使用

正規表現（regex）を使用して、テキスト文字列を検索することができます。これを行うには、最初に *正規表現*  オプションをオンに切り替えます。これは検索用語フィールド内のテキストが正規表現として評価されるように指定します。次に正規表現と検索フィールドに入力します。正規表現の作成をヘルプするために、検索用語フィールドの右にある **正規表現 ビルダー**  ボタンをクリックします。これにより、検索用語フィールド内のテキストが正規表現として評価されます。下のスクリーンショットは、電子メールアドレスを検索するための簡単な正規表現を表示しています。



正規表現メタ文字のカスタムセットは、テキストを検索し置き換える際にサポートされます。

.	任意の文字を一致する。これは単一の文字のプレースホルダーです。
(abc)	(and) メタ文字は、タグ付けされた式の開始と終了をマークします。一致する箇所を、後で参照する(バックレファレンス)目的のためにタグ(「記録」) する場合、役に立つ可能性があります。9個までのサブ式をタグ付け(そして後から参照)することができます 例えば、 **(the) \1** は、文字列 the the に一致します。この式は、以下のように説明することができます: 以前に一致したタグ付けされた箇所が後に続き、スペース文字が後に続く、文字列「the」を一致、(および、タグされた箇所として記録)します。
\n	n が 1 から 9 の箇所では、n は、最初から9番目のタグ付けされた箇所を指します(上を参照してください)。
\<	単語の先頭に一致。
\>	単語の末尾に一致。
\	バックスラッシュが後に置かれている文字をエスケープします。すなわち、式 ¥x は、文字 x を文字通り使用することができます。例えば、 \[は、文字のセットの開始としてではなく、 [として解釈されます。
[...]	このセット内の文字に一致します。例えば、 **[abc]** は、a, b または c の文字に一致します。範囲を使用することができます: 例えば、小文字のために **[a-z]** を使用します。
[^...]	このセット内では内文字に一致します。例えば、 **[^A-Za-z]** は、アルファベット文字以外の文字に一致します。
^	(上記のとおり)セット内で使用されていない限り)行頭 に一致します。

$	行末に一致します。例えば、 **A+$** は、行末の A に一致します。
*	前の式のゼロ、または複数の発生に一致します。例えば、 **Sa*m** Sm、Sam、Saam、Saaam などに一致します。
+	前の式の1つの、または複数の発生に一致します。例えば、 **Sa+m** は Sam, Saam, Saaam などに一致します。

特別文字の検索

3.1.9 XSLT コードのプレビュー

XSLT 1.0、XSLT 2.0、または XSLT 3.0 をデータ変換言語変換言語として選択している場合、MapForce により生成された XSLT コードをプレビューすることができます。

XSLT コードをプレビューするには、マッピングウィンドウの下の **XSLT** タブ（または **XSLT2** または **XSLT3**）タブを必要に応じてクリックします。

メモ XSLT 1.0、XSLT 2.0, または XSLT 3.0 変換言語として選択すると使用することができます。

3.1.10 XSLT コードの生成

マッピングは対応する言語のためにデザインされていると仮定して、マッピングから XSLT 1.0、XSLT 2.0、または XSLT 3.0 コードを生成することができます。変換言語の選択も参照してください。

XSLT コードを生成する:

1. メニューアイテム「**ファイル | コードの生成 | XSLT 1.0**（または **XSLT 2.0**、**XSLT 3.0**）を選択します。
2. 生成された XSLT ファイルを保存するフォルダーを選択し、「**OK**」をクリックします。 MapForce は、コードを生成し、メッセージウィンドウ内にオペレーションの結果を表示します。

生成された xslt ファイルの名前には、 **<A>MapTo<B>.xslt** の書式を持ちます:

- "<A>" はマッピング設定内の **アプリケーション名** フィールド内の値です。
- "<B>" は、ターゲットマッピングコンポーネントの名前です。この値を変更するには、ターゲットコンポーネントの設定を開き、**コンポーネント名** フィールドの値を変更してください（次を参照してください: コンポーネント設定の変更）。

生成されたコード内のライブラリパスも参照してください。

RaptorXML Server を使用して自動化する

XSLT コードの生成後、.xslt ファイルを持つ同じディレクトリ内に **DoTransform.bat** という名前のバッチファイルが生成されます。RaptorXML Server を使用して **DoTransform.bat** を実行することができます。RaptorXML Server を使用した自動化も参照してください。

RaptorXML Server を使用して変換を行う:

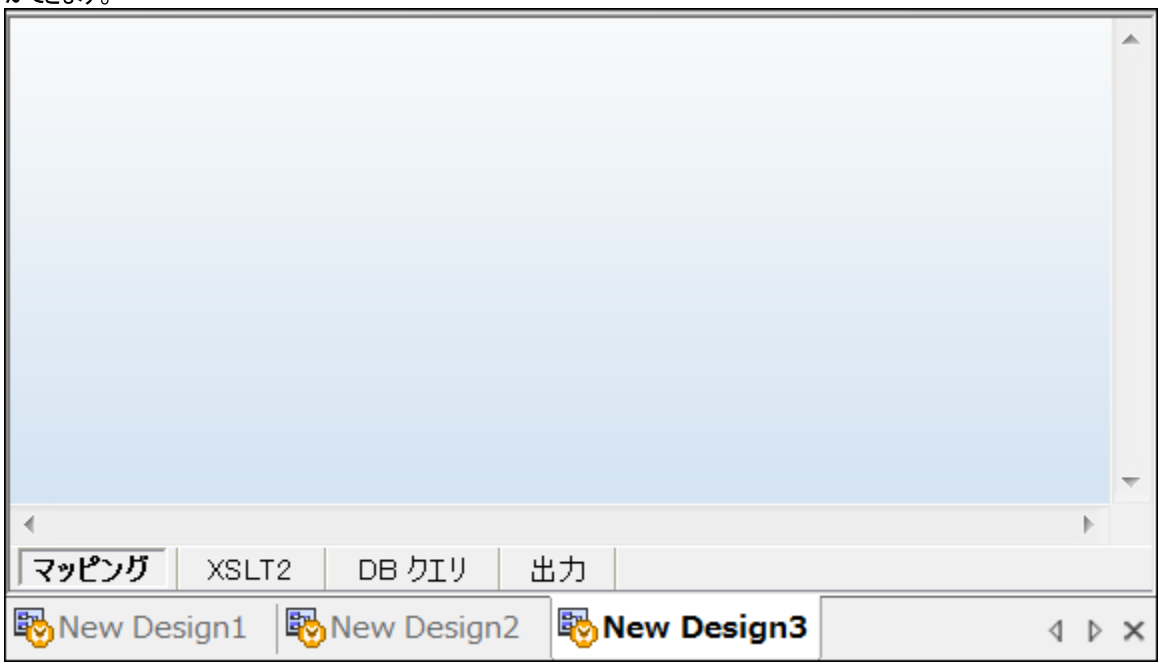
1. RaptorXML をダウンロードページからダウンロードしインストールします（https://www.altova.com/ja/download。
2. 上の操作で指定した出力フォルダーに入りされている **DoTransform.bat** バッチファイルを起動します。

RaptorXML がインストールされた場所をオペレーティングシステムの **path** 環境変数に追加する必要がある場合があります。RaptorXML ドキュメントを確認することができます（https://www.altova.com/ja/documentation）。

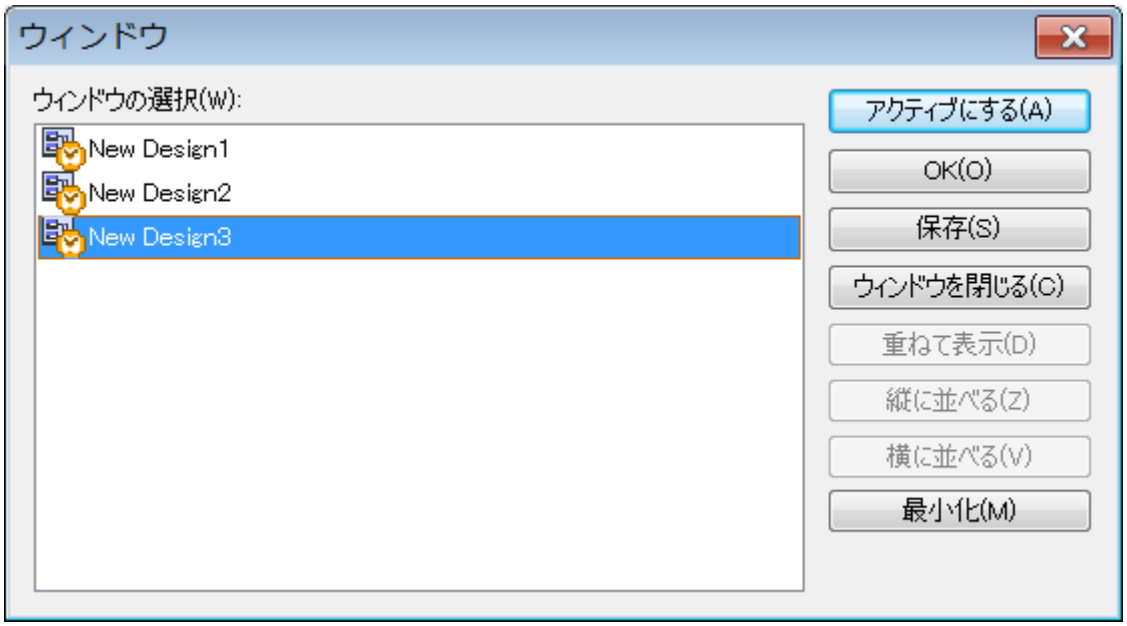
3.1.11　複数のマッピングウィンドウとの作業

MapForce は、複数のドキュメントインターフェイス（Multiple Document Interface）（MDI） を使用します。MapForce で開く各マッピングファイルには個別のウィンドウがあります。これにより、複数のマッピングウィンドウと作業、または並べ替えやサイズ調整を MapForce メイン（親） ウィンドウで行うことができます。で行うことができます。全ての開かれているウィンドウを標準ウィンドウのレイアウトを使用して以下のように並べ替えることができます: 上下に並べる、左右に並べる、重ねて表示。

複数のマッピングが MapForce 内で開かれている場合、マッピングペインの下の部分に表示されているタブを使用して素早く切り替えることができます。



ウィンドウ管理オプションは、「ウィンドウ」 メニューと「ウィンドウ」 ダイアログボックスで使用することができます。「ウィンドウ」 ダイアログボックスから、アクションを実行、または現在開かれているマッピングウィンドウで（保存、閉じる、または最小化を含む） アクションを使用することができます。



ウィンドウダイアログボックス

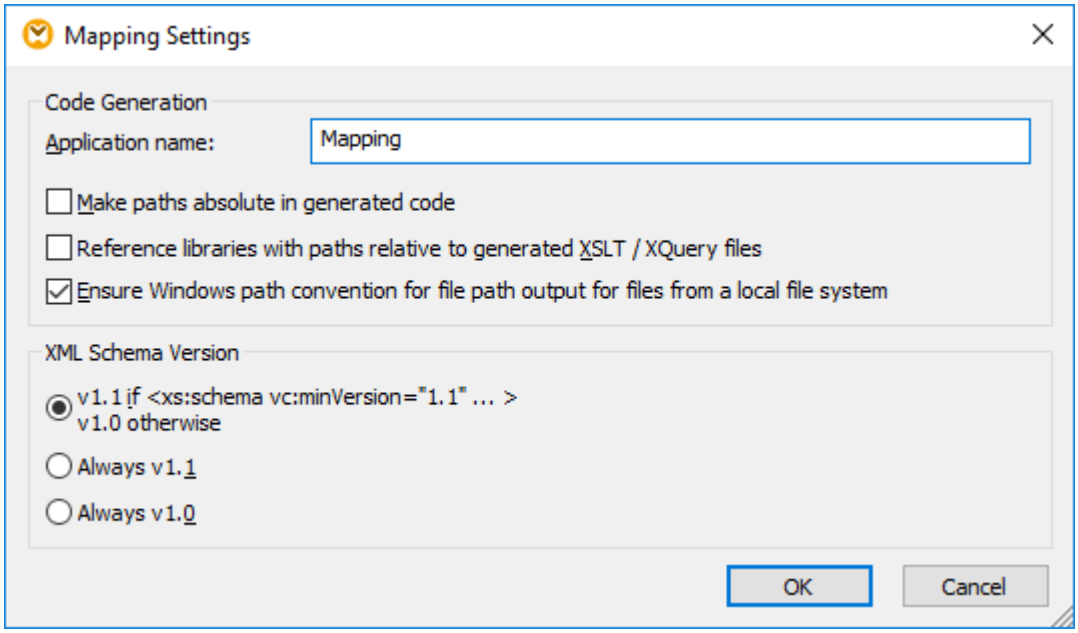
「**ウィンドウ**| **ウィンドウ**」を使用して、ウィンドウダイアログボックスを開くことができます。ウィンドウダイアログボックスから複数のウィンドウを選択するには、必要なエントリを「Ctrl」キーを押しながらクリックしてください。

3.1.12 マッピング設定の変更

マッピング設定ダイアログボックスから現在アクティブなマッピングデザインファイルのドキュメント特有の設定を変更することができます。この情報は*.mfd ファイルに保管されます。

マッピング設定ダイアログボックスの開きかた

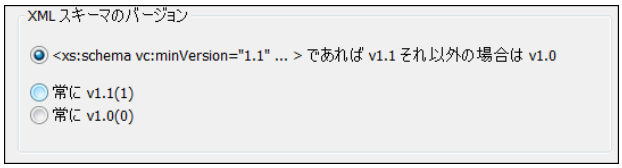
- 「**ファイル**」メニューから「**マッピング設定**」をクリックします。



マッピング設定ダイアログボックス

使用することのできる設定は以下の通りです。

アプリケーション名	生成されたXSLT ファイルを定義します。
生成されたコード内でパスを絶対パスにする	このチェックボックスは（XSLT ライブラリなどの）外部ライブラリファイルへのパスを除き、マッピングコンポーネント内のすべてのパスに影響を与えます。 チェックボックスは生成されたプログラムコード内でファイルパスが相対パス化または絶対パスかを定義します。多種の実行環境内でのパスも参照してください。以下も参照多種の実行環境内のパス
生成されたXSLT／XQuery ファイルに相対的なパスを持つレファレンスライブラリ	マッピング言語がXSLT またはXQuery* の場合このチェックボックスを適用することができます。 XSLT またはXQuery ライブラリをマッピング参照する場合、およびマッピングからXSLT またはXQuery ファイルを生成する場合このオプションは通常役に立ちます。ライブラリパスを生成されたXSLT またはXQuery コードのディレクトリに対して相対的にする場合、チェックボックスを選択します。 チェックボックスが選択されていない場合、ライブラリパスは生成されたXSLT またはXQuery コード内で絶対パスになります。生成されたコード内のライブラリパスも参照してください。
Windows パスがファイルパスの規則であることを確認	マッピング言語がXSLT 2.0、XSLT 3.0、またはXQuery* 2.0の場合このチェックボックスは適用可能です。 チェックボックスはWindows のパス規則が使用されていることを確認します。XSLT 2.0、XSLT 3.0、またはXQuery を生成する

	場合、現在処理されているファイル名は、ローカルファイルのためにフォーム`file://URI` 内のパスを返す **`document-uri`** 関数を使用して取得されます。 このチェックボックスがアクティブな場合、`file://URI` パス仕様は更なる処理のために自動的に完全な W indows ファイルパス（例、"C:¥..."）に変換されます。
XML スキーマバージョン	マッピングファイルで使用された XML スキーマバージョンを定義することができます。バージョン 1.0 または 1.1 に準拠するスキーマをロードするかを定義します。すべてのバージョン 1.1 特有の機能は現在サポートされています。 *xs:schema vc:minVersion="1.1"* 宣言が存在する場合、バージョン 1.1 が使用されます。それ以外の場合、バージョン 1.0 が使用されます。  XSD ドキュメントが`vc:minVersion` 属性 または `1.0` または `1.1` 以外の`vc:minVersion` 属性の値を持たない場合、XSD 1.0 がデフォルトのモードです。 **メモ** **vc:minVersion** 属性と **xsd:version** 属性を混同しないでください。前者が XSD バージョン番号を持つ場合、後者はドキュメントのバージョン番号を持ちます。 既存のマッピング内のこの設定を変更することは、選択されたスキーマバージョンの全ての XML スキーマを再ロードすることを意味し、妥当性を変更する可能性があります。

＊MapForce Professional または Enterprise Edition を必要とします。

3.2 コンポーネントとの作業

コンポーネントは、MapForce 内のマッピングデザインで中心的な要素です。全般的に、「コンポーネント」という用語は、データソースとデータターゲットとしての役割を果たすオブジェクト、または中間処理の段階にあるマッピング内のデータに簡単な名称を与えるために役に立ちます。

コンポーネントには以下の2つの主なカテゴリがあります: 構造コンポーネントと変換コンポーネント。

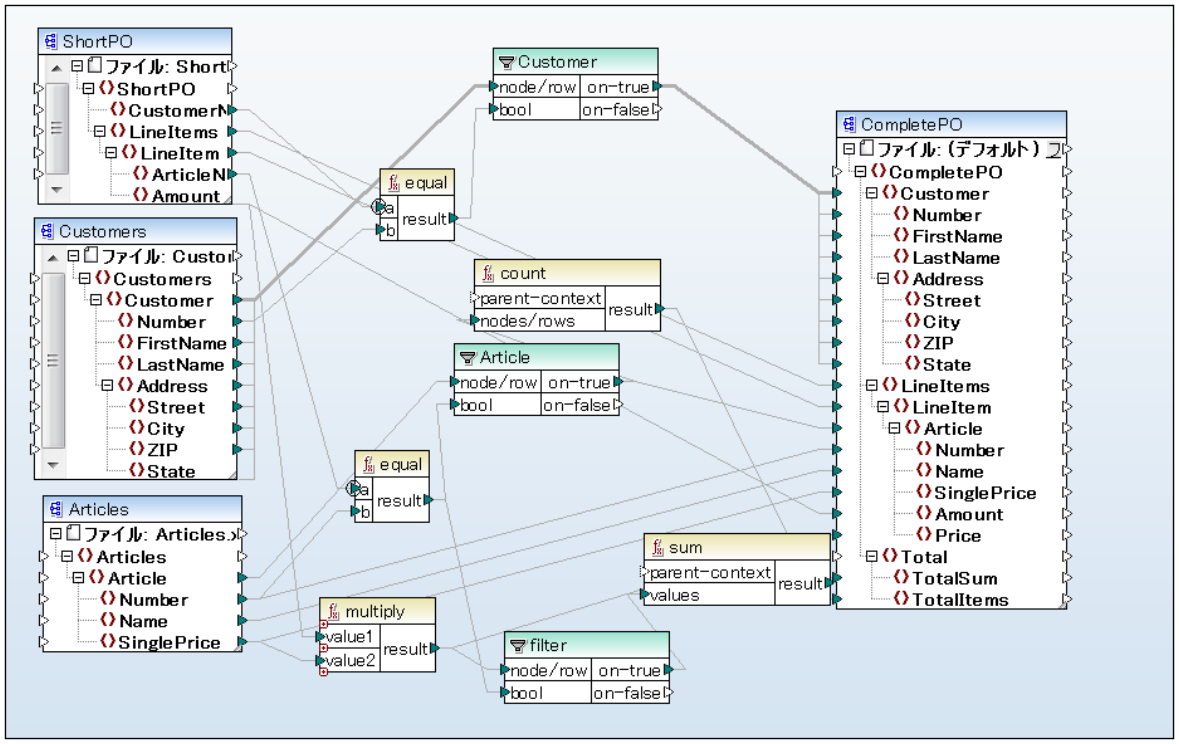
構造コンポーネントは、データの抽象的な構造とスキーマを表します。例: (メニューコマンド「**挿入 | XML スキーマ/ファイル**」を使用して) マッピングエリアにXML ファイルを追加すると、マッピングコンポーネントになります。構造コンポーネントとその特化についての詳細は、*データソースとターゲット*を参照してください。一部の例外を除いて、構造コンポーネントは、*アイテム*と*シーケンス*から構成されます。アイテムはマッピングユニットの最も低いレベルです (例: XML ファイル内の単一の属性、または単純型の要素)。*シーケンス*は*アイテム*のコレクションです。

変換コンポーネントは、データを変換 (例: 関数)、または変換の手助け (例: 定数または変数) をします。これらのコンポーネントを使用して、複数のデータ変換タスクを達成する方法に関しては、*マッピングのデザイン*を参照してください。

構造コンポーネントを使用することにより、ファイルまたはその他の*ソース*からデータを読み込み、また、ファイルまたはその他の*ソース*へデータを書き込み、マッピング処理内の中間の段階で (例: プレビューするために) データを保存することができます。この結果、構造コンポーネントには、以下の3つの型があります:

- *ソース*。マッピングエリアの左側に配置し、MapForce にデータを読み込むように命令し、コンポーネントを*ソース*として宣言します。
- *ターゲット*。マッピングエリアの右側に配置し、MapForce にデータを書き込むように命令し、コンポーネントを*ソース*として宣言します。
- *パススルー*。これは特別なコンポーネントの型で*ソース*と*ターゲット*としての役割を果たします。(更に詳しい情報に関しては、*チェーンマッピング/パススルーコンポーネント*を参照)。

マッピングエリアでは、コンポーネントは三角形で表示されます。次のサンプルマッピングは、3 つの*ソース*コンポーネント、1つのXML コンポーネント、*ソース*に書き込まれる前にデータが通過する複数の変換コンポーネント (関数とフィルター) を表しています。



CompletePO.mfd

このマッピングサンプルは以下のパスで見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\CompletePO.mfd**。

3.2.1 コンポーネント内の検索

特定のノード/アイテムをコンポーネント内で検索する方法:

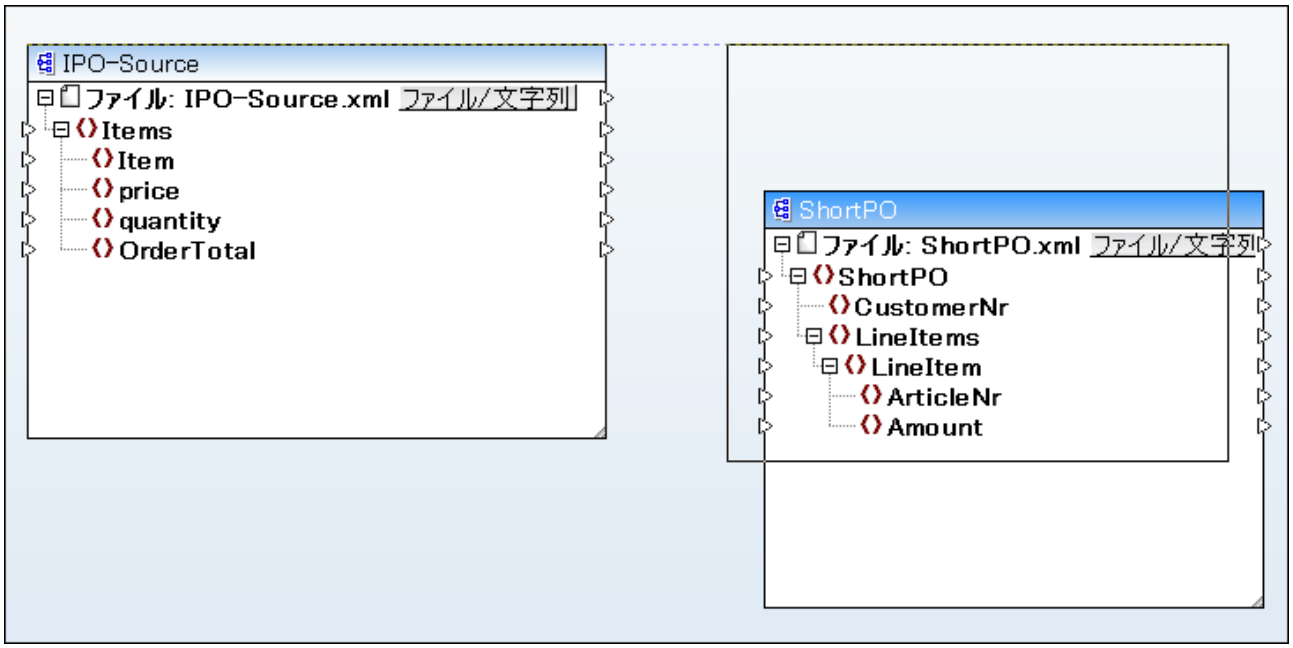
1. 検索するコンポーネントをクリックします。「CTRL+F」キーをクリックします。
2. 検索用語を入力して、「**次を検索**」をクリックします。

高度なオプションを使用して、どのアイテム（ノード）が検索されるか、また特定の接続をベースにした厳密な検索オプションを定義します。

3.2.2　コンポーネントの整列

コンポーネントをマッピングペイン内でドラッグすると、MapForce は、自動配置 ガイドラインを表示します。このガイドラインは、マッピングウィンドウ内で他のコンポーネントと位置を合わせる際とても役に立ちます。

下のサンプルマッピングでは、下のコンポーネントを移動します。ガイドラインは、マッピングの左のコンポーネントに位置を合わせることができます。



コンポーネント 自動配置 ガイドライン

このオプションを有効化または無効化する:

1. 「**ツール**」メニューから「**オプション**」をクリックします。
2. 「**編集**」グループから、「**マウスのドラッグにてコンポーネントを整列**」のチェックボックスをチェックします。

3.2.3 コンポーネント設定を変更する

コンポーネントをマッピングエリアに追加した後、から適用することのできる設定をコンポーネント設定ダイアログボックスから構成します。コンポーネント 設定 ダイアログボックスを以下の方法で開くことができます:

- コンポーネントを選択して、「**コンポーネント**」メニューから「**プロパティ**」をクリックします。
- コンポーネントヘッダーをダブルクリックします。
- コンポーネントヘッダーを右クリックして、「**プロパティ**」をクリックします。

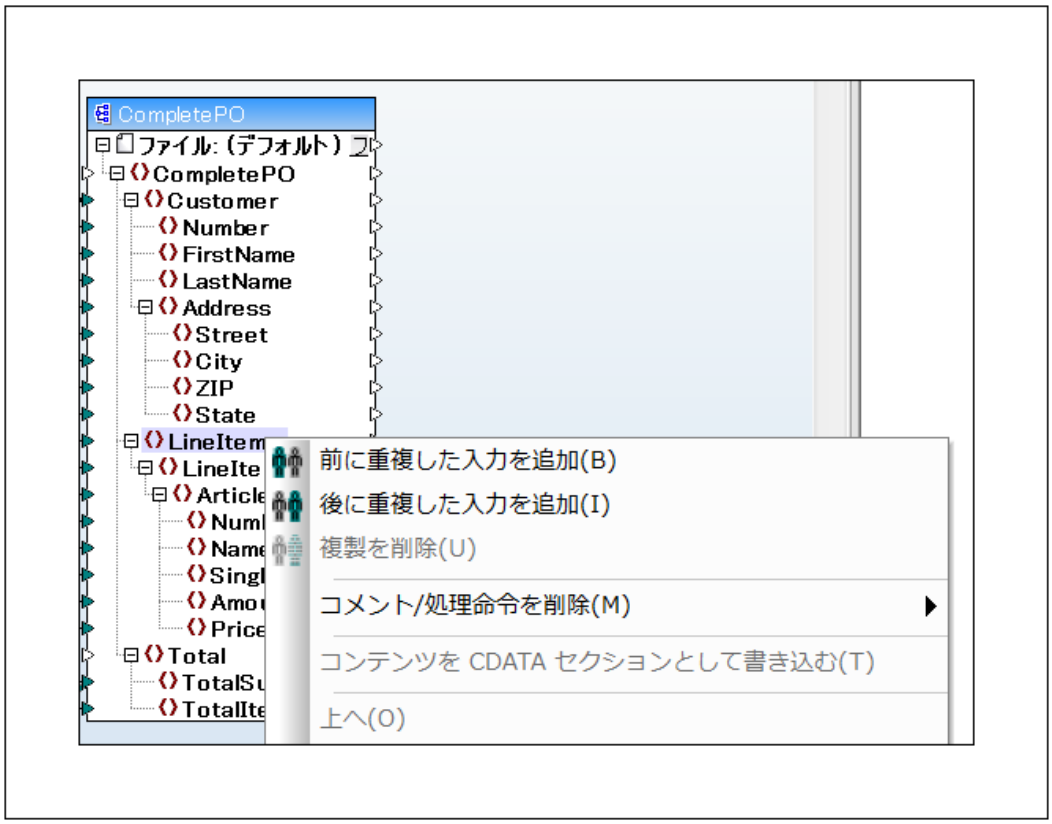
コンポーネント設定ダイアログボックスで使用することのできる設定に関しては XML コンポーネント設定を参照してください。

XML などの、ファイルベースのコンポーネント、「**ファイル**」(File) ボタンがルートノードの横に表示されます。このボタは、単一のマッピング内で複数のファイルを処理または生成する際に適用することのできる高度なオプションを指定します (複数の入力または出力ファイルを動的に処理を参照)。

3.2.4 入力の複製

1つ以上のソースからデータを受け入れるためにコンポーネントを構成する必要がある場合があります。例えば、2つの異なるスキーマからのデータを単一のスキーマに変換する必要があるとします。目的のスキーマが2つのソーススキーマからのデータを受け入れるようにするには、コンポーネント内の入力アイテムを複製して行います。入力の複製は、ターゲットコンポーネントであるコンポーネントにとってのみ意味があります。与えられたターゲット コンポーネントは、必要に応じて、必要な数複製することができます。

特定の入力アイテムを複製するには、アイテムを右クリックして、コンテキストメニューから「後/前に入力を複製する」を選択します。



上のイメージでは、アイテム `LineItem` が2 番目のソースからマップを可能にするために複製されています。

入力を複製すると、元の入力と複製された入力の間をマップすることができます。例: これによりデータをソース A からもとの入力に、またデータをソース B から複製された入力にコピーすることができます。

メモ　XML インスタンスを無効にするため、XML 属性の複製は許可されていません。スキーマ内の要素の `maxOccurs` 属性の値に関わらず、XML 要素の場合、入力の複製は許可されています。この振る舞いは、スキーマが後に変更することが可能なため、また、ソースデータが任意のため、意図的です。 例えば、マッピングは、マッピング上で入力が複製されても、単一の XML 要素を生成することができます。

順序を追った例は、複数のソースから1 つのターゲットにマップを参照してください。

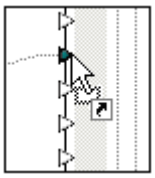
3.3 接続との作業

マッピングは、データを1つのフォーマットから他のフォーマットに変換することを意味します。基本的なマッピングシナリオでは、マッピングエリアにソースとターゲットデータを表すコンポーネントを追加します（例: ソースXMLスキーマと目的のスキーマ）、そして視覚的なマッピング接続を2つの構造間に描きます。ですから、接続はソースから目的地へどのようにデータがマップされたかを表す視覚的な表示といえるでしょう。

コンポーネントには、マッピングで小さい三角形として表示される入力と出力があり、これらはコネクタと呼ばれます。出力コネクタは接続を描くアイテムの右横に配置されます。

2つのアイテム間に接続線を引く:

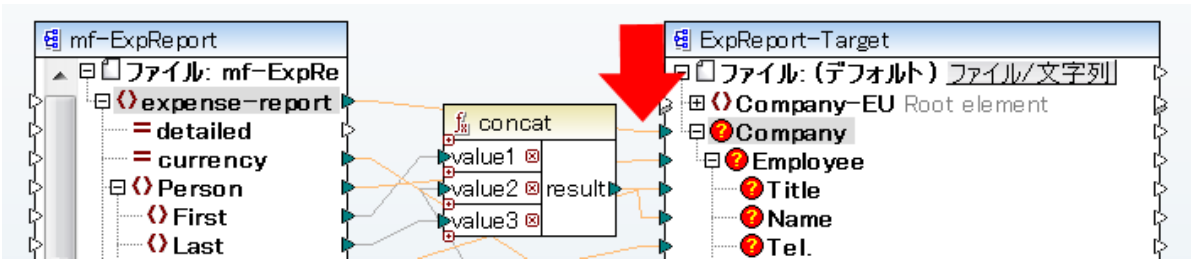
- ソースアイテムの出力コネクタをクリックし、目的アイテムまでドラッグします。ドロップアクションが許可されると、ヒントのリンクがカーソルの横に表示されます。



入力コネクタは入力される接続のみを受け入れます。同じ入力に第2番目の接続を追加使用とすると、接続を新規の接続と置き換えるか、または入力アイテムを複製するかを問うメッセージボックスが表示されます。出力コネクタは複数の接続を持つことができ、それぞれが異なった入力を持ちます。

異なったアイテムに接続を移動する:

- 接続のスタッブ（ターゲットに近い直線のセクション）をクリックして、移動先までドラッグします。



異なったアイテムへの接続をコピーする:

- 接続のスタッブ（ターゲットに近い直線のセクション）をクリックして「Ctrl」キーを押しながら移動先までドラッグします。

接続の対極にあるアイテムを表示する:

- マウスポインターを（入力/出力アイコンの近くにある）コネクタの直線部へマウスオーバーすることで、コネクタがハイライトされ、ポップアップが表示されます。ポップアップには、コネクタの対極にあるアイテムの名前が表示されます。同一の出力アイコンから複数の接続が定義されている場合には、最大10の名前が表示されます。スクリーンショットでは、**SinglePrice** と**value2** という複数のターゲットに対して接続が行われていることが理解できます

接続設定 を変更するには以下を行います:

- 「**接続**」メニューから「**プロパティ**」をクリックします。(このメニューアイテムは、接続を選択すると有効化されます)。
- 接続をダブルクリックします。
- 接続を右クリックして、「**プロパティ**」をクリックします。

接続設定を参照してください。

接続を削除するには、以下を行います:

- 接続をクリックして、「**削除**」キーを押します。
- 接続を右クリックして、「**削除**」をクリックします。

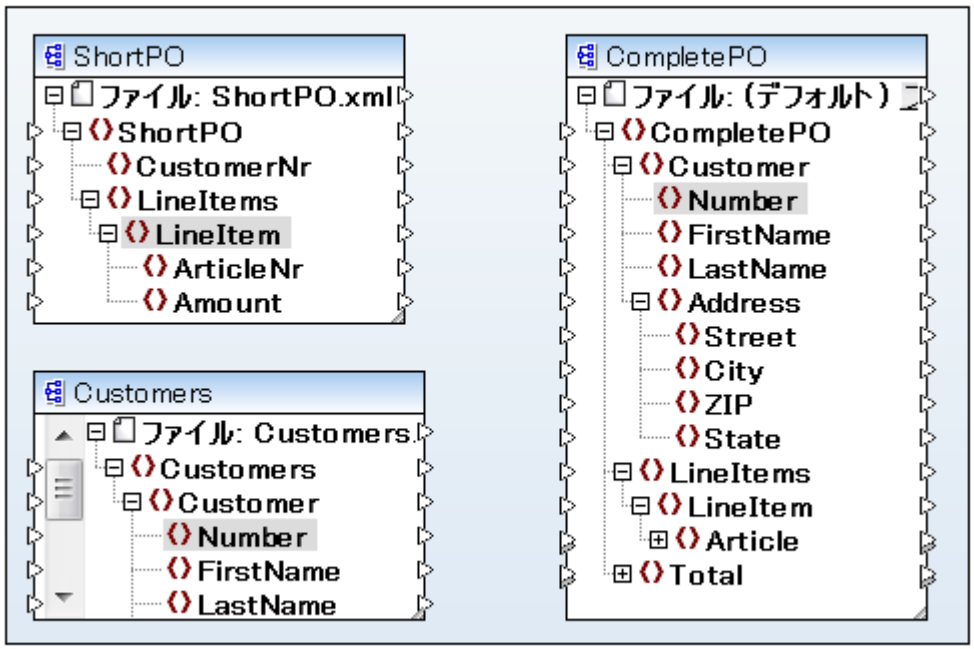
3.3.1 必須の入力について

マッピング処理内での手助けをするために、MapForce は、ターゲットコンポーネント内の必須の入力をオレンジ色でハイライトします:

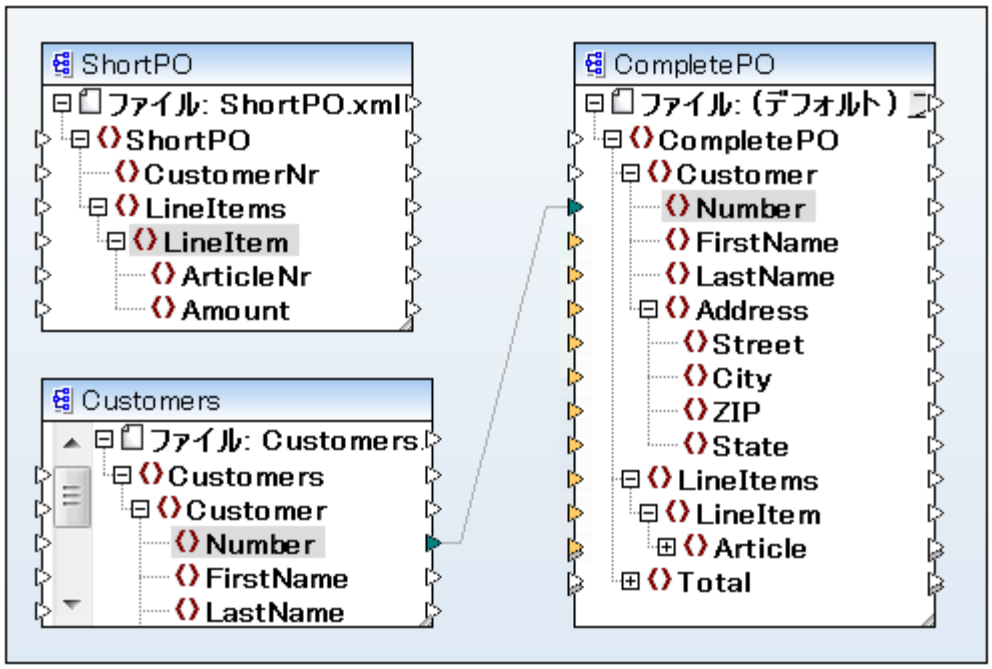
- XML とEDI コンポーネントは、minOccurs パラメーターが 1 と同じまたは 1 より大きいアイテムです。
- データベース内では、これらは「ゼロではない」として定義されるフィールドです。
- W SDL 呼び出しとW SDL レスポンス(全てのノード)
- 必須と定義されたXBRL ノード
- 関数内では、これらは、パラメーターがマップされると、接続の必要な他の必須のパラメーターがハイライトされる特定の必須のパラメーターです。例: フィルター入力パラメーターの 1 つがマップされると、他のパラメーターは自動的にハイライトされます。
- MS Excel シート内のワークシート名

例

…¥MapForceExamples フォルダー内にある、CompletePO.mfd などのマッピング作成の際、挿入されたXML スキーマファイルは下に表示されるとおりです。



Customers コンポーネントのNumber 要素はCompletePO コンポーネントのNumber 要素に接続されています。接続が作成されると、CompletePO コンポーネントの必要なアイテム/ノードは、ハイライトされます。折りたたまれた「Article」ノード/アイコンもハイライトされます。



3.3.2 優先する接続線の表示の変更

MapForce ではマッピングウィンドウに表示されるコネクタの表示方法を選択することができます

選択コンポーネントの接続線の表示 により以下のように表示が切り替わります:
- 全てのマッピングコンポーネントが黒色で表示されるか、
- 現在選択されているコンポーネントに関係したコネクタが黒色で表示されるとともに、その他の接続がグレーで表示されます。

ソースからターゲットへの接続線の表示により以下の表示が切り替わります:
- 現在選択されているコンポーネントに直接接続されているコネクタが黒色で表示されるか、
- 現在選択されているコンポーネントの入力または出力アイコンへリンクされているコネクタが黒色で表示されます。

3.3.3 接続の注釈

個別の接続をマッピングに詳細なコメントを与えラベル付けすることができます。このオプションは「**全ての接続の型**」で使用することができます。

接続に注釈をつける:

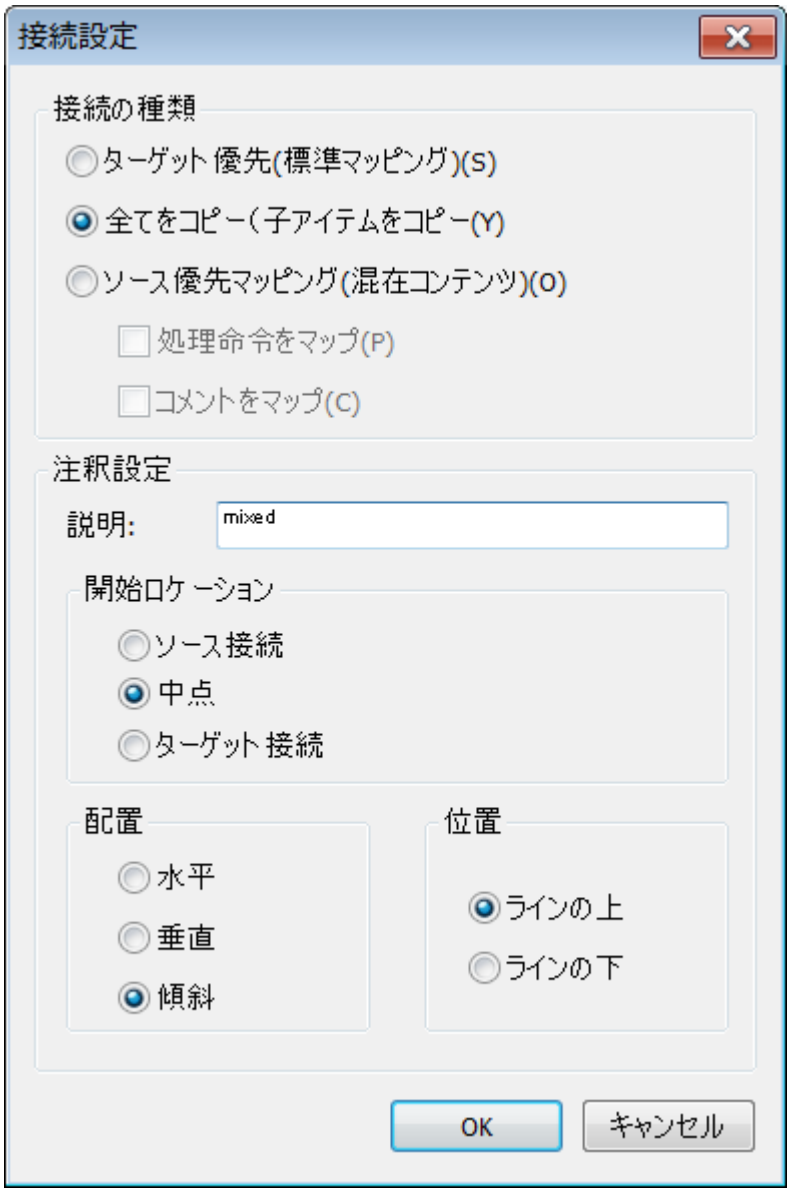
1. 接続を右クリックして、コンテキストメニューからを「**プロパティ**」選択します 、
2. 現在選択されている接続の名前 を「**詳細**」フィールドに入力します。これにより注釈設定グループ内の全てのオプションが有効化されます。

3. 残りのグループを使用して、ラベルの「開始の箇所」、「配置」、「位置」を定義します。
4. 注釈テキストを確認するには、表示オプションツールバー内の「注釈の表示」 アイコンを有効化します。

メモ 「注釈の表示」アイコンが無効化されている場合、接続にカーソルをポイントすると、注釈を表示することができます。「ヒントの表示」 ツールバーボタンが表示オプションツールバー内で有効化されている場合、注釈のテキストは吹き出し内に表示されます。

3.3.4 接続設定

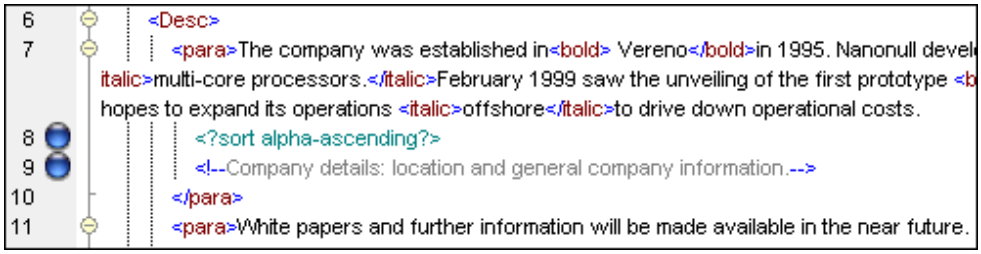
コネクタを右クリックしてコンテキストメニューからプロパティを選択するか、コネクタをダブルクリックして接続設定ダイアログボックスを開き、現在のコネクタに対して特別な(混合コンテンツ)設定を定義することができます。使用することのできないオプションは無効化されています。



接続設定ダイアログボックス

complexType のアイテムのために、以下のマッピングのための接続型を選択することができます(これらの設定は、テキストノードを持たない**complexType** アイテムにも適用することができます):

ターゲット優先(標準)	接続の型をターゲット優先に変更します(ターゲット優先 / 標準 マッピング を参照)。
全てコピー(子アイテムのコピー)	接続の種類を「全てコピー」に変更し、ソースとターゲットコンポーネント内の全ての等しいアイテムと自動的には接続します(全てコピー接続 を参照)。
ソース優先(複合コンテンツ)	接続の種類を「ソース優先」に変更し、またマップされる追加要素の選択を有効化します。マッピングに適するために、追加要素はXML ソースファイル内のマップされたアイテムの子アイテムである必要があります。 「**処理命令をマップ**」または「**コメントをマップ**」チェックボックスを有効化することにより、これらのデータグループを出力ファイルに含むことができます。



メモ CDATA セクションはテキストとして扱われます。

注釈設定グループにより接続へ注釈を与えることができます（ 接続の注釈 を参照）。

3.3.5 一致する子要素の接続

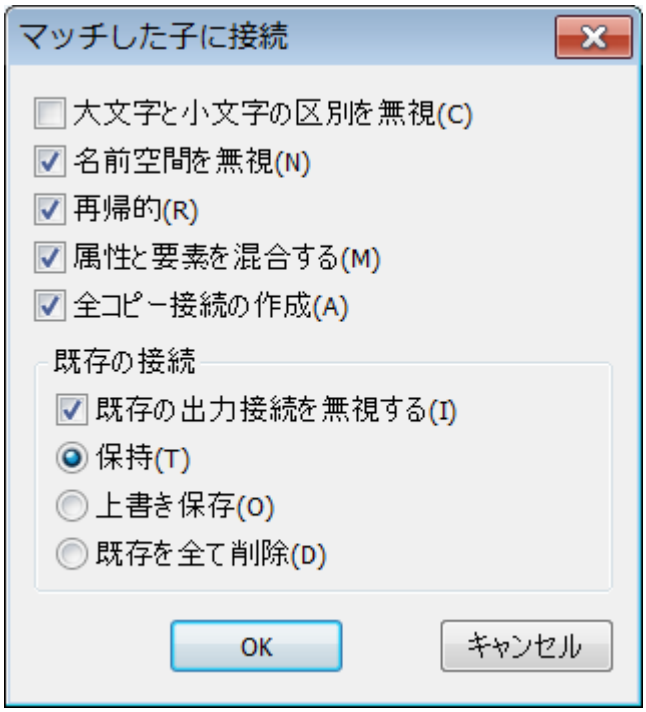
ソースとターゲットコンポーネントの両方内で同じ名前のアイテム間で複数の接続を作成することができます。「全てコピー」接続は、デフォルトで作成されます（ 全てコピー接続を参照）。

「マッチする子を自動的に接続する」オプションをオンまたはオフに切り替えるには、以下を行います:

- 「一致する子要素の自動接続」（ ）ツールバーボタンをクリックします。
- 「接続」メニューから「一致する子要素の自動接続」をクリックします。

「マッチする子を接続する」のための設定を変更する方法:

1. 両方のコンポーネント内の「子アイテム」という名前を共有する2つの（親）アイテムを接続します。
2. 接続を右クリックして、「一致する子要素の接続」オプションを選択します。



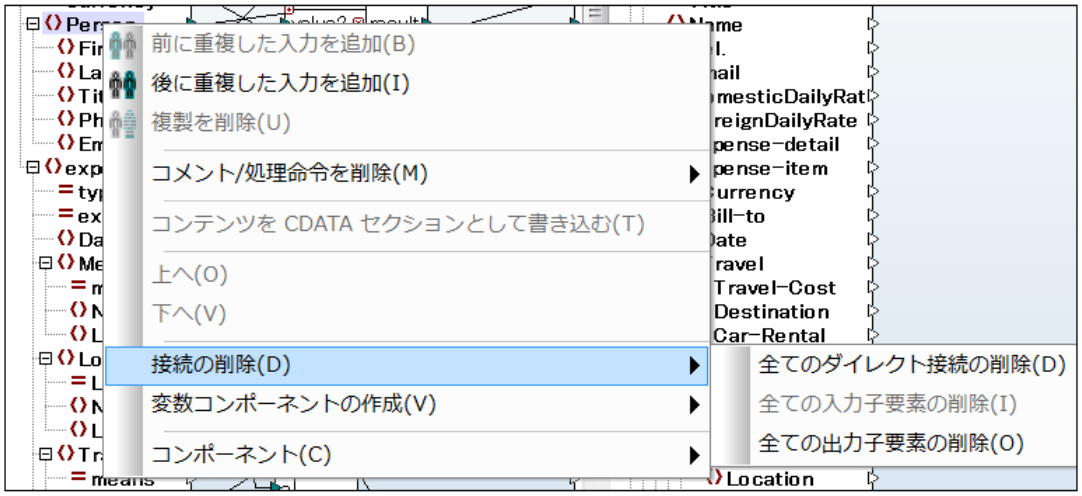
3. 必要なオプションを選択して（下のテーブルを参照してください）、「OK」をクリックします。ダイアログボックス内の設定定義に当てはまる、接続が同じ名前の全ての子アイテムのために作成されます。

メモ　「子の自動接続の切り替え」（ ）ツールバーボタンがアクティブな場合、ここで定義する設定が2つのアイテムを接続する際に適用されます。

大文字と小文字の区別を無視	子アイテム名の大文字と小文字の区別を無視します。
名前空間を無視	子アイテムの名前空間を無視します。
再帰的	一致するアイテム間の接続を再帰的に作成します。すなわち、階層構造内でどれほど深くアイテムがネストされていても、アイテムが同じ名前を持つ限り、アイテム間の接続が作成されます。
属性と要素を混合	有効化されている場合、同じ名前を持つ属性と要素の間の接続を許可します。例: 2つの「Name」 アイテムが存在する場合、1つが属性、もう1つが要素である場合でも、接続を作成します。
全コピー接続の作成	この設定は、デフォルトの設定では有効化されています。この設定は、「全てコピー」とソースとターゲット アイテム間の接続を（可能であれば）作成します。
既存の出力接続を無視	出力接続が既存の場合でも、追加の接続を一致するアイテムのために作成します。
保持	既存の接続を保持します。
上書きを保存	定義された設定に従い接続を再作成します。既存の接続は破棄されます。
既存を全て削除	新規接続を作成する前に、既存のすべての接続を削除します。

接続の削除

接続 マッチする子を接続するダイアログを使用して作成された接続、またはマッピング処理中の接続をグループ化して削除することができます。



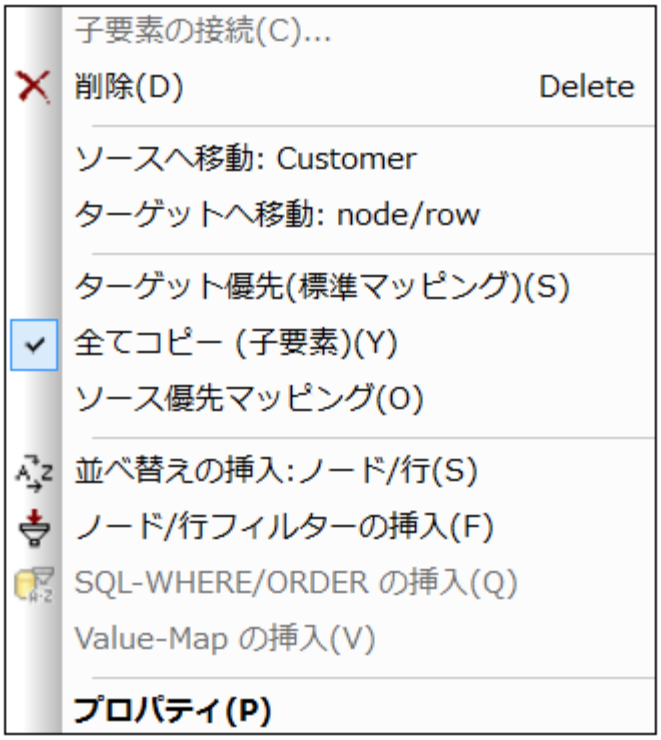
接続の削除:

1. コンポーネント内のアイテムの名前を右クリックします（このサンプルでは「Person」）。
2. 「**接続の削除 | 全ての..接続の削除**」を選択します。

全てのダイレクト接続を削除	直接マップされた、または現在のコンポーネントから他のソースまたはターゲットコンポーネントにマップされたすべての接続を削除します。
全ての入力子接続を削除	ターゲットコンポーネント内でアイテムを右クリックした場合のみアクティブになります。全ての入力子接続を削除します。
全ての出力子接続を削除	ソースコンポーネント内でアイテムを右クリックした場合のみアクティブになります。全ての出力子接続を削除します。

3.3.6 コンテキストメニューの接続

接続を右クリックすると、以下のコンテキストコマンドを使用することができます。



マッチする子を接続	「マッチする子を接続する」ダイアログボックスを開きます。(一致する子要素の接続 を参照)。接続がマッチする子アイテムを持つことができる場合、このコマンドを有効化することができます。
削除	選択された接続を削除します。
ソースへ移動: <item name>	現在の接続のソースコネクタを選択します。
ターゲットに移動: <item name>	現在の接続のターゲットコネクタを選択します。
ターゲット優先 (標準)	接続の型をターゲット優先に変更します(ターゲット優先接続 を参照)。
全てコピー (子アイテムのコピー)	接続の種類を「全てコピー」に変更し、ソースとターゲットコンポーネント内の全ての等しいアイテムと自動的に接続します(全てコピー接続 を参照)。 ソースアイテムとターゲットアイテムの両方が子アイテムを持つ場合、このコマンドは有効化されます。
ソース優先 (複合型コンテンツ)	接続の種類を「ソース優先」に変更します(ソース優先マッピング を参照)。

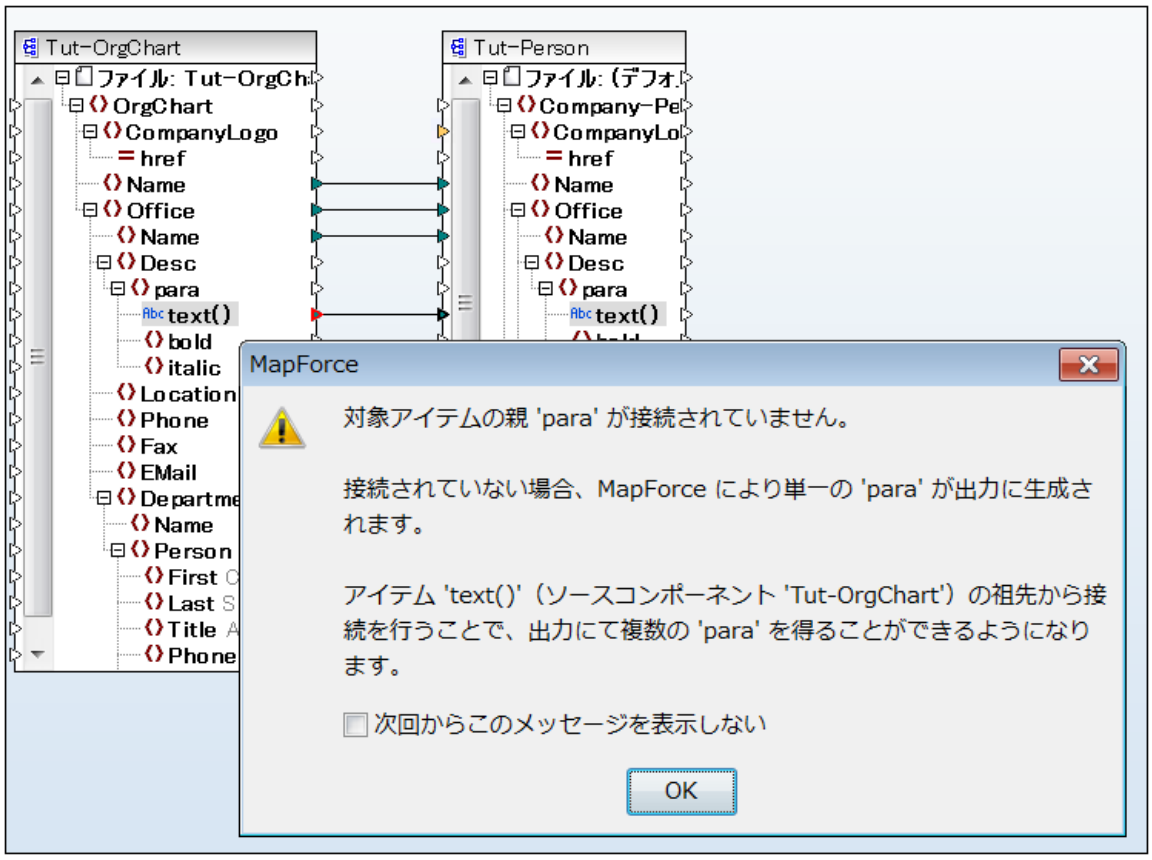
	ソースアイテムとターゲットアイテムの両方が子アイテムを持つ場合、このコマンドは有効化されます。
並べ替えの挿入: Nodes/Rows	ソースとターゲットアイテム間のソートコンポーネントを追加します。(データの置換えを参照)。
フィルターの挿入: Nodes/Rows	ソースとターゲットアイテム間のフィルターコンポーネントを追加します。(フィルターと条件を参照)。
Value-Map の挿入	ソースとターゲットアイテム間のValue-Map コンポーネントを追加します。(Value-Mapの使用を参照)。
プロパティ	「接続設定」ダイアログボックスを開きます。(接続設定 を参照)。

3.3.7 見つからない親接続の通知

ソースならびにターゲットアイテム間の接続を手動で作成する場合、起こりえるマッピング結果がMapForce により分析されます。子アイテム同士をマッピングした場合、ソースアイテムの親とターゲットアイテムの親も接続するように促すプロンプトが表示されます。

これはマッピングのプレビューを出力ウィンドウにて行う際に、子アイテムの表示を 1 つに限定しないためのものです。ソースノードから得られるものが単一の値ではなく、シーケンス形式の場合、通常親アイテムの接続を行う必要があります。

<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥ フォルダーに収められている **Tut-OrgChart.mfd** マッピングを以下に示します。ソースコンポーネントにある text() アイテムがターゲットコンポーネントにある text() アイテムへ接続されると、メッセージボックスが表示され、親アイテムの para が接続されておらず、出力では一度だけ生成される旨が通知されます。複数の para アイテムをターゲットにて表示するには、ソースならびにターゲットアイテムの para を接続する必要があります。



Tut-OrgChart.mfd (MapForce Basic Edition)

ターゲット内で複数の`para` アイテムを生成するには、ソースとターゲット `para` アイテムをそれぞれ接続します。

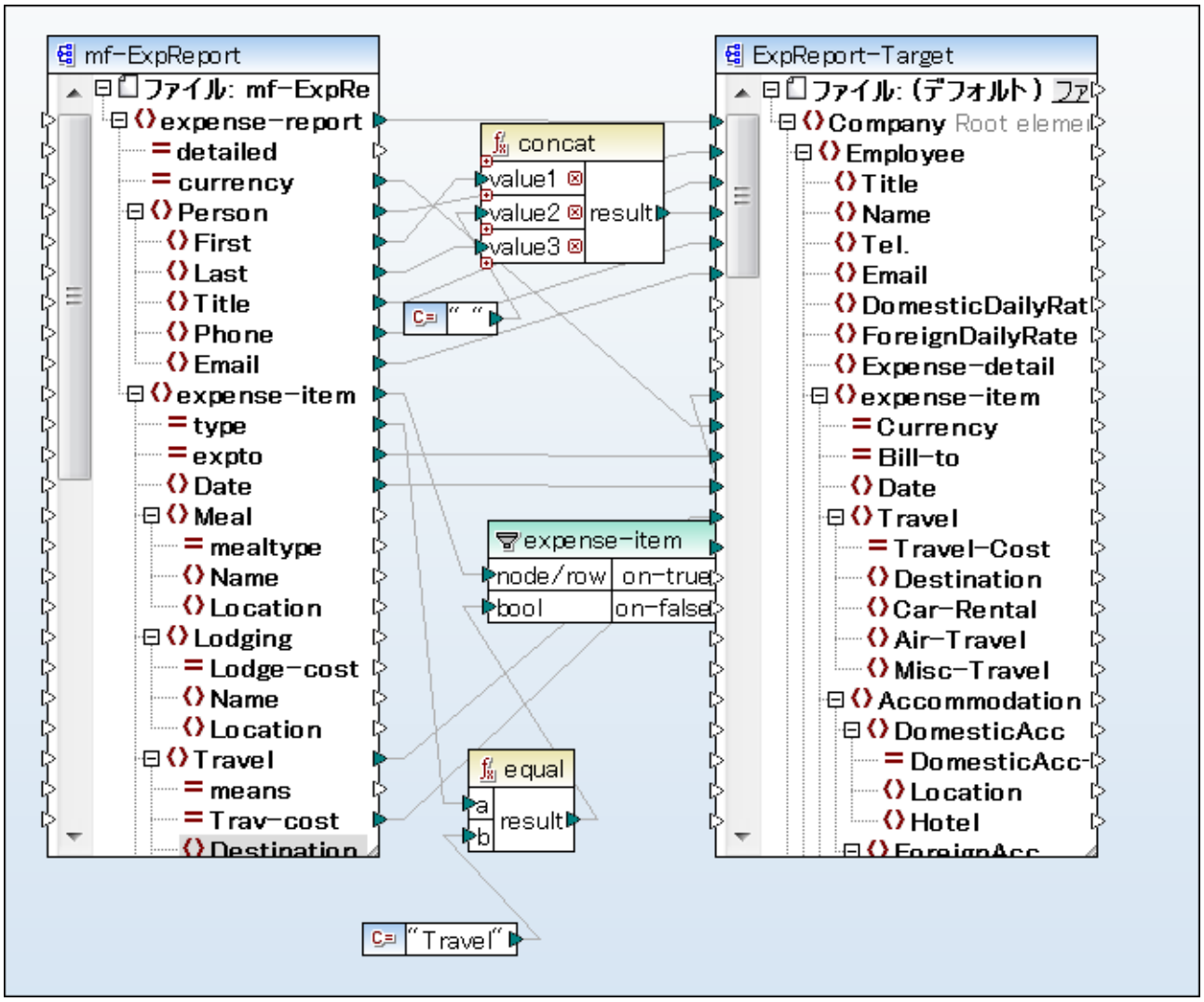
上に表示されるようなメッセージを非表示にするには 以下を行います:

1. 「ツール」メニューから「オプション」をクリックします。
2. 「メッセージ」グループをクリックします。
3. 接続を作成した際、祖先アイテムへの接続を提案アイテム チェックボックスをクリックします。

3.3.8 接続と子接続の移動

異なったコンポーネントに接続を移動する場合、MapForce は、自動的に同じ子接続をマッチし、この接続が新しい場所に移動されるべきかプロンプトします。この機能の通常の使用方法は、既存のマッピングがある場合、ターゲットスキーマのルート要素の変更です。通常、このような状況が発生すると、すべての 降順 接続 を手動でマッピングを再度行う必要がありますが、この機能はそのような状況を回避するために役に立ちます。

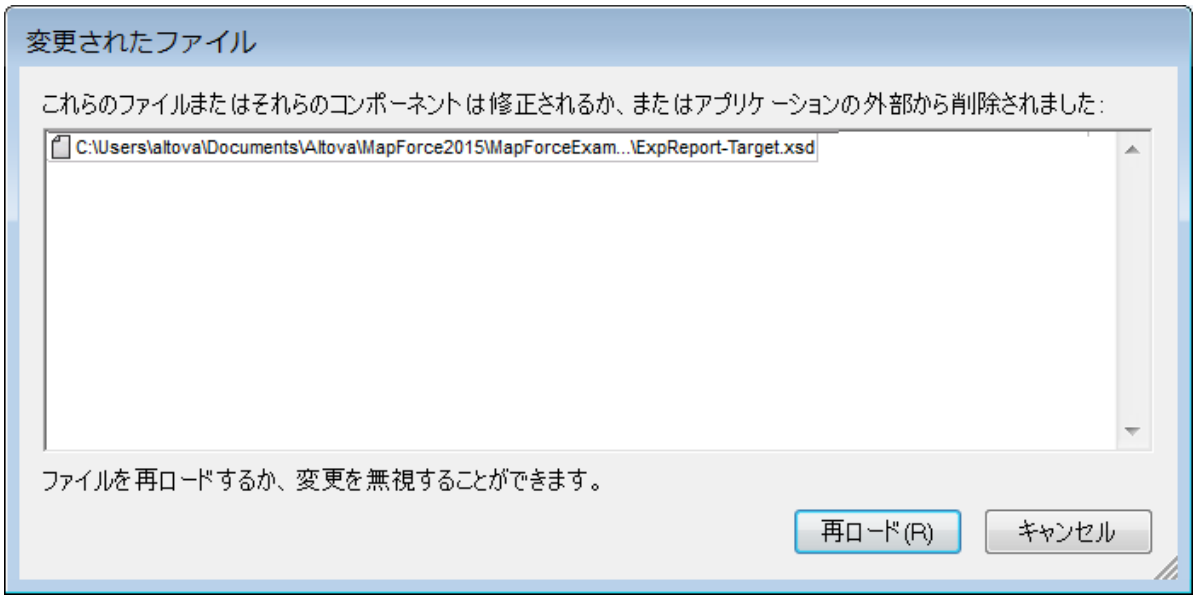
<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダーで使用することのできる **Tut-ExpReport.mfd** ファイルは、この例で修正され、別名で保存されます。



Tut-ExpReport.mfd (MapForce Basic Edition)

動作を理解するためには、以下を行います:

1. **Tut-ExpReport.mfd** サンプルマッピングを開きます。
2. ターゲットスキーマの `Company` ルート要素を `Company-EU` に変更するために、MapForce の外部で **ExpReport-Target.xsd** スキーマを編集します。MapForce を閉じる必要はありません。
3. ターゲットスキーマの `Company` ルート要素を `Company-EU` に変更すると、MapForce に「変更されたファイル」プロンプトが内に表示されます。

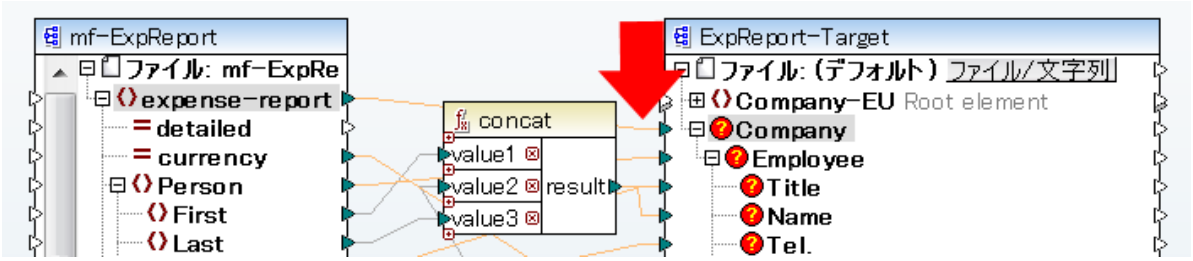


4. 更新されたスキーマを再ロードするために、「**再ロード**」ボタンをクリックします。ルート要素が削除されたため、コンポーネントは複数の見つからないノードを表示します。

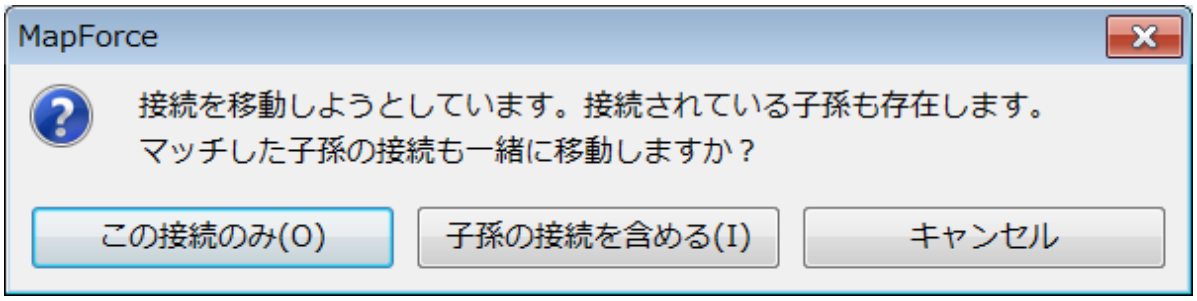
5. コンポーネントの上の「**新しいルート要素の選択**」をクリックします。(コンポーネントヘッダーを右クリックしてコンテキストメニューから「**ルート要素の変更**」を選択することで、ルート要素を変更することができます)。

6. `Company-EU` を新しいルート要素として選択し、「**OK**」をクリックして確認します。`Company-EU` ルート要素がコンポーネントの上に表示されます。

7. ソースコンポーネントの`expense-report`アイテムとターゲットコンポーネントの`Company`アイテム間にある接続のターゲットスタブをクリックして、`Company-EU`ターゲットコンポーネントのルート要素の上にドラッグアンドドロップします。



通知ダイアログボックスが現れます。



8. 「子孫の接続を含む」をクリックする。これにより、新しいルート要素の下で再度マップするようにMapForce に命令し、マッピングは再度有効となります。

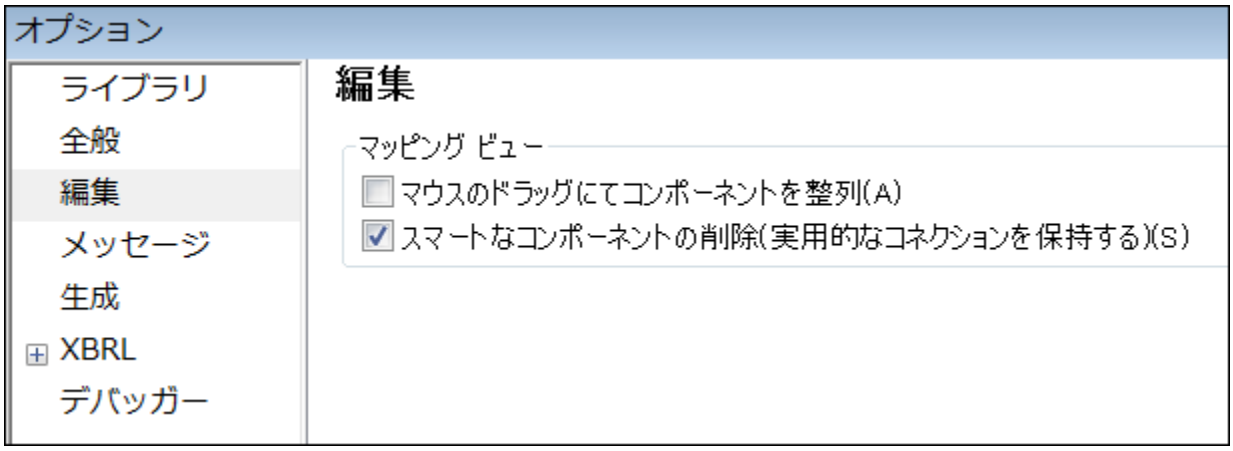
メモ マッピングするノードが、ソースノードと同じ名前を持ち、異なる名前空間にある場合、通知ダイアログボックスに、以下を表示する追加ボタンが表示されます:「子孫とマップ名前空間を含む」。このボタンをクリックすると、同じ名前空間の子接続が同じ子ノードのソース親ノードとして移動されます。このボタンをクリックすると、異なる名前空間ノードの下の同じ子ノードのソース親ノードと同じように同じ名前空間の子接続を移動します。

3.3.9 コンポーネント削除後の接続の保持

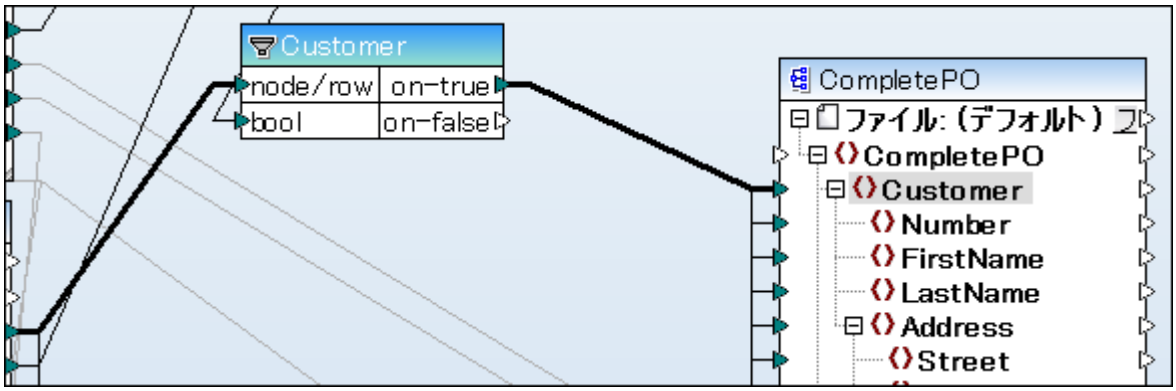
他のコンポーネントに複数の(子) 接続を持つ、コンポーネントを削除すると、何が起こるかを決定することができます。例: フィルターまたはソートコンポーネント。全ての子接続を保持し、個別に復元する必要がないためこの機能は役に立ちます。

コンポーネントが削除された後に、子接続を保持または復元すること、または全ての子接続をすぐに接続することが選択できます。

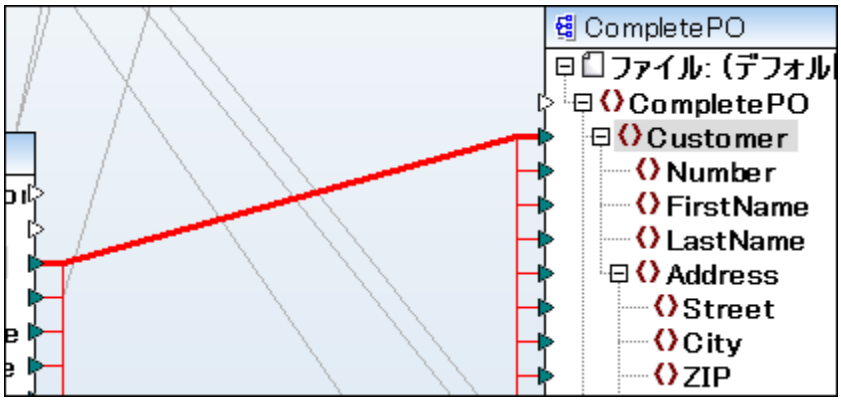
現在の設定を確認するために**ツール | オプション | 編集**」(タブ) を選択します。チェックボックスのデフォルト設定は、**非アクティブ**です。すなわち「スマートなコンポーネント削除(役に立つ接続を保持)」が無効化されます。



例: ...¥MapForceExamples フォルダー内で CompletePO.mfd マッピングを使用するには、チェックボックスがアクティブ化されていると、Customer フィルターは、以下に表示される子アイテムに接続された全てコピー 接続になります。



Customer フィルターを削除すると、削除の確認を問うプロンプトが表示されます。「はい」を選択すると、フィルターは削除されますが、子接続は保持されます。



その他の接続はまだ選択されています(すなわち、赤で表示される箇所)。削除する場合は、Del キーを押します。

マッピングエリアをクリックすると、コネクタの選択が解除されます。

「スマートなコンポーネント削除...」チェックボックスが無効化されている場合、フィルターを削除すると全ての子接続がすぐに削除されます。

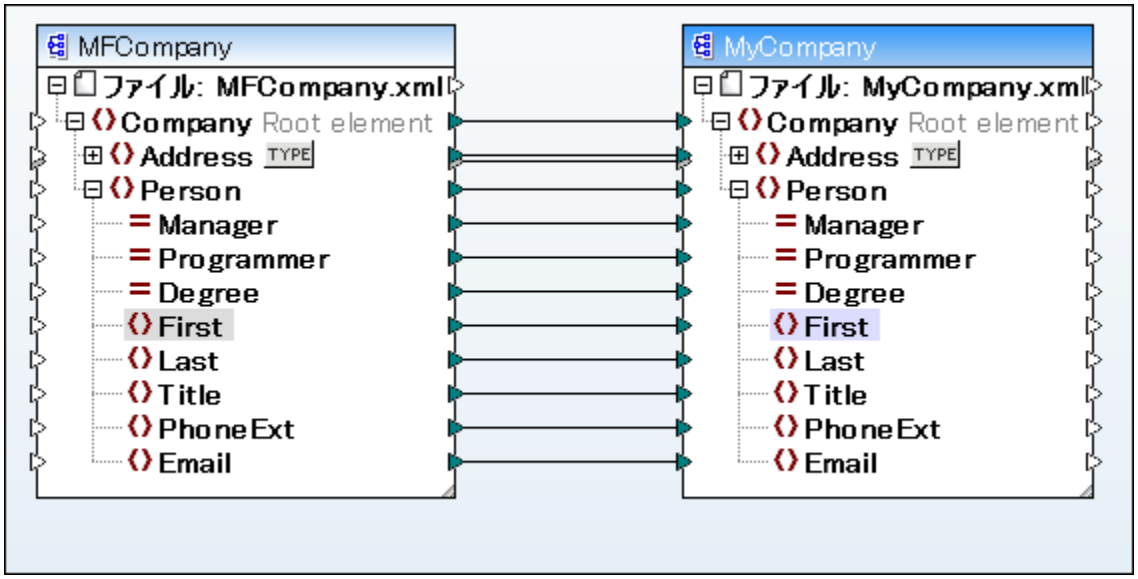
メモ　　フィルターコンポーネントが「on-true」と「on-false」出力に接続されている場合、両方の出力のための接続は保持されます。

3.3.10 見つからないアイテムの扱い方

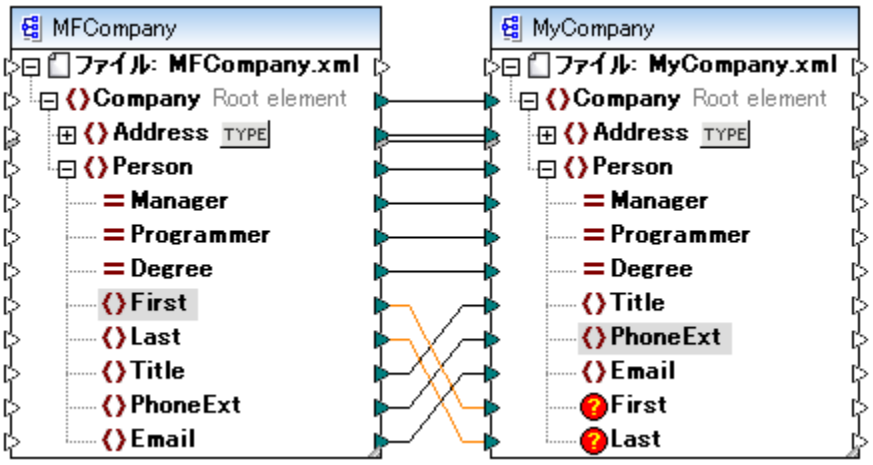
時が経つに従って、XML スキーマにて定義された要素や属性が追加/削除され、マッピング内にあるコンポーネントの構造が変更されるということがあります。MapForce ではプレースホルダーアイテムを使用することで、全ての接続を保持し、アイテムが削除された際にコンポーネント間で関連する接続データの保持を行います。

サンプル

MFCompany.xsd スキーマファイルをサンプルファイルとして使用します。スキーマファイルを MyCompany.xsd という名前でコピーし、両スキーマ間にて Company アイテムに対する接続を作成します。これで子要素の自動接続が有効になっている場合、コンポーネント間にある全ての子アイテムに対する接続が作成されます。



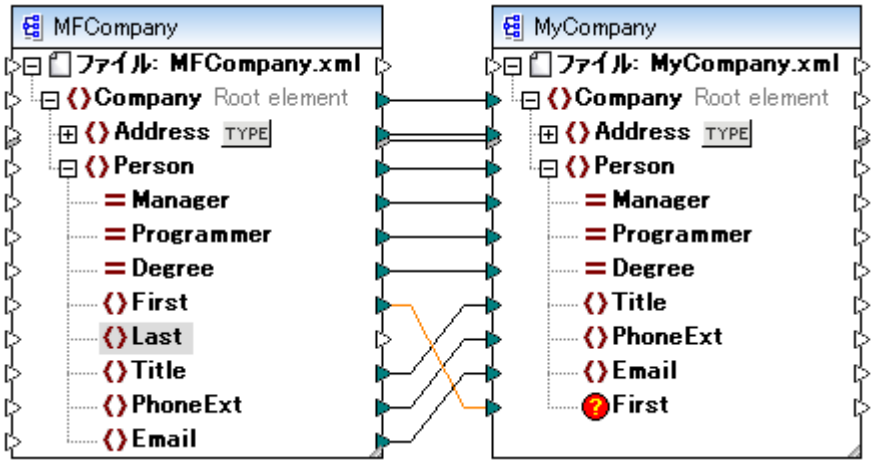
XMLSpy にて MyCompany.xsd を編集し、スキーマ内にある First ならびに Last アイテムを削除します。MapForce へ再度切り替えると、スキーマを再ロードするよう促されます。再ロードすることで、MapForce 内にあるコンポーネントが更新されます。



削除されたアイテムならびにコネクタが、アイコンとともに MyCompany コンポーネントに表示されます。必要な場合は他のアイテムへコネクタを再接続することができるほか、コネクタを削除することもできます。

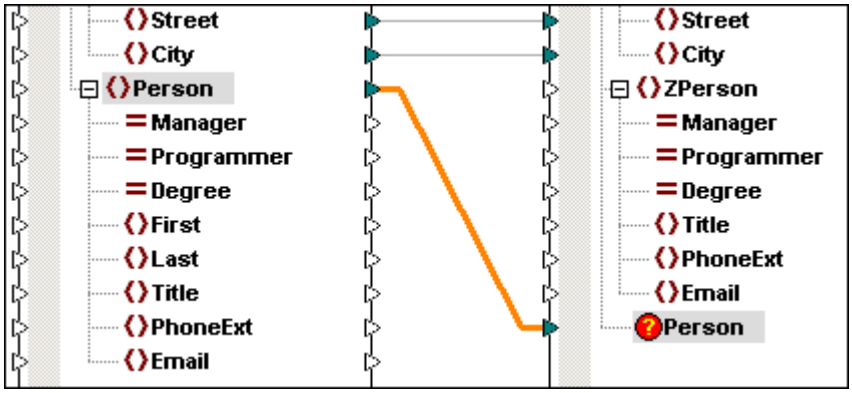
この状態でマッピング（または生成されたコード）のプレビューを行うこともできますが、警告がメッセージウィンドウに表示されます。プレビューならびにコード生成において、上のアイコンで示されるような、見つからないアイテムは無視されます。

ハイライトされているコネクタをクリックし、削除することで、見つからないアイテム（この場合はMyCompany 内部のLast）がコンポーネントから削除されます。



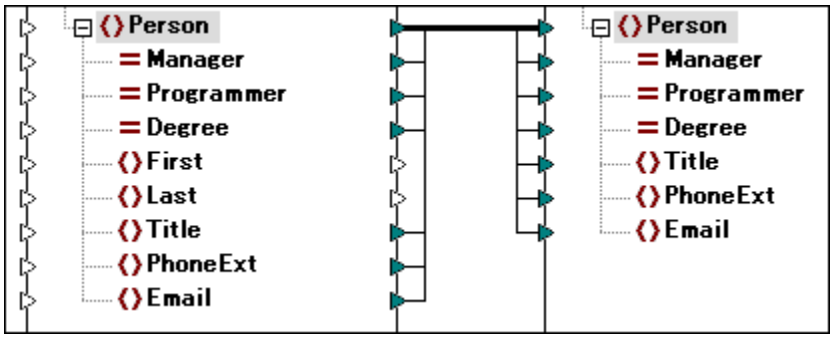
名前の変更されたアイテム

例えばPerson という親アイテムがZPerson という名前に変更された場合、オリジナル（Person）の親アイテムコネクタが保持され、子アイテムのコネクタが削除されます。



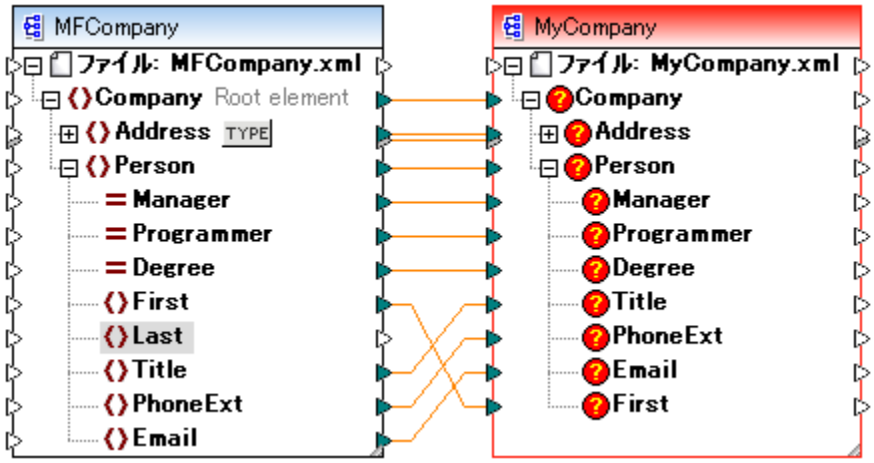
「全てコピー」接続と見つからないアイテム

全てコピー接続は通常の接続と同じように扱われますが、見つからないアイテムへの接続は保持も表示もされません。

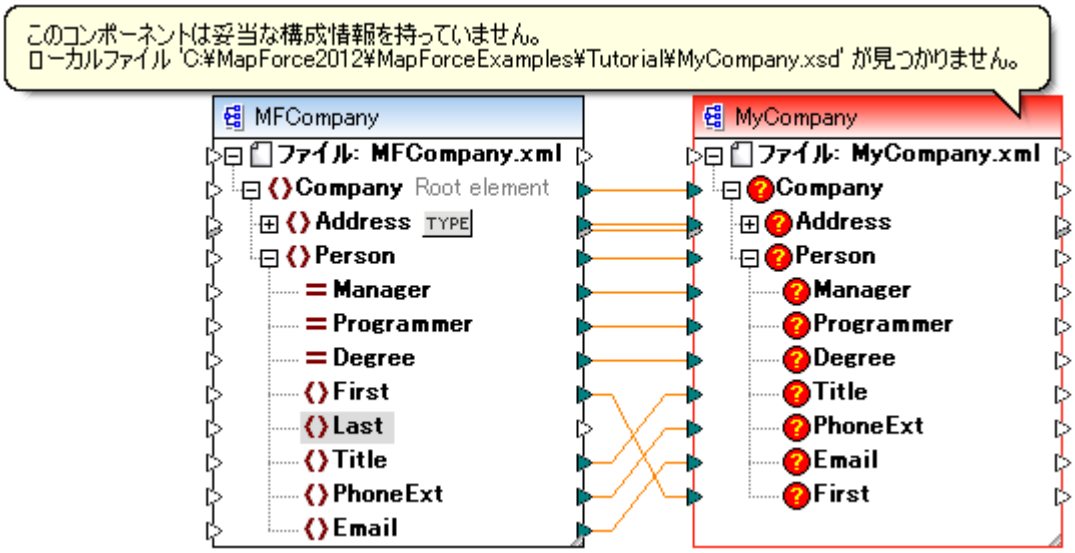


名前変更または削除されたコンポーネントソース

スキーマやコンポーネントのデータソースが名前変更または削除された場合、コンポーネントに含まれる全てのアイテムがハイライトされます。コンポーネントの外周であるフレームにより、スキーマに対して正常な接続がなされていないことが示されます。この状態でプレビューやコード生成を行うことはできません。



マウスカーソルをハイライトされているコンポーネントへマウスオーバーすることで、ポップアップが表示されます。



ハイライトされたコンポーネントのタイトルバーをダブルクリックすることで、コンポーネント設定ダイアログボックスが表示、異なるスキーマを選択することができます。コンポーネント設定の変更を参照してください。

4 マッピングのデザイン

Altova W eb サイト: データ統合ツール

このチャプターは、データマッピングのデザイン方法とマッピングエリア上のデータの変換方法について説明します。また、マッピングデザインに関連した、注意点に関しても説明します。以下のロードマップを利用して、特定のタスクまたはコンセプトに素早くアクセスすることができます:

...を実行するには	以下を参照してください...
その他のスキーマのパス参照とマッピングで使用されるインスタンスと他のファイルの作成と編集を行います。	相対と絶対パスの使用
必要に応じてデータマッピングを微調整します。(例: ターゲットコンポーネント内のアイテムのシーケンスの調整)。	マッピング接続の種類
異なるスキーマを持つ複数のソースからのデータを単一のスキーマにマップします。	複数のスキーマからデータをマージする
コンポーネントの出力を他のコンポーネントの入力として使用します。	チェーンマッピング/パススルーコンポーネント
複数のファイルを同じマッピングにソースまたはターゲットとして処理します。(例: ディレクトリ内の全てのファイル)。	複数の入力または出力ファイルを動的に処理
(文字列パラメーターなど) 外部の値をマッピングにパスします。	マッピングにパラメーターを与える
ファイルの代わりに、マッピングから文字列の値を取得します。	マッピングから文字列の値を返す
マッピングデータの一部を後の処理のために一時的に保管します (プログラミング言語内の変数に類似)。	変数の使用
昇順 または 降順の順序でデータを並べ替える。	データの置換え
特定の条件に従いnodes/rows をフィルター、または値を処理します。	フィルターと条件
異なるスキーマを持つ複数のソースからデータをマージします。	複数のスキーマからデータをマージする
キーと値のペアの処理を行います。例: 数値の表示を月数に変換します。(01、02、などを) テキストの表示 (一月、二月、などに) 変換します。	Value-Map の使用
複雑なマッピングのデザインで予期しない結果の発生を回避します。	マッピングのルールと戦略

重要な点は、MapForce は、多種の処理タスクを助ける、広範囲なビルドイン関数ライブラリを搭載している点です (関数ライブラリファレンスを参照)。ビルドインライブラリでも、十分ではない場合、自身のカスタム関数を MapForce で作成し、外部 XSLT ファイルで再利用することができます。更に詳しい情報に関しては、関数の使用を参照してください。

4.1 相対と絶対パスの使用

マッピングデザインファイル(*.mfd) は、複数のスキーマとインスタンスファイルを参照する場合があります。MapForce により使用されるスキーマファイルは、マップされるデータの構造を決定し検証します。インスタンスファイル、は他方、スキーマに対してソースデータを読み込み、プレビューし、検証することが要求されます。

マッピングにコンポーネントを追加する場合、マッピングデザインで使用されるファイルへの参照はすべて MapForce により作成されます。ですが、上記のパス参照を常に手動で設定または変更することができます。

このセクションは設定のための命令、またはマッピングにより参照されるその他のファイル型にパスの変更方法、相対と絶対パスの使用が指する意味について説明します。

4.1.1 コンポーネント上で相対パスを使用する

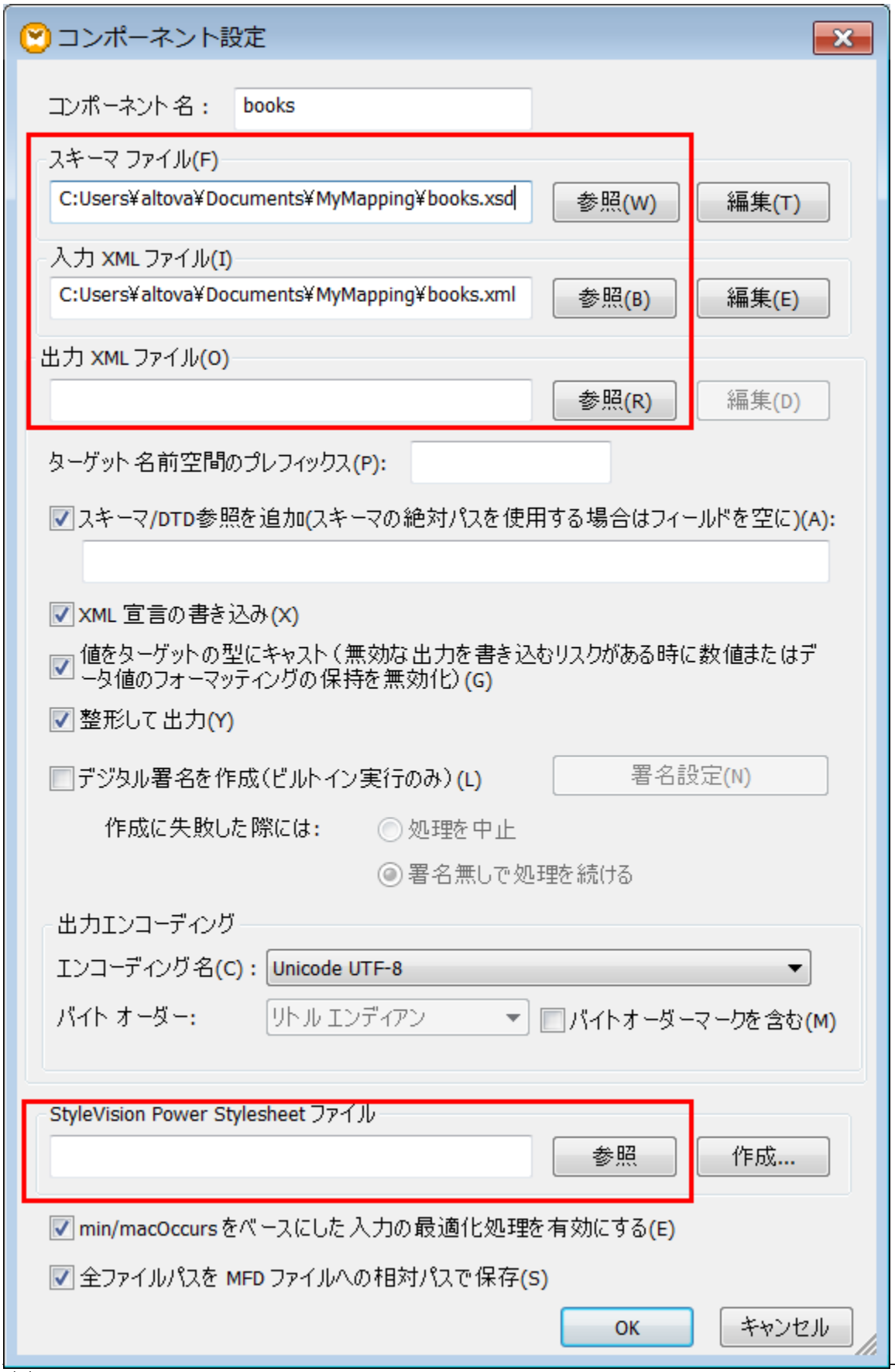
コンポーネント設定ダイアログボックス(XML コンポーネントのために下に表示されています) は、コンポーネントにより参照される可能性のある複数のファイルのための絶対または相対パスを指定するオプションを提供します:

- 入力ファイル(すなわち、MapForce がデータを読み込むファイル)
- 出力ファイル(すなわち、MapForce がデータを書き込むファイル)
- スキーマファイル(スキーマを持つコンポーネントに適用可能)
- 構造ファイル(ユーザー定義関数の入力または出力パラメーター, または変数などの複雑な構造を持つコンポーネントに適用可能)
- PDF、HTML とW ord などの出力のためのデータをフォーマットするために使用される、StyleVision Power Stylesheet (*.sps) ファイル

対応するテキストボックス内で相対パスを直接入力することができます(下のイメージで赤線の枠内に表示されています)。

> 相対ファイルパスを入力する前に、マッピングファイル(.mfd) を最初に保存してください。それ以外の場合、全ての相対パスはの W indows のパーソナルアプリケーションフォルダーに対して解決されます(ドキュメント¥Altova¥MapForce2021)これは、予期しない振る舞いの場合があります。

.mfd ファイルマッピングに対して相対的な上記のファイルパスすべてを保存するようにMapForce に命令することができます。下のサンプルイメージでは、「MFD ファイルに相対的なすべてのファイルパスを保存する」オプションに注意してください。(デフォルトかつ奨励されるオプションである)チェックボックスが有効化されていると、コンポーネントにより参照されるファイルのパスは、マッピングデザインファイル(.mfd) のパスに相対的に保存されます。これはコンポーネントに参照される全てのファイルに影響を及ぼします(イメージで赤線の枠内に表示されています)。


コンポーネント設定
コンポーネント名：
books
スキーマ ファイル(F)
C:Users¥altova¥Documents¥MyMapping¥books.xsd
参照(W)
編集(T)
入力 XML ファイル(I)
C:Users¥altova¥Documents¥MyMapping¥books.xml
参照(B)
編集(E)
出力 XML ファイル(O)
参照(R)
編集(D)
ターゲット名前空間のプレフィックス(P):
スキーマ/DTD参照を追加(スキーマの絶対パスを使用する場合はフィールドを空に)(A):
XML 宣言の書き込み(X)
値をターゲットの型にキャスト（無効な出力を書き込むリスクがある時に数値またはデータ値のフォーマッティングの保持を無効化）(G)
整形して出力(Y)
デジタル署名を作成（ビルトイン実行のみ）(L)
署名設定(N)
作成に失敗した際には:
処理を中止
署名無しで処理を続ける
出力エンコーディング
エンコーディング名(C)：
Unicode UTF-8
バイト オーダー:
リトル エンディアン
バイトオーダーマークを含む(M)
StyleVision Power Stylesheet ファイル
参照
作成...
min/macOccurs をベースにした入力の最適化処理を有効にする(E)
全ファイルパスを MFD ファイルへの相対パスで保存(S)
OK
キャンセル

コンポーネント設定ダイアログボックス

上で説明されているコンポーネントはXML コンポーネンですが、 設定「**MFD ファイルに相対的なすべてのファイル パスを保存する**」は、下のファイルでも同じように動作します:

- 複雑なユーザー定義関数の入力または出力パラメーターと複雑な型の変数により使用される構造ファイル
- 入力または出力 フラットファイル*
- XML フィールドをサポートするデータベースコンポーネントにより参照されるスキーマファイル*
- 入力または出力 XBRL、FlexText、EDI、Excel 2007+、JSON ファイル**

*MapForce Professional と Enterprise Edition
**MapForce Enterprise Edition のみ

上のコンポーネントを例として、.mfd ファイルが**books.xsd** と**books.xml** ファイルと同じフォルダーに存在する場合、パスは、以下のとおり変更されます:

C:\Users\altova\Documents\MyMapping\books.xsd は、**books.xsd** に変更されます。
C:\Users\altova\Documents\MyMapping\books.xml は、**books.xml** に変更されます。

ローカルではないドライブまたはURL を使用するパスは相対的にはなりません。

「**上書き保存**」メニューコマンドを使用して、マッピングを新規のフォルダーに保存する場合、チェックボックスが有効化されている場合、MapForce は、コンポーネントにより参照されるファイルを記録します。また、全てのファイルがマッピングと同じディレクトリに存在する場合、ディスク上の新しい場所にディレクトリ全体を移動してもパスの参照が壊されることはありません。

相対パスを使用することと「**MFD ファイルに相対的なすべてのファイル パスを保存する**」チェックボックスを有効化）は多くの場合とても重要です。例:

- オペレーティングシステム上のマッピングのロケーションが将来変更される可能性のある場合。
- （SVN などの）バージョンコントロールシステムを使用して）マッピングがソースコントロール化にあるディレクトリに保管されている場合。
- 異なったマシンに、または異なったオペレーティングシステムにマッピングを実行のためにデプロイする場合。

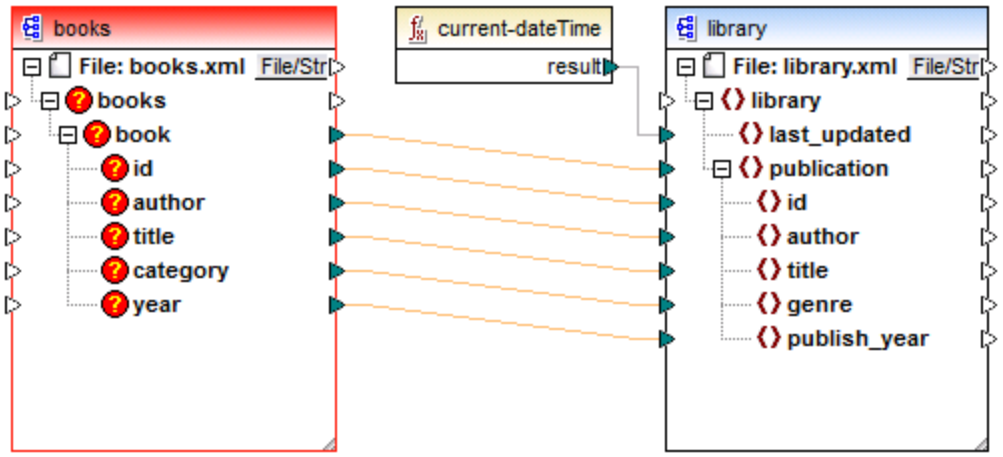
「**MFD ファイルに相対的なすべてのファイル パスを保存する**」チェックボックスが無効化されている場合、マッピングの保存は、ファイル パスを変更しません（すなわち、コンポーネント設定ダイアログボックス内で表示されるままになります）。

4.1.2 破損したパスの参照を修正する

マッピング内でファイルレファレンスを追加または変更する場合にパスが解決されない場合、MapForce は、警告メッセージを表示します。こうすることにより、MapForce は、破損したパスへの参照を削減します。しかしながら、以下の場合破損したパスへの参照が起こる可能性があります:

- 相対パスを使用し、マッピングファイルをスキーマとインスタンスファイルを移動することなく、新しいディレクトリに移動する場合。
- マッピングファイルと同じディレクトリ内のファイル絶対パスを使用し、ディレクトリを他の場所に移動する場合。

このような状況が発生すると、MapForce は、このコンポーネントを赤でハイライトします。例:



破損したパスの参照

この場合の解決方法は、コンポーネントヘッダーをダブルクリックして、「コンポーネント設定」ダイアログボックス内で破損したパスを更新します（コンポーネント設定を変更するも参照）。

4.1.3 多種の実行環境内のパス

コードをマッピングから生成し、生成されたファイルは、MapForce で実行することはできません。選択されたターゲット環境で実行できるようになります。（例: RaptorXML Server）マッピングの実行に成功するには、全ての相対的なパスはマッピングが実行される個所で意味をなす必要があります。

従い、インスタンスまたはスキーマファイルに対してマッピングが相対パスを使用する場合、各ターゲット言語のベースパスに関しては以下のテーブルを考慮してください：

ターゲット言語	ベースパス
XSLT、XSLT2、XSLT3	XSLT ファイルのパス
XQuery*	XQuery ファイルのパス
C++、C#、Java*	生成されたアプリケーションの作業ディレクトリ
BUILT-IN*（MapForce でマッピングをプレビューする場合）	マッピング（.mfd）ファイルのパス
BUILT-IN*（MapForce Server でマッピングを実行する場合）	現在の作業ディレクトリ
BUILT-IN*（FlowForce Server の管理下で、MapForce Server でマッピングを実行する場合）	ジョブの作業ディレクトリまたは FlowForce Server の作業ディレクトリ

＊MapForce Professional と Enterprise エディションで使用可能な言語

必要な場合、マッピングのためにコードを生成する際、MapForce に全てのパスを相対から絶対パスに変換するように命令することができます。このオプションは、マッピングコードを、同じオペレーティングシステム内で、またはマッピングにより使用される絶対パスへの参照がある他のオペレーティングシステムで、実行する際に役に立ちます。

生成されたコード内で全てのパスを絶対パスに変換する方法:

1. マッピングの空のエリアを右クリックします。マッピングの設定 ダイアログボックスが表示されます。

2. 「生成されたコードでは絶対パスを使用する」チェックボックスを選択します。

コードを生成する際、チェックボックスが選択されていると、MapForce は、マッピングファイル(.mfd) のディレクトリをベースにした相対パスを解決し、生成されたコード内でこれらを絶対パスにします。この設定は、以下のファイルのパスに影響を及ぼします:

- すべてのファイルベースのコンポーネント型のための入力と出力インスタンスファイル

チェックボックスが選択されていない場合、ファイルパスは、コンポーネント設定内で定義されたとおり保存されます。

生成されたコード内のライブラリパス

マッピングファイルにはオプションで多種のライブラリへのパス参照が含まれている場合があります。例えば、マッピングファイルは他のマッピングまたはカスタム XSLT、XQuery*、C#*、または Java* ライブラリまたは .mff* (MapForce 関数) ファイルからの関数からのユーザー定義関数をインポートする場合があります。関数ライブラリの管理も参照してください。

** MapForce Professional と Enterprise エディションで使用可能な機能*

オプション「生成されたコード内でパスを絶対パスにする」はマッピングコンポーネントのみに適用され、外部ライブラリへのパスに影響を与えません。代わりに XSLT と XQuery 以外のすべてのライブラリのためにライブラリパスは解決され、生成されたコード内で絶対パスに変換されます。例えば、マッピングファイルに .NET .dll または Java .class ファイルが含まれ、生成されたコードを他の環境 (例えば他のコンピューター) で実行する場合、参照されるライブラリはターゲット環境内の同じパスに存在する必要があります。

XSLT または Xquery ファイルをマッピングから生成する場合、*生成された XSLT または Xquery ファイルに相対的な* ライブラリパスを以下のように作成することができます:

1. マッピングの空のエリアを右クリックします。マッピングの設定 ダイアログボックスが表示されます。
2. チェックボックス「XSLT / Xquery ファイルに相対的なパスを持つレファレンスライブラリ」を選択します。

> 上のチェックボックスを選択すると XSLT または Xquery ライブラリファイルが実際にそのパスに存在することを確認してください。

XSLT または Xquery ライブラリファイルへのパスを生成されたコード内で絶対パスにするにはこのチェックボックスをクリアします。

4.1.4 コピーと貼り付けと相対パス

マッピングからコンポーネントをコピーし、貼り付ける場合は、スキーマファイルの相対パスを目的のマッピングのフォルダーに対して解決することができます。もし、パスが解決されない場合、相対パスを絶対パスにソースマッピングのフォルダーを使用して変更するようにプロンプトされます。最初に目的のマッピングに保存することが奨励されます。それ以外の場合、相対パスは、個人のアプリケーションフォルダーに対して解決されます。

4.2 マッピング接続の種類

マッピング接続を作成すると、オプションで接続（子アイテムを持つソースとターゲットアイテム）の種類を以下から選択することができます。

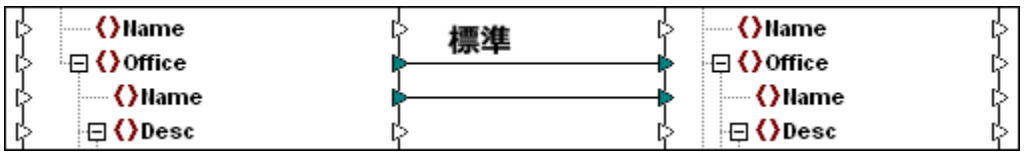
- ターゲット優先（標準）
- ソース優先（混合コンテンツ）
- 全てコピー（子アイテムのコピー）

接続の種類はマッピングにより生成された出力内の子アイテムのシーケンスを決定します。このセクションは各接続の種類と役に立つシナリオについての情報を提供します。

4.2.1 ターゲット優先マッピング

ターゲット優先（標準マッピング）は、MapForce におけるデフォルトのマッピング方法で、ターゲットスキーマ内のノードのシーケンスにより出力が決定されます。このマッピングの種類は大部分のシナリオに適しており、MapForce のデフォルトのマッピングとして設定されています。

標準マッピングは実線で表示されます。



ターゲット優先マッピングは（文字データや子要素などの）複合コンテキストを含むXML ノードをマップする場合適していない場合があります。例:

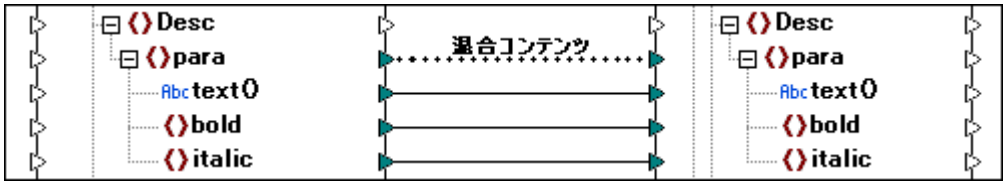
```
<p>This is our <i>best-selling</i> product.</p>
```

複合コンテンツが含まれるマッピングの場合、ソースファイル内で現れるアイテムのシーケンスを保持ため、ソース優先マッピングを使用することが奨励されます（ソース優先マッピング を参照）。

4.2.2 ソース優先マッピング

ソース優先（混合コンテンツ）マッピングを使うことで、テキストならびに子ノードを、XML ソースファイルにある順序通りにマッピングすることができます。

- 混合コンテンツテキストノードコンテンツがサポートされます。
- 子ノードの順序はソース XML インスタンスファイルに依存します。



混合コンテンツのマッピングは点線により表示されます。

勿論、ソース優先 / 混合コンテンツマッピングを XML スキーマの複合型に適用することもできます。XML ソースファイルの順序に従って子ノードのマッピングは行われます。

ソース優先 / 混合コンテンツマッピングでは以下がサポートされます:

マッピング

- ソースコンポーネントとして:
  - XML スキーマ複合型(mixed=true となっている混合コンテンツを含む)
- ターゲットコンポーネントとして:
  - XML スキーマ複合型(混合コンテンツを含む) メモ: CDATA セクションはテキストとして扱われます。

4.2.2.1 混合コンテンツのマッピング

以下のサンプルでは **...\MapForceExamples\Tutorial** フォルダー以下にある **Tut-OrgChart.mfd**、**Tut-OrgChart.mfd.xml**、**Tut-OrgChart.mfd.xsd, Tut-Person.xsd** を使用します。

ソース XML インスタンス

このセクションで使用されている **Tut-OrgChart.XML** ファイルの一部を以下に示します。ここで注目したいのは「bold」ならびに「italic」子ノードも含まれている「para」の混合コンテンツです。

para 要素には処理命令 (`<?sort alpha-ascending?>`) だけではなく、コメントテキスト(`<!--Company details... -->`)も含まれていることにご注目してください。これらのデータもマッピングすることができます。

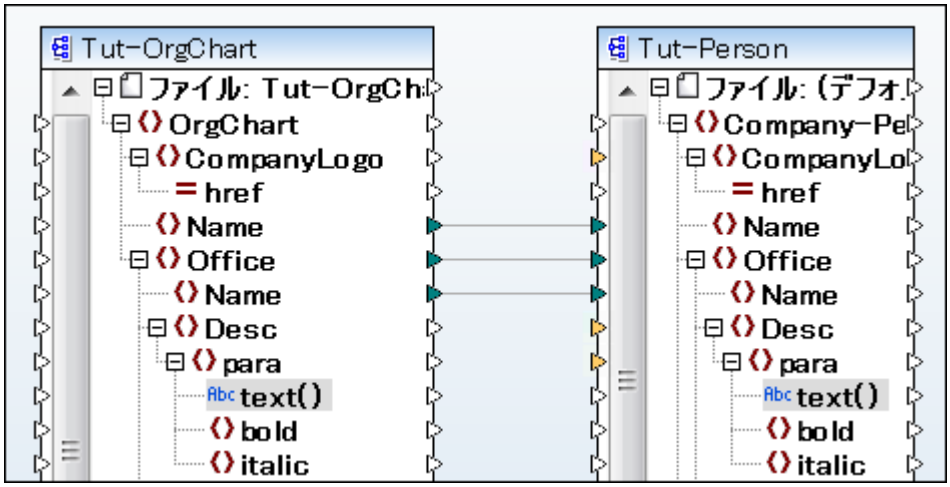
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 sp2 U (http://www.altova.com) by Mr. Nobody (Altova GmbH) -->
<OrgChart xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Tut-OrgChart.xsd">
  <CompanyLogo href="nanonull.gif"/>
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull
develops nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999
saw the unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand
its operations <italic>offshore</italic>to drive down operational costs.
        <?sort alpha-ascending?>
        <!--Company details: location and general company information.-->
      </para>
      <para>White papers and further information will be made available in the near future.
    </Desc>
```

XML インスタンスファイルにあるテキストのシーケンスならびに bold/italic ノードは以下のような構成(順序)となっています:

```
<para> The company...
    <bold>Vereno</bold>in 1995 ...
    <italic>multi-core...</italic>February 1999

    <bold>Nano-grid.</bold>The company ...
    <italic>offshore...</italic>to drive...
</para>
```

初期のマッピング

Tut-OrgChart.mfd を開いたときのマッピングを以下に示します。



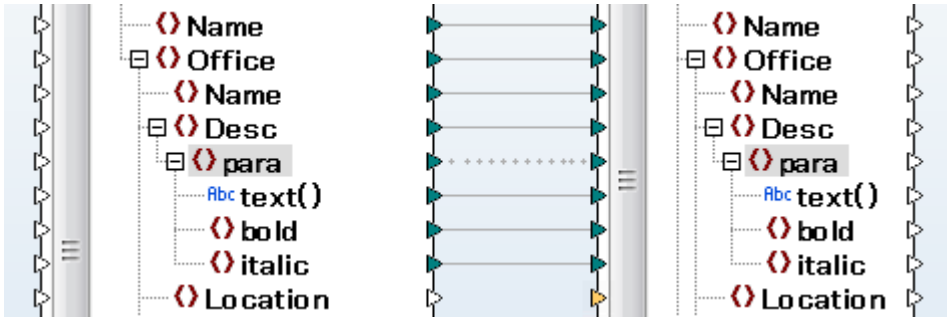
上のマッピングを出力

上に示されるマッピング結果を以下に示します。組織図 や個々のオフィス名が表示されます。

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
3     <Name>Organization Chart</Name>
4     <Office>
5       <Name>Nanonull, Inc.</Name>
6     </Office>
7     <Office>
8       <Name>Nanonull Europe, AG</Name>
9     </Office>
10  </Company-Person>
11
```

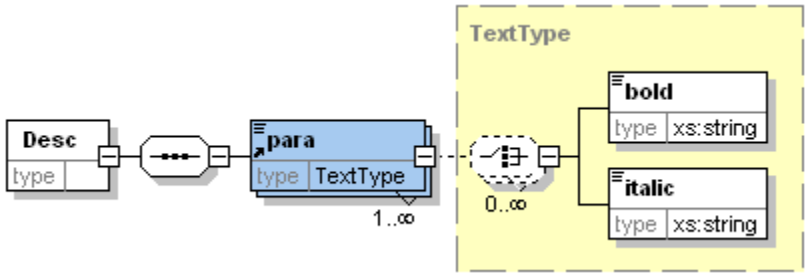
para 要素のマッピング

以下のスクリーンショットは混合コンテンツマッピングの例です。混合コンテンツのpara 要素にコネクタが接続されており、点線によりそれが混合コンテンツであることを表しています。**text()** ノードにはテキストデータが含まれており、ターゲットコンポーネントへテキストを表示するには、このノードをマッピングする必要があります。



コネクタを右クリックしてプロパティを選択することで、コネクタに注釈を追加し、ラベルを付与することができます。詳細については接続の注釈を参照ください。

以下のイメージは**Tut-OrgChart.xsd** スキーマファイルにあるDesc 要素のコンテンツモデルを表しています。この例の場合、この定義はソースならびにターゲットスキーマの両方にて共通したものとなっています。

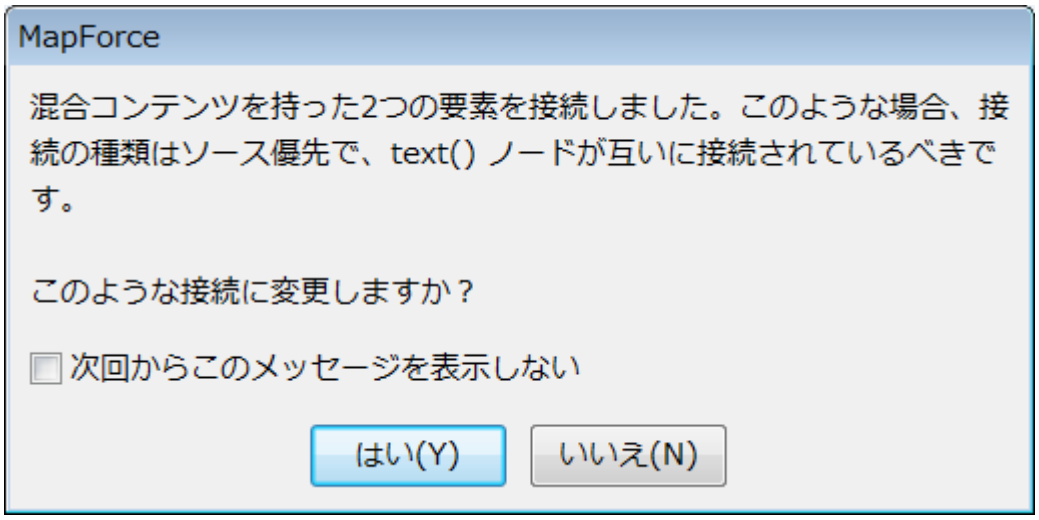


コンテンツモデルにある para 要素について、以下の事柄を理解できます:

- **para** は「TextType」という型で、mixed="ture" の複合型です。
- **bold** ならびに**italic** 要素は両方とも "xs:string" 型となっており、この例では再帰的な型として定義されていません(つまり **bold** も **italic** も "TextType" 型ではありません)。
- **bold** ならびに**italic** 要素は、**para** 内部においてどのような順序でも無制限の数だけ出現することができます。
- **para** 要素内では**bold** ならびに**italic** 要素とともに、任意のテキストを入力することができます。

アイテム間で混合コンテンツの接続を作成:

1. メニューオプションの「接続 | 子要素の自動接続」を選択して、有効にします(既に有効になっていない場合)。
2. ソーススキーマにて **para** アイテムから、ターゲットスキーマの **para** アイテムへの接続を行います。MapForce にコネクタをソース優先にするかを問うメッセージが表示されます。



3. 「はい」をクリックして混合コンテンツ接続を作成します。

メモ para は混合コンテンツであるため、この時点でメッセージが表示されます。自動接続オプションを無効にした状態で、para アイテムを直接接続した際にも混合コンテンツのメッセージが表示されます。

para アイテムにある全ての子アイテムが接続されました。para アイテムへ接続しているコネクタが点線で表示され、それが混合コンテンツであることを表しています。

4. 出力タブをクリックして、マッピングの結果を確認します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Company-Person xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNames
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull devel
        </para>
      <para>White papers and further information will be made available in the near future.
      </para>
    </Desc>
  </Office>
  <Office>
    <Name>Nanonull Europe, AG</Name>
    <Desc>
      <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team co
    </Desc>
  </Office>
</Company-Person>
```

5. 出力タブのアイコンバーにあるワードラップアイコン  をクリックして、全てのテキストを出力ウィンドウにて表示します。

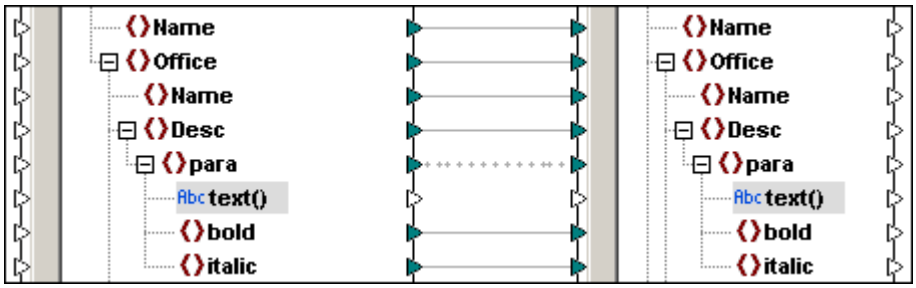
```
  <Name>Organization Chart</Name>
  <Office>
    <Name>Nanonull, Inc.</Name>
    <Desc>
      <para>The company was established in<bold> Vereno</bold>in 1995. Nanonull develops
nanoelectronic technologies for<italic>multi-core processors.</italic>February 1999 saw the
unveiling of the first prototype <bold>Nano-grid.</bold>The company hopes to expand its
operations <italic>offshore</italic>to drive down operational costs.
        </para>
      <para>White papers and further information will be made available in the near future.
      </para>
    </Desc>
  </Office>
  <Office>
    <Name>Nanonull Europe, AG</Name>
    <Desc>
      <para>In May 2000, Nanonull<italic>Europe</italic> was set up in Vienna. The team
consists of<bold> five research scientists </bold>and one administrative staff.</para>
    </Desc>
  </Office>
</Company-Person>
```

各オフィスに関する記述である混合コンテンツテキストが正しくマッピングされました。テキストだけでなく bold ならびに italic タグコンテンツが、XML ソースファイルにある通りにマッピングされました。

6. マッピングビューに切り替えます。

混合コンテンツからテキストノードを削除する:

1. **text()** ノードのコネクタをクリックして、Del キーを押下することで接続を削除します。



2. 出力タブをクリックしてマッピングの結果を確認します。

```
<Name>Nanonull, Inc.</Name>
<Desc>
<para>
<bold> Vereno</bold>
<italic>multi-core processors.</italic>
<bold>Nano-grid.</bold>
<italic>offshore</italic>
</para>
<para/>
</Desc>
</Office>
<Office>
<Name>Nanonull Europe, AG</Name>
<Desc>
<para>
<italic>Europe</italic>
<bold> five research scientists </bold>
</para>
</Desc>
```

結果:

- para 要素内にある全てのテキストノードが削除されました。
- それまでマッピングされていた bold ならびに italic 要素のテキストはまだ残っています。
- bold ならびに italic アイテムのシーケンスもソース XML ファイルと同様のものとなっています。

処理情報とコメントのマッピング:

1. コンテンツ接続を右クリックして、**プロパティ**を選択します。
2. **ソースドライブ（混合コンテンツ）**から、**処理命令をマップ**と**コメントをマップ**のチェックボックスを選択します。

4.2.2.2 混合コンテンツのサンプル

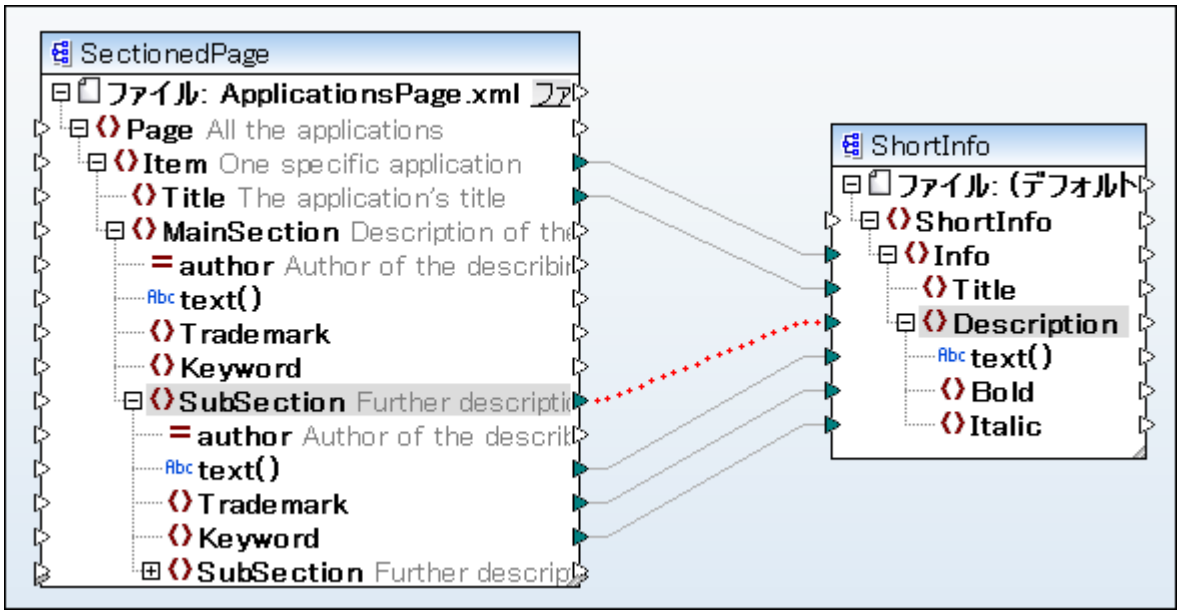
以下のサンプルでは **...\MapForceExamples** フォルダー以下にある **ShortApplicationInfo.mfd** を使用します。

この例で使用する XML ソースファイルの一部を示します:

```
<Page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SectionedPage.xsd">
    <Item>
        <Title>XMLSpy</Title>
        <MainSection author="altova">
            Altova <Trademark>XMLSpy</Trademark>
            <SubSection>Altova <Trademark>XMLSpy</Trademark> 2005 Enter
is the industry standard <Keyword>XML</Keyword> development environment
editing, debugging and transforming all <Keyword>XML</Keyword> technolo
automatically generating runtime code in multiple programming languages
        </MainSection>
    </Item>
```

以下にマッピングを示します。以下の点に注意してください:

- 「SubSection」アイテムのコネクタは混合コンテンツで、ターゲット XML スキーマの詳細 アイテムへマッピングされています。
- text() ノード同士がマッピングされています。
- Trademark テキストがターゲットの Bold アイテムへマッピングされています。
- Keyword テキストがターゲットの Italic アイテムへマッピングされています。



マッピングの結果

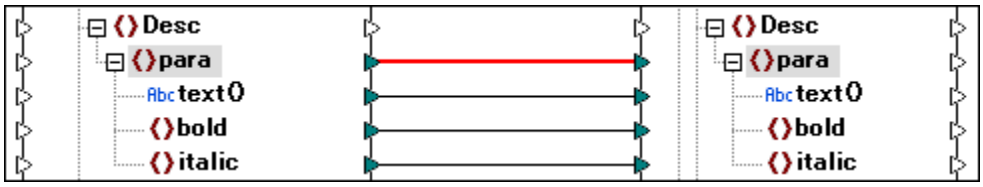
各説明文の混合コンテンツテキストが正しくマッピングされ、テキストだけではなく、bold と italic タグコンテンツが XML ソースファイルにある順序の通り、正しくマッピングされます。

```
<?xml version="1.0" encoding="UTF-8"?>
<ShortInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="
C:/PROGRA~1/Altova/MapForce2005/MapForceExamples/ShortInfo.xsd">
  <Info>
    <Title>XMLSpy</Title>
    <Description>Altova <Bold>XMLSpy</Bold> 2005 Enterprise Edition is the industry standard
<Italic>XML</Italic> development environment for modeling, editing, debugging and transforming
all <Italic>XML</Italic> technologies, then automatically generating runtime code in multiple
programming languages.</Description>
  </Info>
```

4.2.2.3 混合コンテンツアイテムに対して標準接続を使用する

上記のように、ソース優先の(標準ではない) 接続は、通常、複合コンテンツノードからデータをマッピングする際に使用されます。それ以外の場合、結果する出力は、希望する結果ではない可能性があります。通常の標準(ターゲット優先)接続を使用して、複合コンテンツノードからデータをマッピングする場合の結果を確認するには、以下の手順を踏んでください:

1. <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥ フォルダーからマッピング **Tut-OrgChart.mfd** を開きます。
2. ソース内の `para` ノード とターゲット内の `para` ノード 間に接続を作成します。MapForce が接続をソース優先と定義するかを問うメッセージが表示されます。「**いいえ**」をクリックします (これにより、MapForce による提案を破棄し、通常の接続を作成します)。



メモ 接続が上で表示されているように標準(ターゲット優先)であることを確認してください。すべてコピー 接続が自動的に作成されると、接続を右クリックして、コンテキストメニューから「**ターゲット優先**」(**標準**)を選択します。

3. 「**出力**」タブをクリックして、マッピングの結果を表示します。

```
<Office>
  <Name>Nanonull, Inc.</Name>
  <Desc>
    <para>The company was established inin 1995. Nanonull develops nanoelectronic techn
unveiling of the first prototype The company hopes to expand its operations to drive down opera
      <bold> Vereno</bold>
      <bold>Nano-grid.</bold>
      <italic>multi-core processors.</italic>
      <italic>offshore</italic>
    </para>
    <para>White papers and further information will be made available in the near future.
    </para>
  </Desc>
</Office>
<Office>
```

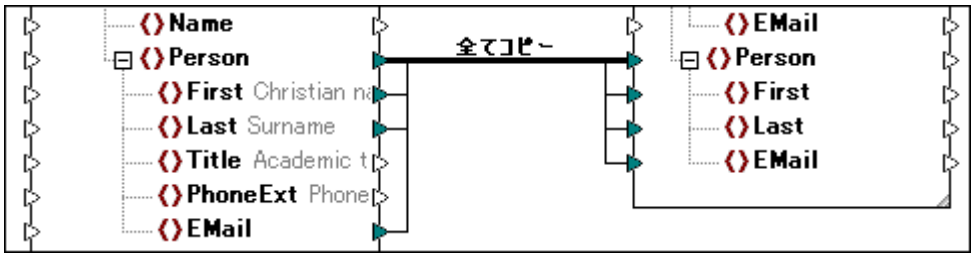
上に示されるように、混合コンテンツのアイテムに対して標準マッピングを適用すると、以下のような結果となります:

- text() ソースアイテムのコンテンツがターゲットにコピーされます。しかしながら、出力内の子ノード（この場合、bold と italic ）のシーケンスは、ターゲット XML スキーマ内のシーケンスに対応します。すなわち、子ノード（この場合、bold と italic）は、複合コンテンツノードテキストの後に表示されます。
- それぞれのpara 要素に対して、MapForce は、 text() ノード を最初にマップし、すべてのbold アイテムと、最後に全てのitalic アイテムをマップします。この結果、複数のbold とitalic アイテムは、互いに積み上げられて表示されます。各アイテムのコンテンツは、接続がソースから存在する場合に表示されます。

4.2.3　全てコピー接続

全てコピーという種類の接続方法では、作業領域にあるマッピングをより単純なものとして、ソースならびにターゲットコンポーネントにある全ての同一アイテムを自動的に接続します。つまり、ソースならびにターゲットコンポーネントにおける型に従うかたちで、ソースならびにターゲットアイテムの型が同一の場合か、またはターゲットの型がxs:anyType の場合、該当する全てのソースアイテムがターゲットコンポーネントへコピーされます。

ソースならびにターゲットの型が同一で無い場合、かつターゲットの型がxs:anyType で無い場合、ソースデータは同一の階層にある同一名のアイテムにマッピングされます。ターゲットアイテムの名前が異なる場合、そのターゲットアイテムに対するマッピングは作成されません。



全てコピー接続

マッピング上の2 つの構造間にマッピングを描く場合シースとターゲット構造に互換性があると検出した場合、MapForce は、「全てコピー」接続を自動的に作成します（これは、両方の構造が同じ型、またはターゲットがソース型のサブ型であることを意味します）。マッピングのランタイムでは、子を含む全てのインスタンスデータはソースからターゲットに再帰的にコピーされます。

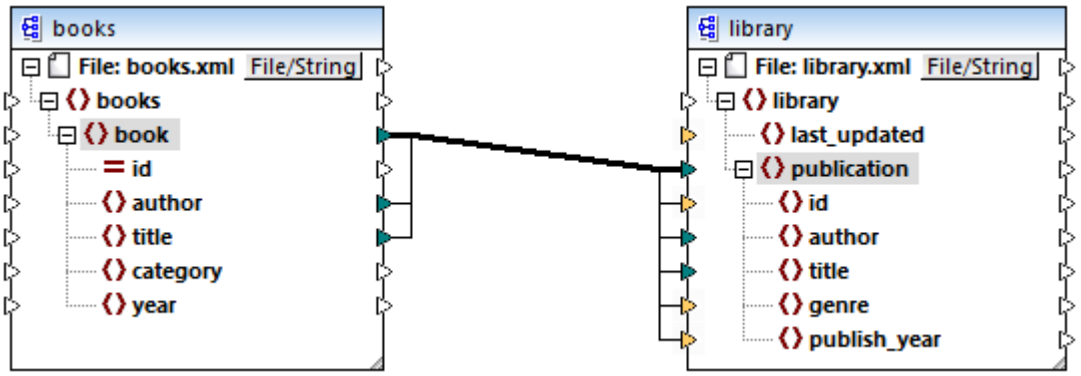
「全てコピー」接続を手動で作成するには、子アイテムを持つ同類の2 つのノード間の既存の接続を右クリックし、コンテキストメニューから全てコピー（子アイテムのコピー）を選択します。

次に注意してください：

- 「全てコピー」接続が意味を成さない、またはサポートされていないコンテキストでは、この種類の接続を手動で作成することはできません。
- 「全てコピー」接続をXML/スキーマコンポーネントのroot 要素に対して作成することはできません。
- ユーザー定義関数のスキーマとパラメーター間に「全てコピー」接続を作成する場合、2 つのコンポーネントは同じスキーマをベースにしている必要があります。しかしながら、同じルート要素を持つ必要はありません。

「全てコピー」接続の例は次のステップで作成することができます:

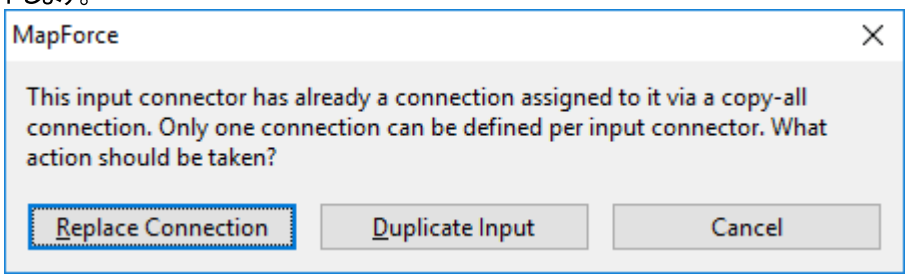
1. 新規マッピングを作成します。
2. 「挿入」メニューからXML スキーマ/ファイルをクリックし、フォルダー<**マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial** 内にある**books.xml** ファイルを参照してください。
3. 「挿入」メニューからXML スキーマ/ファイルをクリックし、フォルダー<**マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial** 内にある**library.xsd** ファイルを参照してください。
4. 「books」コンポーネントのbook ノード と「library」コンポーネントのpublication ノード間にマッピング接続を描きます 。
5. 新規の接続を右クリックして、コンテキストメニューから 全てコピー（子アイテムのコピー）を選択します。



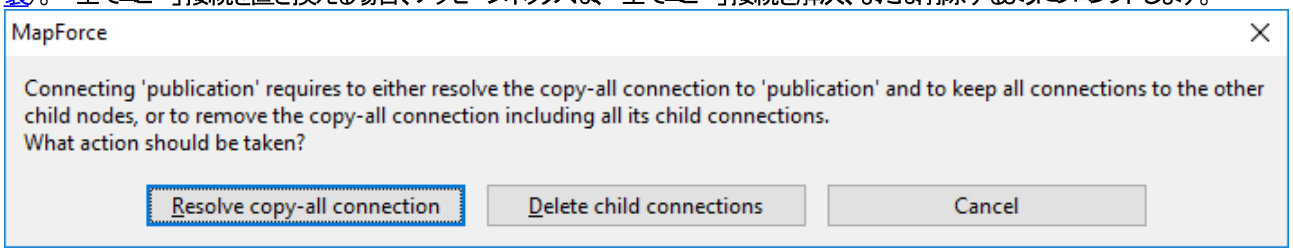
ソースとターゲット 構造間に差異が存在する場合、「全てコピー」接続が列挙します。マッピングのランタイムでは、（要素と属性などの）ソースアイテムは、ターゲット型内に存在するものだけがコピーされます。これは再帰的に繰り返されます。

例えば、上のマッピングでは、2つの構造（`author` と `title`）の2つの子アイテムが同一です、ですから、ターゲットにマップすることができます。アイテム `id` は、ソース属性内の属性とターゲット内の要素であるため、自動的には含まれません。マップする必要がある場合、例えば、`category` から `genre`, へマップする場合、異なるアイテムのため、「全てコピー」接続を使用することはできません。

入力コネクタが（コンポーネントの横の小さい三角で表示されている）「全てコピー」接続を受け取る場合、他の接続を受け入れることはできません。上のサンプルでは、`category` と `genre` 間の接続を作成する場合、MapForce 置換え、または入力の複製を行うようにプロンプトします。



ここでは必要ありませんが、入力の複製は、ターゲットが1つ以上の入力からデータを受け入れる場合に役に立ちます（次を参照: 入力の複製）。「全てコピー」接続を置き換える場合、メッセージボックスは、「全てコピー」接続を解決、または削除するようにプロンプトします。

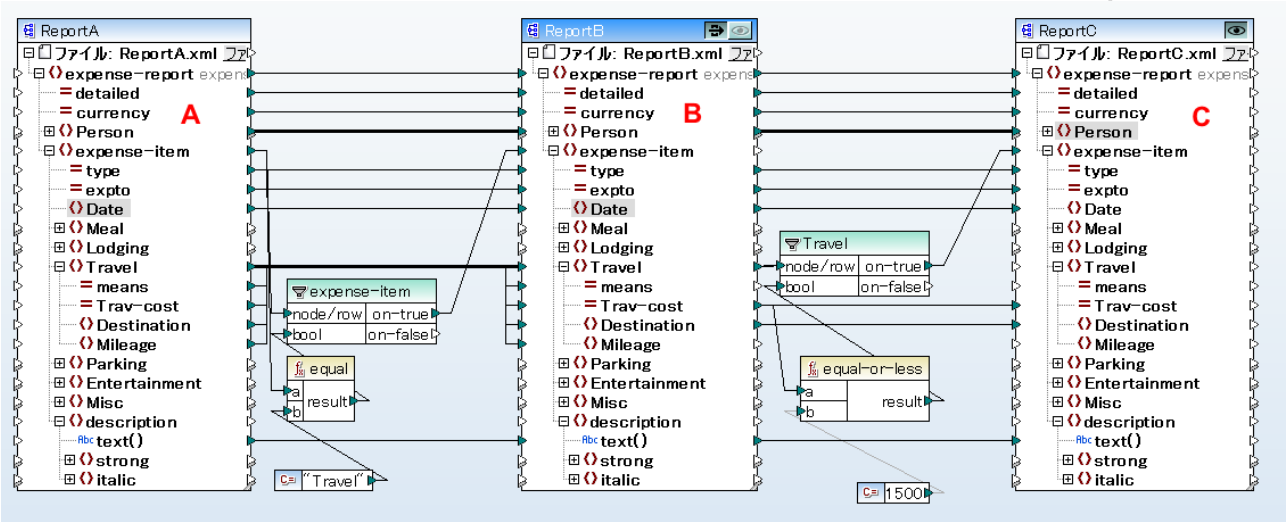


「全てコピー」接続を対応する子アイテムの標準のターゲット優先 接続と置換えるには、**全てコピー 接続を解決する** をクリックします。「全てコピー」接続を削除するには、**「子接続の削除」**をクリックします。

4.3 チェーンマッピング

MapForce ではチェーン構造により複数のコンポーネントを含むようなマッピングがサポートされます。チェーン構造のマッピングとは、少なくとも 1 つのコンポーネントがソースかつターゲットコンポーネントとして機能するマッピングのことです。このようなコンポーネントの出力は、それ以降のマッピングで入力として使用されます。このようなマッピングのことを「中間」コンポーネントと呼びます。

例えば、下に示されるマッピングは、(XML 書式の) 2 つの段階で処理される経費報告書を表示しています。A から B へのマッピングの部分は、「Travel」とマークされる経費のみをフィルターします。B から C へのマッピングの部分は、1500 以下の「Travel」経費のみをフィルターします。コンポーネント B は、入力と出力接続の両方が存在するため、「中間」コンポーネントです。このマッピングは、次のパスで見つけることができます: <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥**ChainedReports.mfd**。



ChainedReports.mfd

「パススルー」と呼ばれる機能がチェーンマッピングに搭載されています。「パススルー」は、出力ウィンドウ内でチェーンマッピングの各段階による出力を確認することのできるプレビュー機能です。例えば、上のマッピングでは、A から B への、および、B から C への、結果する XML 出力をプレビューして、保存することができます。

メモ　XML、CSV、TXT ファイルといったファイルをベースにした「中間」コンポーネントだけが「パススルー」機能を搭載します。データベースコンポーネントを中間コンポーネントとして使用することはできますが、パススルーコンポーネントは表示されません。プレビューやコードの生成を行うたびに、中間コンポーネントは再度生成されますが、生成を行う前に一度削除しなければならないため、データベースにてこのような操作を行うことはできません。

MapForce Server または、生成されたコードによりマッピングが実行される場合、マッピングのチェーン全体が実行されます。チェイン内の各ステップで必要な出力ファイルをマッピングは生成し、マッピングのチェーンのステップの出力が次のマッピングステップへ入力として転送されます。

中間コンポーネントのために、動的なファイル名を生成することもできます。コンポーネントが対応して構成されていることを条件に、マッピングから "File:" アイテムへの接続を受け入れることができます。更に詳しい情報に関しては、次を参照してください: 複数の入力、および、出力ファイルを動的に処理する。

プレビューボタン

コンポーネント B ならびに C にはプレビューボタンが表示されます。この機能を使用することにより MapForce 内で B の中間マッピング結果のプレビューだけではなく、チェーン構造の最終結果をプレビューすることができます。対応するコンポーネントのプレビューボタンをクリックし、出力ボタンをクリックすることでマッピング結果を確認することができます。

パススルーボタンが有効化されている中間コンポーネントをプレビューすることはできません。同時にデータをプレビューし、パススルーすることは意味を成さないため、プレビューボタンは、自動的に無効化されます。このようなコンポーネントの出力を確認するためには、パススルーボタンをクリックして、無効化してから、プレビューボタンをクリックします。

パススルーボタン

中間コンポーネント B には、更に「パススルー」という名前のボタンがコンポーネントのタイトルバーに表示されます。

パススルーボタンが有効  になっている場合、コンポーネント A からコンポーネント B へ、そしてコンポーネント C へというかたちで全てのデータをプレビューウィンドウへマッピングします。2 つの結果が生成されます。

- マッピングコンポーネント A の結果が中間コンポーネント B へマッピングされます。
- 中間コンポーネント B の結果がターゲットコンポーネント C へマッピングされます。

パススルーボタンが無効  になっている場合、マッピングチェーンの一部だけが実行されます。生成されるマッピング結果は、(コンポーネント B と C のうち) どちらのプレビューボタンがアクティブになっているかに依存します。

- コンポーネント B のプレビューボタンがアクティブになっている場合、コンポーネント A から B へのマッピング結果が生成されます。マッピングチェーンはコンポーネント B で停止し、プレビューを行うのにコンポーネント C は全く使用されません。
- コンポーネント C のプレビューボタンがアクティブになっている場合、中間コンポーネント B から C へのマッピング結果が生成されます。パススルー機能が無効になっているため、コンポーネント B に対する自動チェーンが中断されます。マッピング右側にあるチェーンだけが実行され、コンポーネント A は使用されません。

> "パススルー" ボタンが無効化されている場合、中間コンポーネントは、"入力 XML ファイル" と "出力 XML ファイル" フィールド内で同一のファイル名を持つ必要があります。これにより、B と C の間のマッピングの部分をプレビューする際に、出力として生成されるファイルが A と B の間のマッピングを入力と使用することができます。また、生成されたコード内で、または MapForce Server 実行内で、マッピングのチェーンが壊れないことを保証することができます。

前記のように、マッピング MapForce Server、または生成されたコードにより実行された場合、すべてのコンポーネントの出力が生成されます。この場合、コンポーネント B のパススルーボタン、および現在選択されているプレビューコンポーネントは破棄されます。上のマッピングを例にして、2 つの結果ファイルは以下のように生成されます:

1. A から B へのマッピングコンポーネントから結果する出力ファイル
2. B から C へのマッピングコンポーネントから結果する出力ファイル

次のセクションである 例: パススルーが有効な場合 と 例: パススルーが無効な場合 は、パススルーボタンが有効化、または無効化されている場合に、どのようにソースデータが転送されるかについて説明しています。

4.3.1 例: パススルーが有効な場合

このサンプルで使用されているマッピング (**ChainedReports.mfd**) は、**<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial** フォルダー内にあります。このマッピングは、旅費の経費を含む **ReportA.xml** という名前の XML ファイルを処理します。簡素化のために、名前空間宣言と `expense-item` 要素の一部は省略されています:

```
<?xml version="1.0" encoding="UTF-8"?>
<expense-report currency="USD" detailed="true">
   <Person>
      <First>Fred</First>
      <Last>Landis</Last>
      <Title>Project Manager</Title>
```

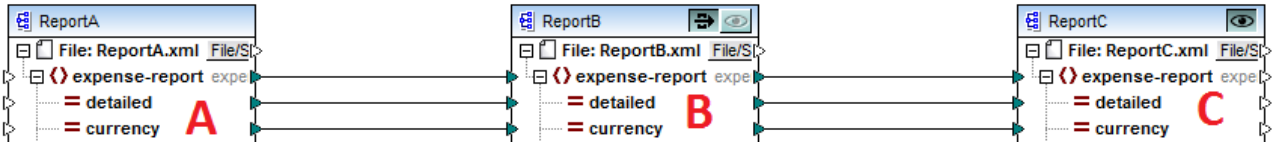
```
      <Phone>123-456-78</Phone>
      <Email>f.landis@nanonull.com</Email>
   </Person>
   <expense-item type="Travel" expto="Development">
      <Date>2003-01-02</Date>
      <Travel Trav-cost="337.88">
         <Destination/>
      </Travel>
      <description>Biz jet</description>
   </expense-item>
   <expense-item type="Lodging" expto="Sales">
      <Date>2003-01-01</Date>
      <Lodging Lodge-cost="121.2">
         <Location/>
      </Lodging>
      <description>Motel mania</description>
   </expense-item>
   <expense-item type="Travel" expto="Marketing">
      <Date>2003-02-02</Date>
      <Travel Trav-cost="2000">
         <Destination/>
      </Travel>
      <description>Hong Kong</description>
   </expense-item>
</expense-report>
```

ReportA.xml

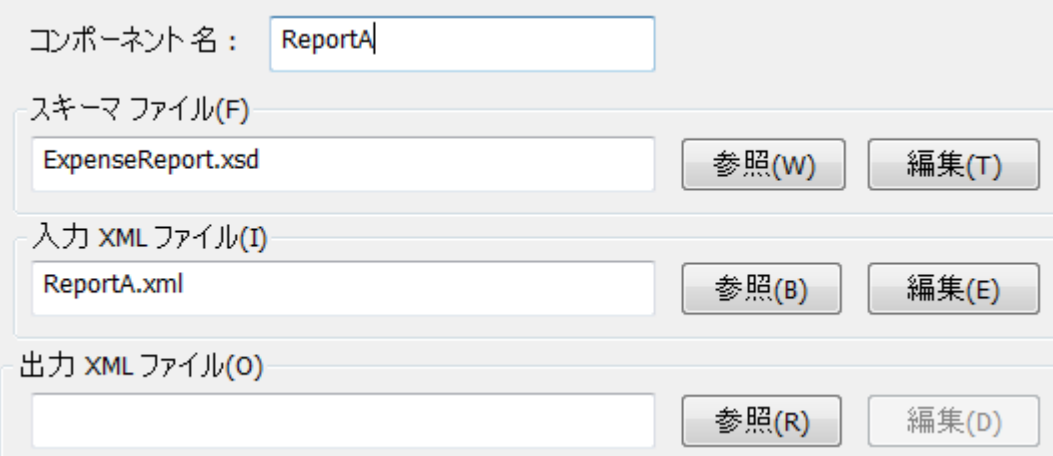
マッピングの目的は、上のファイルを基にして、2つの更なるレポートを生成することです:

- **ReportB.xml** – このレポートは、"Travel" 型の旅費のみを含みます。
- **ReportC.xml** – このレポートは、1500 を超えない "Travel" 型の旅費のみを含みます。

この目的を達成するためには、マッピング（コンポーネント B）の中間コンポーネントでは、表示されているとおりパススルーボタン が有効化されています。これによりマッピングは段階的に実行されます: A から B は、そして、B から C へ実行されます。中間コンポーネントにより作成される出力は、B と C の間のマッピングのために使用されます。



マッピングのチェーン内の各段階で生成される出力ファイルは、各コンポーネントの設定により決定されます。（コンポーネント 設定を開くには、右クリックし、コンテキストメニューから **プロパティ** を選択します）。具体的には、最初のコンポーネントは、**ReportA.xml** という名前の XML ファイルからデータを読み取るように構成されています。ソースコンポーネントであるからであり、出力 XML ファイル フィールドは関連性がないため、空白のままです。



ソースコンポーネントの設定

下に示されるように、2番目のコンポーネント（**ReportB**）は、**ReportB.xml** という名前の出力ファイルを作成するように構成されています。**入力 XML ファイル** フィールドは、灰色で表示されています。（このサンプル内で示されているように）パススルーが有効化されている場合、中間コンポーネントの **入力 XML ファイル** フィールドは、自動的に無効化されます。マッピングを実行するために入力ファイル名が存在する必要はなく、マッピング内のこの段階で生成される出力は、一時的なファイル内に保管され、マッピングの後の段階で再利用することができます。また、（下で示されるように）**出力 XML ファイル**が定義されている場合、中間出力ファイルのファイル名のために使用されます。**出力 XML ファイル** が定義されていない場合、デフォルトのファイル名が自動的に使用されます。

中間コンポーネントの設定

最後に、3番目のコンポーネントは、**ReportC.xml** という名前の出力ファイルを生成するように構成されています。これはターゲットコンポーネントであるため、**入力 XML ファイルフィールド** には関連性はありません。

ターゲット コンポーネントの設定

マッピングウィンドウ内の出力タブをクリックして、マッピングをプレビューする場合は、2つのファイルが出力に表示されます:

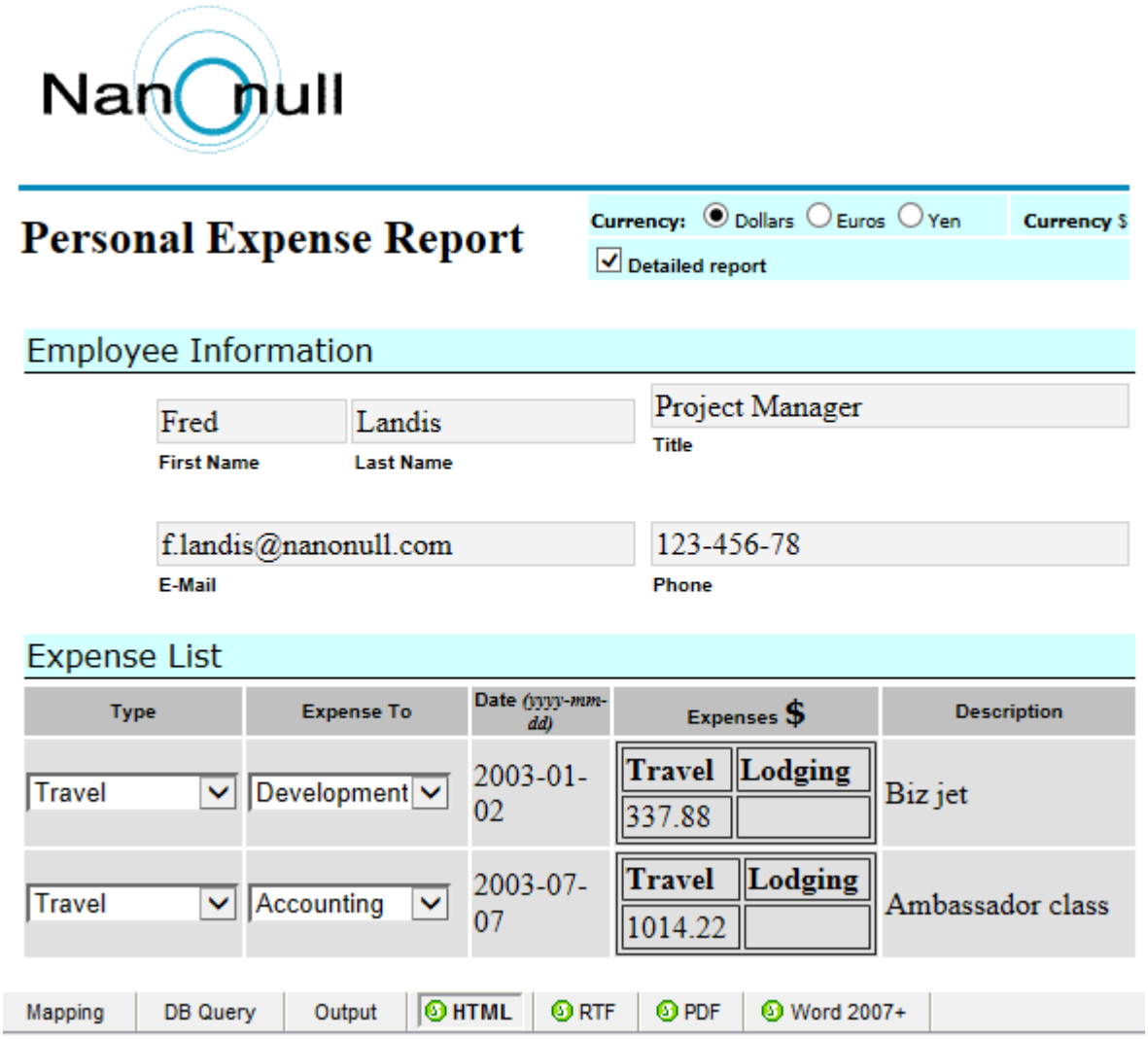
1. **ReportB.xml** は、A から B へのマッピングの結果を表します。
2. **ReportC.xml** は、B から C へのマッピングの結果を表します。

2件の生成された出力ファイルから選択するには、出力ウィンドウ内に表示されているファイルを矢印ボタンをクリックして、またはドロップダウンリストからエントリを選択します。

生成された出力ファイル

MapForce によりマッピングが実行される場合、(「ツール | オプション | 一般」から構成することのできる) 設定「出力ファイルに直接書き込む」により、中間ファイルが一時的なファイル、または物理的ファイルとして保存されるかを決定します。MapForce 内でマッピングが直接プレビューされる場合のみ、これは妥当であることに注意してください。このマッピングが MapForce Server により、または生成されたコードにより実行された場合、実際のファイルは、マッピングのチェーンの各段階で生成されます。

StyleVision がインストールされている場合、そして、StyleVision Power Stylesheet (SPS) ファイルがターゲット コンポーネントに (このサンプル内で示されているように) 割り当てられている場合、最後のマッピング出力を HTML、RTF ファイルとして確認 (保存) することができます。MapForce 内でこの出力を表示するには、対応する名前を持つタブをクリックしてください。

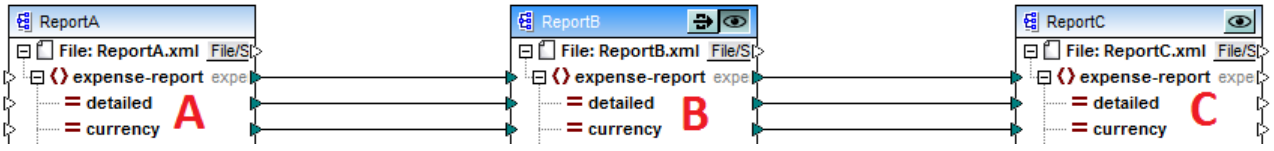


生成されたHTML 出力

マッピングのチェーンの最後のターゲットコンポーネントの出力のみが表示されることにご注意してください。中間コンポーネントのStyleVision 出力を表示するには、パススルーボタンを無効化して中間コンポーネントを（ 例: パススルーが無効な場合 内で表示されているとおり）プレビューする必要があります。

4.3.2 例: パススルーが無効な場合

このサンプル（**ChainedReports.mfd**）で使用されているマッピングは、**<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial** フォルダー内で見つけることができます。このサンプルは、中間コンポーネントのパススルーボタン が無効化されている場合、出力がどのように生成されるかを説明しています。



例: パススルーが有効な場合 で説明されているように、マッピングの目的は、2件の個別のレポートを生成することです。前のサンプルで、パススルーボタン は有効化されており、両方のレポートは期待されるとおり生成され、**出力** タブ内で確認することができました。しかしながら、レポートの1件(**ReportB.xml** または **ReportC.xml**)のみをプレビューする場合、パススルーボタン ( ) は無効化されている必要があります。更に具体的には、パススルーボタンを無効化することは、次の目的を達成するためには有効な場合があります:

- B から C へのマッピングを無視して、A から B により生成された部分をプレビューする。
- A から B へのマッピングを無視して、B から C により生成された部分をプレビューする。

上に示されるように パススルーボタンを無効化すると、**ReportB** または **ReportC** を選択することができます(両方に ボタンが表示されます)。

パススルーボタンの無効化により、中間コンポーネントがどの入力ファイルを読み取るかを選択することができます。多くの場合、これは**出力 XML ファイル** フィールド(このサンプル内で示されているように)内で定義されたファイルと同じです。

中間コンポーネントの設定

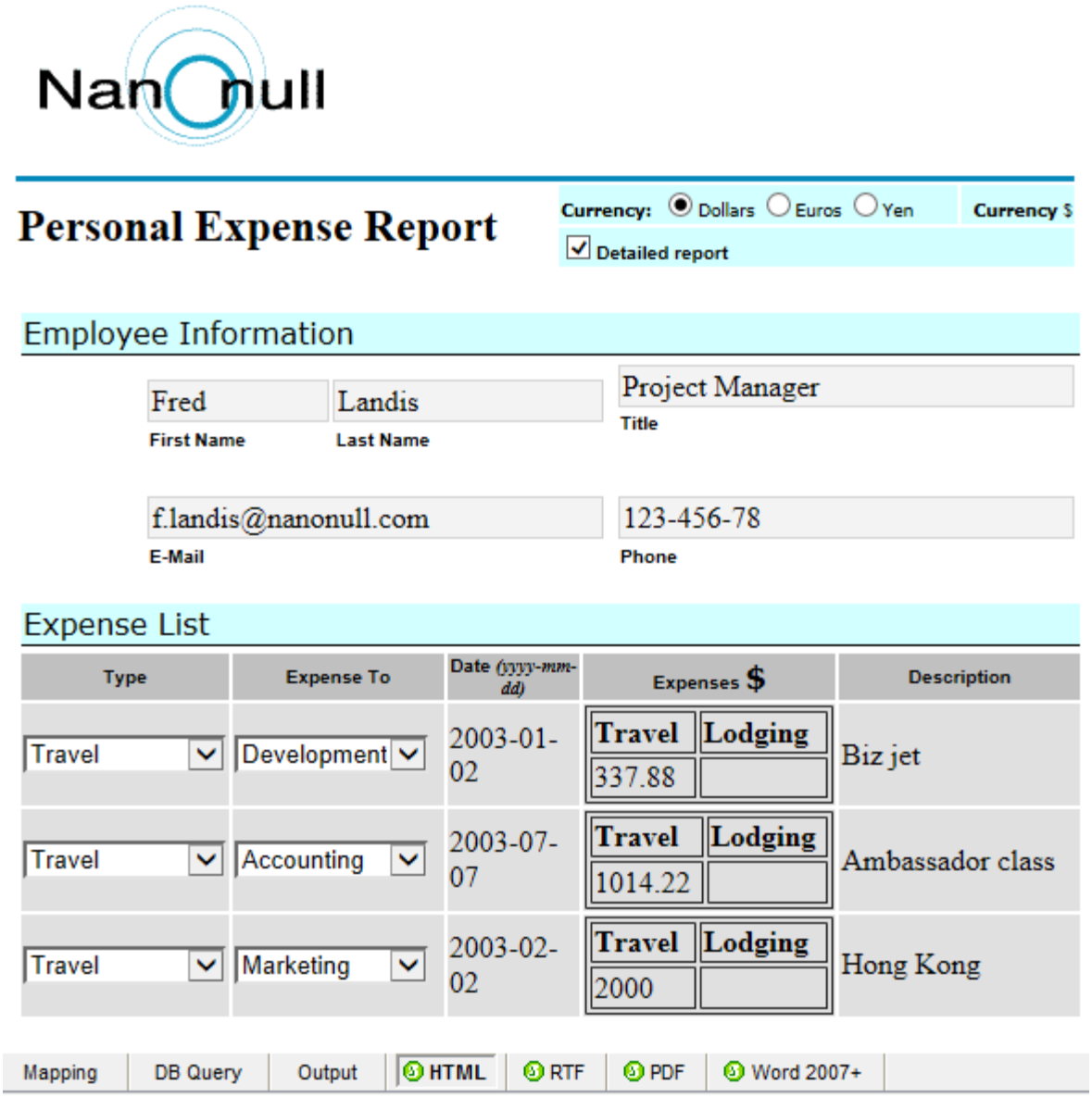
中間コンポーネント上に同じ入力と出力 ファイルが存在することは、マッピングからコードを生成する場合、またはマッピングを MapForce Server 上で実行する場合、特に重要です。前記のように、この環境は、マッピングチェーン内の各コンポーネントにより作成される出力により生成されます。ですから、中間コンポーネントが処理するファイル(この場合は **ReportB.xml**) を受け取り、異なるファイルを検索するよりも、同じファイルを次のマッピングに送ることは意味があります。(パススルーボタンが無効化されており) 中間コンポーネント上で同じ入力と出力ファイル名が存在しないと、生成されたコード内で、または MapForce Server 実行中に「システムが指定されたファイルを見つけることができません」などのエラーが発生する場合があります。

3番目のコンポーネント(**ReportC**) のプレビューボタン をクリックし、MapForce 内でマッピングをプレビューしようと試みると、エラーが発生します。上記の設定に従い、**ReportB.xml** という名前のファイルが入力として期待されるため、これは予想されている振る舞いです。しかしながら、マッピングは、(パススルーボタンが有効化されておらず、B から C へのマッピングの部分のみが実行されているため) またこの名前のファイルは生成されていません。この問題は簡単に解決することができます:

1. 中間コンポーネントのプレビューボタンをクリックします。
2. マッピングをプレビューするために、**出力** タブをクリックします。
3. マッピング(<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥) と同じフォルダー内に結果出力ファイルを **ReportB.xml** として保存します。

3番目のコンポーネント(**ReportC**) のプレビューボタンをクリックすると、エラーは表示されなくなります。

パススルーボタンが無効化されている場合、関連した StyleVision Power StyleSheet (SPS) ファイルを持つ各コンポーネントのための StyleVision により生成された出力をプレビューすることができます。特に、(最後のレポートに加え) 中間 レポートの HTML バージョンを確認することもできます:



中間コンポーネントのHTML 出力

4.4 複数の入力または出力ファイルを動的に処理

マッピングが実行される際、MapForce に複数のファイルを処理するように構成することができます（例: ディレクトリ内の全てのファイル）。この機能を使用して、以下のようなタスクを解決することができます:

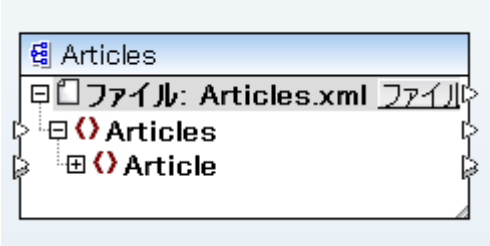
- マッピングに処理する入力ファイルのリストを与えます。
- 単一の出力ファイルの代わりにファイルのリストのマッピング出力を生成します。
- 入力と出力ファイル名がランタイムで定義されている箇所のマッピングアプリケーションを生成します。
- ファイルのセットを他のフォーマットに変換します。
- 大きなファイルを小さなパーツに分割します。
- 複数のファイルを1つの大きなファイルにマージする

MapForce コンポーネントに複数のファイルを処理するように以下の方法で構成することができます:

- コンポーネント 設定で必要とされる入力または出力ファイルに固定されたファイル名の代わりにワイルドカード文字を使用してパスを与えます（コンポーネント設定を変更するを参照）。 すなわち、ワールドカード * と? をコンポーネント設定ダイアログボックスに入力し、MapForce は、マッピング実行される際に対応するパスを解決します。
- パスを動的に提供する、コンポーネント シーケンスルートノードに接続します。（例: `replace-fileext` 関数の結果）。マッピングが実行されると、MapForce は、全ての入力ファイルを動的に読み込み、また全ての出力ファイルを動的に生成します。

目的により異なりますが、同じマッピングで1つまたは両方のアプローチを取ることができます。しかしながら、同じコンポーネント上で同時に両方のアプローチを使用することは、あまり意味をなしません。特定のコンポーネントのために、MapForce がどのアプローチを使用するかを命令するには、コンポーネントのルートノードの横の「**ファイル**」（ File ）または「**ファイル/文字列**」（ File/String ）ボタンをクリックします。このボタンにより以下の振る舞いを指定することができます:

コンポーネント設定からファイル名を使用する	コンポーネントが1つまたは複数のインスタンスファイルを処理する場合、このオプションはMapForce にコンポーネント設定ダイアログボックスで定義されたファイルを処理するように命令します。 このオプションを選択すると、ルートノードには入力コネクタが与えられません。  コンポーネント設定ダイアログボックス内でまだ入力または出力ファイルを指定していない場合、ルートノードの名前は**ファイル: (デフォルト)**です。それ以外の場合、ルートノードは、出力ファイル名、セミコロン（;）が後についた入力ファイル名を表示します。 入力ファイル名が出力ファイルと同じ場合、ルートノードの名前として表示されます。

	 *このオプションまたは「マッピングから与えられた動的なファイル名を使用する」オプション*を選択することができます。
マッピングから与えられた動的なファイル名を使用する	このオプションは、コンポーネントのルートノードの値に接続することにより、マッピングエリアで定義するファイル名を処理するようにMapForce に命令します。 このオプションを選択すると、ルートノードがマッピングの実行中動的に処理されるファイル名を与える入力コネクタの値に接続します。コンポーネント設定ダイアログボックス内に定義されたファイル名を持つ場合、これらの値は無視されます。 このオプションが選択された場合、ルートノードの名前は以下として表示されます **ファイル: <動的>**。  *コンポーネント設定からファイル名を使用する*オプションと共にこのオプションは相互排他的です。

複数の入力または出力ファイルを以下のコンポーネントのために定義することができます:

- XML ファイル
- テキストファイル(CSV*, FLF* ファイルとFlexText** ファイル)
- EDI ドキュメント**
- Excel スプレッドシート**
- XBRL ドキュメント**
- JSON ファイル**
- プロトコルバッファーファイル**

** MapForce Professional Edition 必須*
*** MapForce Enterprise Edition 必須*

以下のテーブルは、 MapForce 言語内での動的な入力と出力ファイルおよびワイルドカードへのサポートについて説明されています。

ターゲット言語	動的入力ファイル名	入力ファイル名のためにサポートされているワイルドカード	動的な出力ファイル名

XSLT 1.0	*	XSLT 1.0 によりサポートされていません	XSLT 1.0 によりサポートされていません
XSLT 2.0	*	*(1)	*
XSLT 3.0		*(1)	*
C++	*	*	*
C#	*	*	*
Java	*	*	*
BUILT-IN	*	*	*

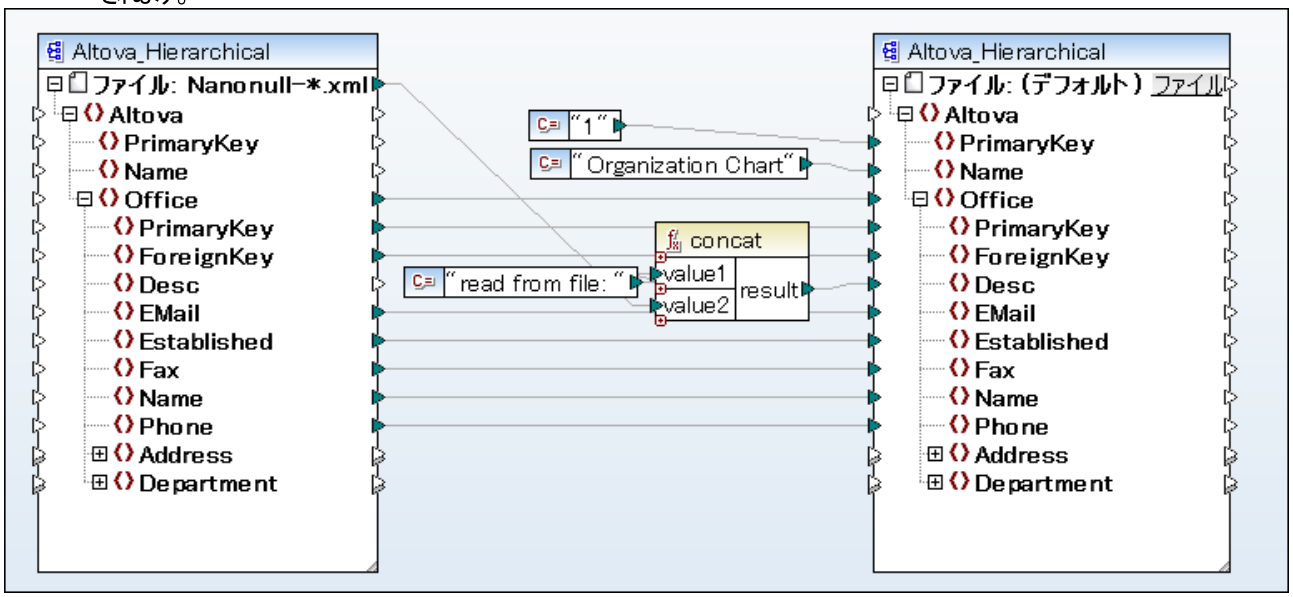
レジェンド:

*	サポートされている
(1)	XSLT 2.0、XSLT 3.0 とXQuery は `fn:collection` 関数を使用します。Altova XSLT 2.0、XSLT 3.0 と XQuery エンジン内の実装はワイルドカードを解決します。他のエンジンは異なる振る舞いをする可能性があります。

4.4.1 複数の入力ファイルを単一の出力ファイルにマップする

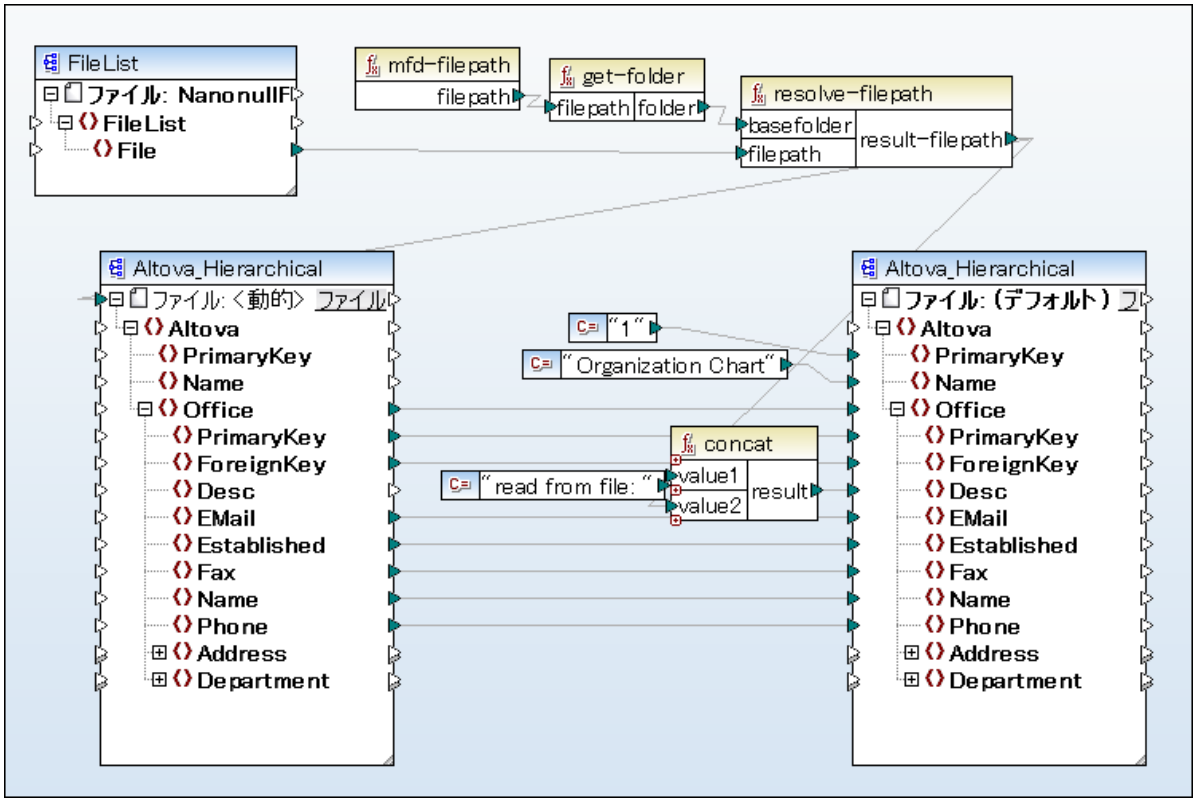
複数の入力ファイルを処理するには、以下を行います:

- コンポーネント設定ダイアログボックスに入力ファイルとしてワイルドカード(*または?)を持つのファイルパスを入力します。一致する全てのファイルが処理されます。下のサンプルは、入力 XML ファイルフィールド内でマッピング入力として名前が"Nanonull-"で開始する全てのファイルを提供する* ワイルドカード文字を使用します。ターゲットコンポーネントに動的コネクタが存在せず、ソースコンポーネントはワイルドカード * を使用して複数のファイルにアクセスしているため、複数の入力ファイルは単一の出力ファイルとしてマージされます。ターゲットコンポーネント内のルートノードの名前は **File: <default>** です。これはコンポーネント設定ダイアログボックス内で出力ファイルパスが定義されていないことを示します。複数のソースファイルは、ですから、ターゲットドキュメント内で追加されます。



MergeMultipleFiles.mfd (MapForce Basic Edition)

- ソースコンポーネントの*ファイル*ノードに文字列の*シーケンス*をマップします。シーケンス内の各文字列が 1 つのファイル名を表します。文字列は自動的に解決されるワイルドカードを含むこともできます。XML ファイルなどのコンポーネントがファイル名の*シーケンス*を提供します。

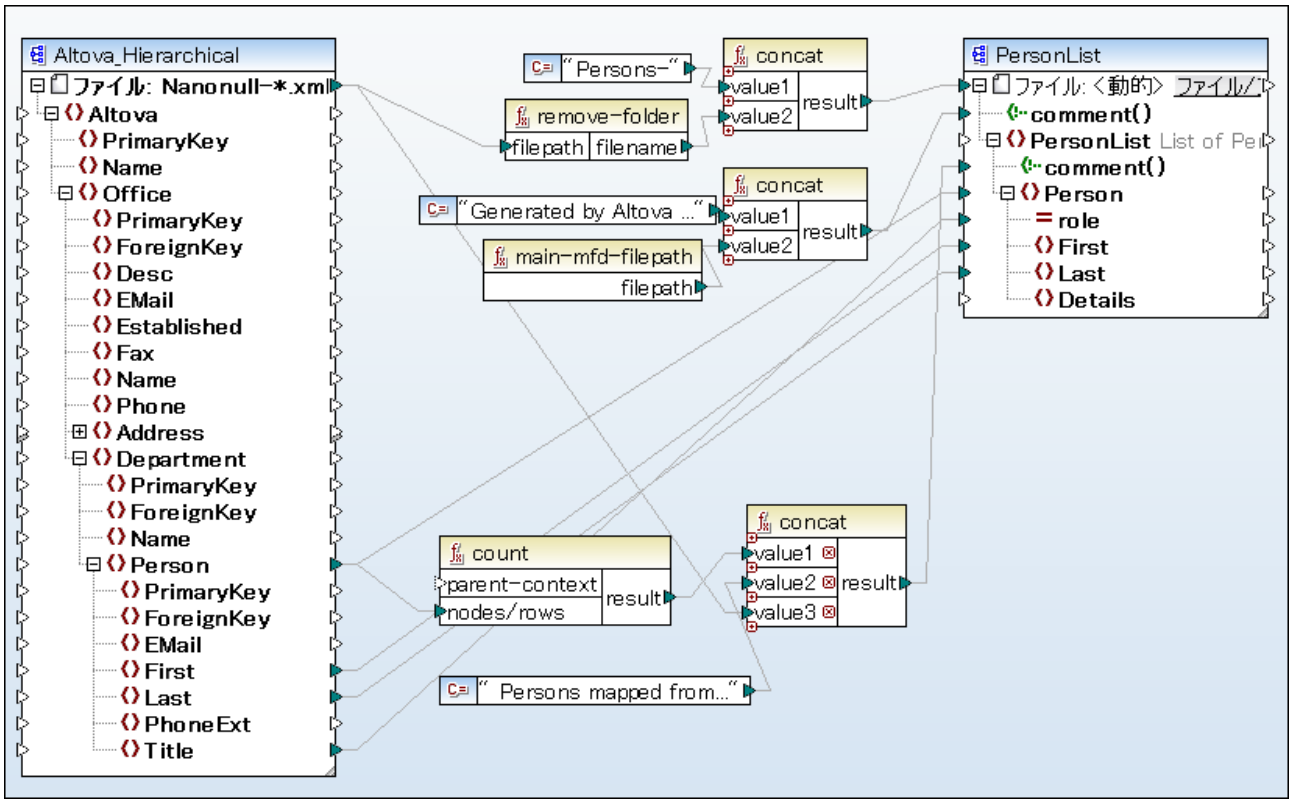


MergeMultipleFiles_List.mfd (MapForce Basic Edition)

4.4.2 複数の入力ファイルを複数の出力ファイルにマッピングする

複数のファイルを複数のターゲットファイルにマップするには、一意の出力ファイル名を生成する必要があります。出力ファイル名が入力データ内の文字列から派生する場合があり、他の場合、入力ファイル名から出力ファイル名を派生させることが役にたつ場合があります。例: ファイル拡張子の変更により。

以下のマッピングでは、`concat` 関数を使用してプレフィックス "Persons-" を追加し、出力ファイル名を入力ファイル名から派生させます。



MultipleInputToMultipleOutputFiles.mfd (MapForce Basic Edition)

メモ　処理関数を使用することなく、入力と出力ルートノードを直接接続することを回避します。これを行うことによりマッピングが実行される際、入力ファイルが上書きされます。上記の`concat` 関数などを使用して、出力ファイル名を変更することができます。

メニューオプション「**ファイル | マッピング設定**」によりマッピングで使用されるファイル パス設定をグローバルに定義することができます（マッピング設定の変更を参照）。

4.4.3　ファイル名をマッピングパラメーターとして提供する

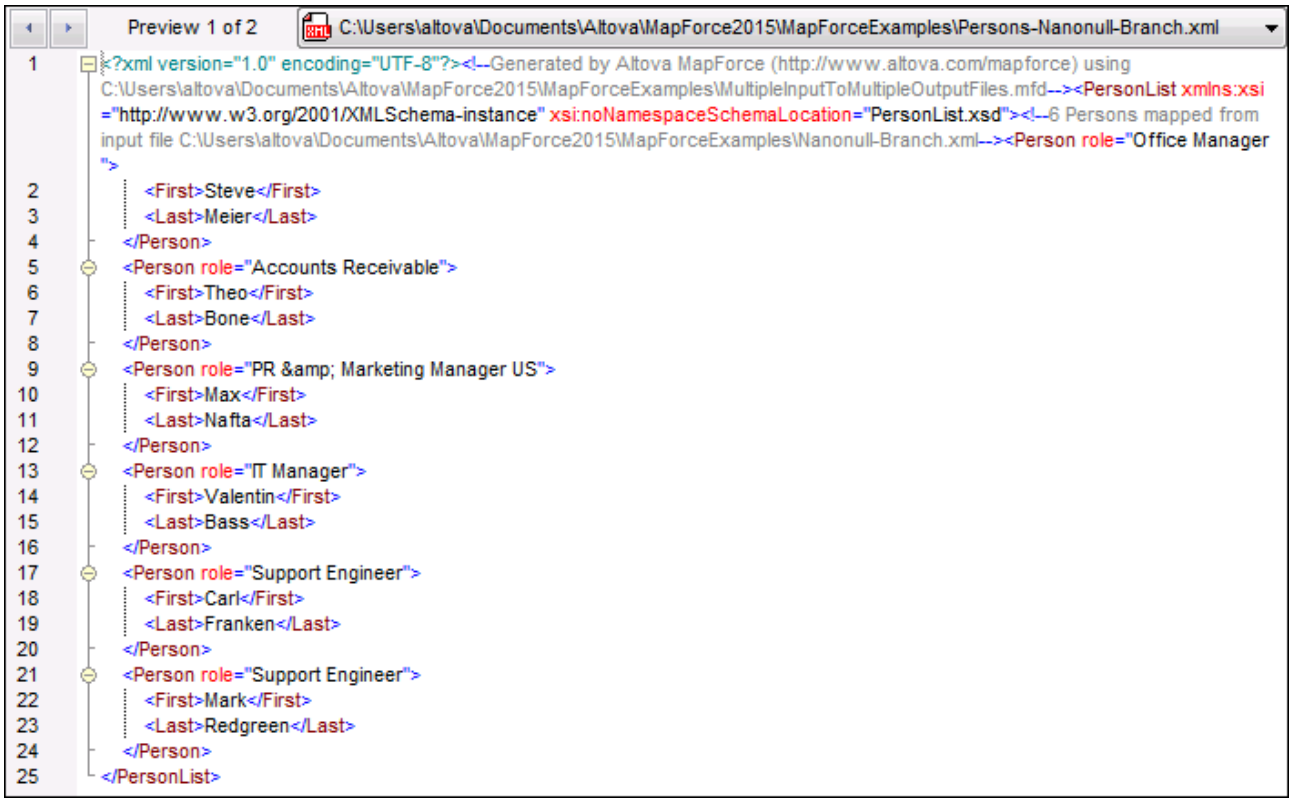
マッピングにカスタムファイル名を入力パラメーターとして提供するには、以下を行います:

1. 単純型入力コンポーネントをマッピングに追加する（「**関数**」メニューから「**入力の挿入**」をクリックします）。コンポーネントについての更に詳しい情報は、マッピングにパラメーターを提供するを参照してください。
2. ソースコンポーネントの「**ファイル**」（ File ）または「**ファイル/文字列**」（ File/String ）ボタンをクリックし、「**マッピングから与えられた動的なファイル名を使用する**」を選択します
3. マッピングソースとして振舞う、コンポーネントのルートノードに単純型入力パラメーターを接続します。

成功例に関しては例: 名前をマッピングパラメーターとして使用するを参照してください。

4.4.4　複数の出力ファイルをプレビューする

プレビューウィンドウにマッピングの結果を表示するために、「出力」タブをクリックします。マッピングが複数の出力ファイルを作成する場合、各ファイルは、「出力」タブ内でペインごとに番号が付けられます。矢印をクリックして、それぞれの出力ファイルを確認してください。



MultipleInputToMultipleOutputFiles.mfd

生成された出力ファイルを保存するためには以下を行います:

- 「出力」 メニューから「すべての出力ファイルを保存」 ( ) をクリックします。
- 「すべての生成された出力を保存」 ( ) ツールバーボタンをクリックします。

4.4.5 例: 1つの XML ファイルを複数のファイルに分割

このサンプルは、単一のソースXML ファイルから動的に複数のXML ファイルを作成する方法を説明します。このサンプルに使用されるマッピングは以下で検索することができます: **<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥Tut-ExpReport-dyn.mfd**。

ソースXML ファイル(マッピングと同じフォルダーで見つけることのできる) は、"Fred Landis" という個人の経費レポートから構成されており、5つの異なる種類の経費アイテムを含んでいます。このサンプルの目的は個々の経費アイテムのためにXML ファイルを生成することです。

Person

First	Fred
Last	Landis
Title	Project Manager
Phone	123-456-78
Email	f.landis@nanonull.com

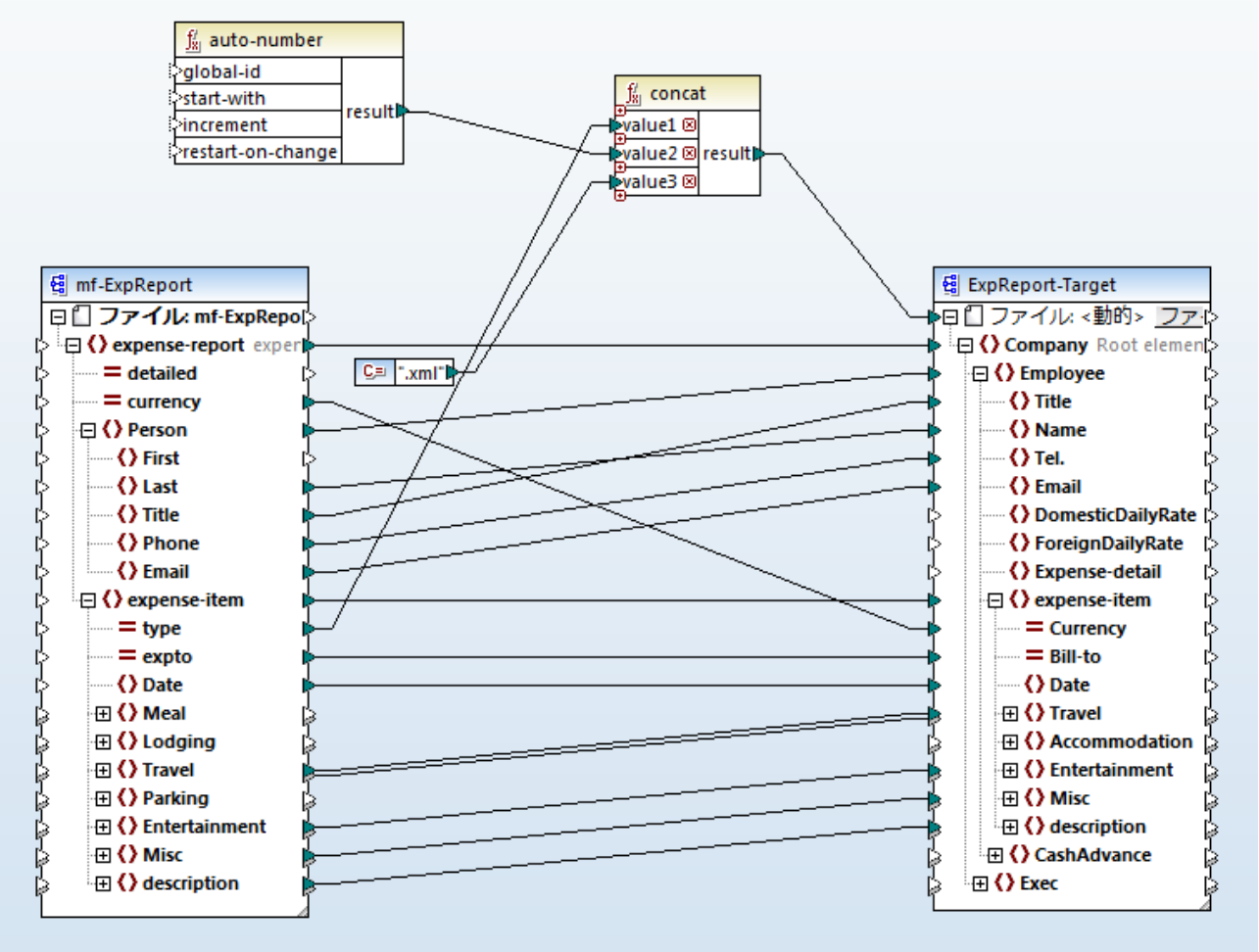
expense-item (5)

	type	expto	Date	Travel	Lodging
1	Travel	Development	2003-01-02	**Travel** Trav-cost=337.88	
2	Lodging	Sales	2003-01-01		**Lodging**
3	Travel	Accounting	2003-07-07	**Travel** Trav-cost=1014.22	
4	Travel	Marketing	2003-02-02	**Travel** Trav-cost=2000	
5	Meal	Sales	2003-03-03		

mf-ExpReport.xml (XMLSpy グリッドビューにて表示)

`type` 属性は特定の経費アイテムの型を定義し、これはソースファイルを分割するために使用されるアイテムの型です。このエクササイズのゴールを達成するには、以下を行います:

1. (「ライブラリ」ペインの「core | string 関数」ライブラリからドラッグして) `concat` 関数を挿入します。
2. 定数を入力するには、(「挿入」メニューから「定数」をクリックして) ".xml" を値として入力します。
3. (「ライブラリ」ペインの「core | generator 関数」からドラッグして) `auto-number` 関数を入力します。
4. 「ファイル」(File) またはターゲットコンポーネントの「ファイル/文字列」(File/String) ボタンをクリックし、「マッピングから与えられた動的なファイル名を使用する」を選択します
5. 下に表示されるとおり接続を作成し、「出力」 タブをクリックしマッピングの結果を確認します。



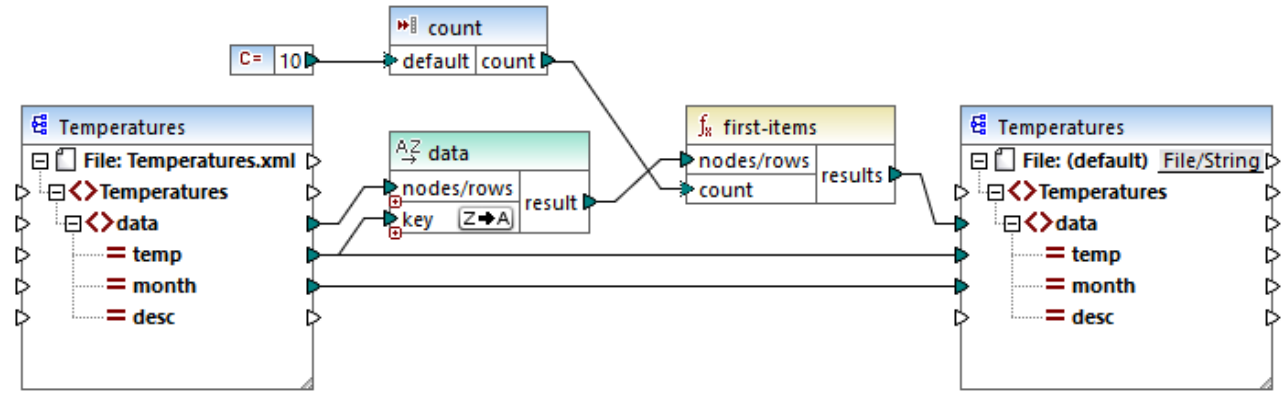
Tut-ExpReport-dyn.mfd (MapForce Basic Edition)

結果的出力ファイルは、以下のとおり動的に名前が付けられます:

- `type` 属性は、ファイル名の最初の部分を提供します。(例: "Travel")。
- **自動連番** 関数は、ファイルの連番を提供します。(例: "Travel1"、"Travel2"、など)。
- 定数は ".xml" であるファイル拡張子を提供します。このため "Travel1.xml" は最初のファイル名です。

4.5 マッピングにパラメーターを与える

パラメーターを入力としてとるマッピングを作成する場合、「単純な入力コンポーネント」と呼ばれる特別なコンポーネント型を追加して行うことができます。単純型入力コンポーネントは常に（例えば、文字列、整数などの）単純型データ型をアイテムとシーケンスの構造の代わりに持っています。例えば、下に表示されるマッピングでは単純型入力コンポーネント **count** が存在します。役割はパラメーターとして（値 **10** をデフォルトとして持つ）ソースXMLファイルから抽出される行の最大数を提供します。重要な点は、*N* がパラメーターの値である個所でマッピングが最高 *N* 気温のみを出力するように first-items 関数に入力として提供されるノードは並べ替えコンポーネントを使用して並べ替えられます。



FindHighestTemperatures.mfd

他の単純型入力コンポーネントの比較的一般的な使用方法はマッピングへのファイル名の提供です。これは入力ファイルを読み取る、または出力ファイルを動的に読み取るマッピング内で役に立ちます。複数の入力または出力ファイルを動的に処理するを参照してください。生成されたファイル内では、単純型入力コンポーネントは、スタイルシートのパラメーターに対応します。

単純型入力コンポーネント（またはパラメーター）を任意または必須で作成することができます 単純型入力コンポーネントを追加を参照してください。必要であれば、マッピング入力パラメーターのためにデフォルトの値を作成することができます デフォルトの入力値を作成するを参照してください。マッピングの実行時にパラメーターの値を明示的に提供していない場合でもこれによりマッピングを安全に実行することができます。例：ファイル名をマッピングパラメーターとして使用するを参照。

メインマッピングエリアに追加される入力パラメーターをユーザー定義関数内の入力パラメーターと区別してください。これら2つの類似点と相違点は、以下のとおりです。

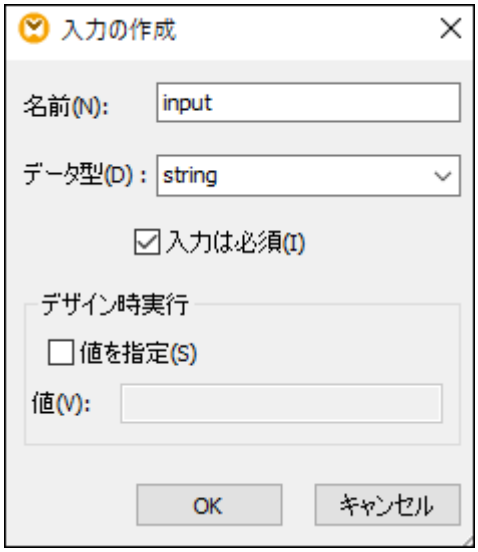
マッピング上の入力パラメーター	ユーザー定義関数の入力パラメーター
「**関数	入力の挿入**」メニューから追加する。	「**関数	入力の挿入**」メニューから追加する。
単純型データ型を持つことができます（文字列、整数など）。	単純型と複雑型のデータ型を持つことができます
マッピング全体に適用することができます。	定義された方法で関数のコンテキストに適用することができます。

逆のマッピングを作成する場合、（メニューコマンド「**ツール | 逆順マッピングの作成**」を使用すると）単純型入力コンポーネントは、単純型出力コンポーネントになります。

4.5.1 単純型入力コンポーネントの追加

マッピングに単純型入力コンポーネントを追加する:

1. マッピング ウィンドウに(ユーザー定義関数ではない)メインマッピングが表示されていることを確認してください。
2. 以下の1つを行ってください:
   - 「**関数**」メニューから「**入力**」をクリックします。
   - 「**挿入**」メニューから「**入力の挿入**」をクリックします。
   - 「**入力の挿入**」ツールバーボタンをクリックします。



3. 名前を入力して、この入力に必要なデータ型を選択します。入力が必須のマッピング パラメーターとして扱われる必要がある場合、「**入力は必須**」チェックボックスを選択します。設定の完全なリストは、単純型入力コンポーネント設定を参照してください。

メモ パラメーター名は文字、数値、およびアンダースコアのみを含むことができます。他の文字は許可されていません。これによりマッピングがコード生成言語すべてのために使用できます。

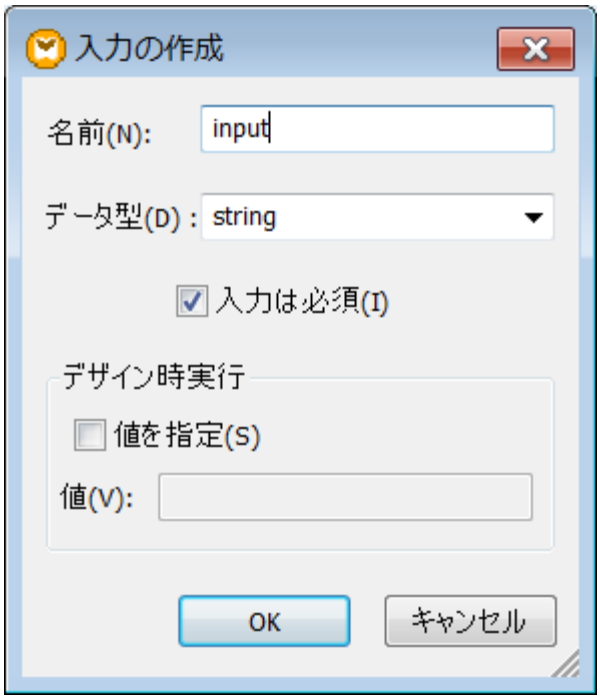
4. 「OK」をクリックします。

ここで定義されている設定は後に変更することができます(単純型入力コンポーネント設定を参照)。

1. マッピング ウィンドウに(ユーザー定義関数ではない)メインマッピングが表示されていることを確認してください。
2. 「**関数**」メニューから「**入力**」をクリックします。
3. 名前を入力して、この入力に必要なデータ型を選択します。入力が必須のマッピング パラメーターとして扱われる必要がある場合、「**入力は必須**」チェックボックスを選択します。設定の完全なリストは、単純型入力コンポーネント設定を参照してください。

メモ パラメーターの名前は、文字、数字、アンダースコアを含むことができますが、その他は許可されていません。これにより、マッピングは全てのコード生成言語と作業可能になります。

4. 「OK」をクリックします。



入力の作成 ダイアログボックス

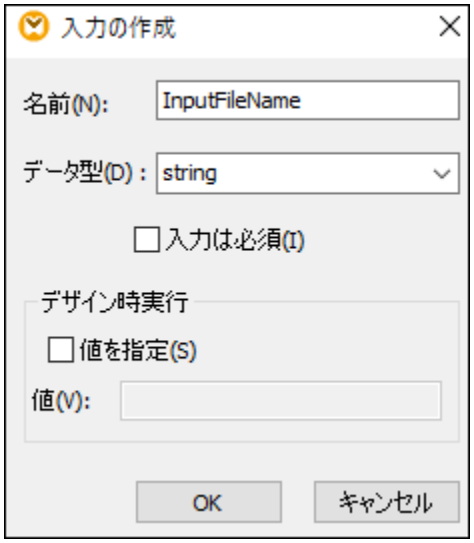
ここで定義されている設定は後に変更することができます(単純型入力コンポーネント設定を参照)。

4.5.2 単純型入力コンポーネント設定

単純型入力コンポーネントに適用することのできる設定をマッピングエリアに追加する際に定義することができます。後に設定を「入力の編集」ダイアログボックスから変更することも可能です。

「入力の編集」ダイアログボックスを開くには、以下を行います:

- コンポーネントを選択して、「コンポーネント」メニューから「プロパティ」をクリックします。
- コンポーネントをダブルクリックします。
- コンポーネントを右クリックして、「プロパティ」をクリックします。



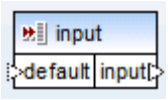
入力の編集 ダイアログボックス

.使用することのできる設定は、以下のとおりです:

名前	このコンポーネントに対応する入力パラメーターのための詳細名を入力します。マッピングの実行時、このテキストボックスに入力された値がマッピングに与えられるパラメーターの名前になります。ですから、スペースまたは特別な文字は許可されていません。
データ型	デフォルトでは、全ての入力パラメーターは文字列データ型として扱われます。パラメーターが異なる型を持つ場合、リストからそれぞれの値を選択します。マッピングが実行された場合、MapForce は、入力パラメーターをここで選択されたデータ型にキャストします。
入力は必須	有効化されている場合、この設定は、入力パラメーターを必須にします(すなわち、パラメーターの値が提供されない限り、マッピングを実行することはできません)。 入力パラメーターのためのデフォルトの値を指定するには、このチェックボックスをクリアします(デフォルトの入力値を作成するを参照)。
値の指定	この設定は、デザイン時にマッピングを実行する場合のみ適用することができます。「プレビュー」タブをクリックすることにより、コンポーネント内に直接マッピング入力として使用される値を入力することができます。
値	この設定は、デザイン時にマッピングを実行する場合のみ適用することができます。「プレビュー」タブをクリックすることにより、マッピング入力として MapForce で使用される値を入力するには、値の指定チェックボックスを選択し、必要な値を入力します。 **メモ** **値の指定** チェックボックスをクリックし横のボックスに値を入力した場合、入力された値は、マッピングをプレビューする際に、以前入力された(すなわち、デザイン時の実行時の)デフォルトの値を上書きします。しかしながら、デザイン時の値は MapForce Server による実行または FlowForce Server へのデプロイ時に生成された XSLT、XQuery、またはプログラムコードコードに影響を与えません。

4.5.3 デフォルトの入力値を作成する

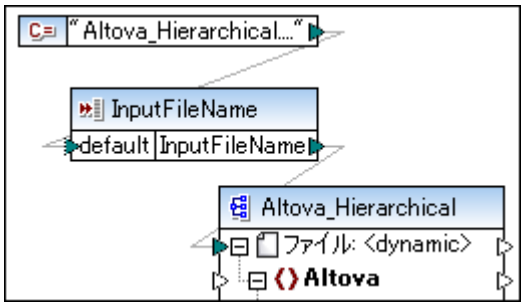
入力コンポーネントをマッピングエリアに追加すると、**default** アイテムがコンポーネントの左側に表示されます。



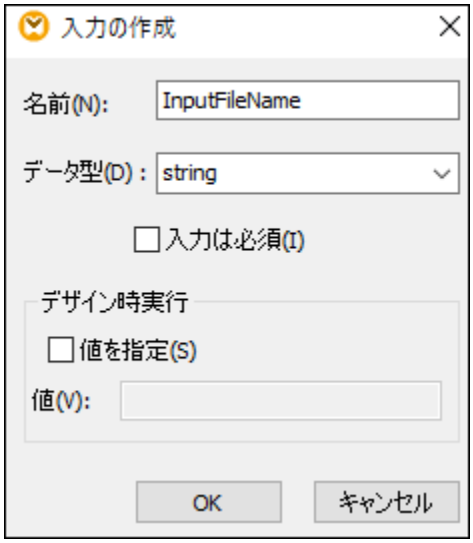
単純型入力コンポーネント

デフォルトアイテムは任意のデフォルトの値のこの入力コンポーネントへの接続を以下のように有効化します:

1. 定数コンポーネントを追加し、(「**挿入**」メニューから「**定数**」をクリックします。) 入力コンポーネントの**default** アイテムに接続します。



2. 入力コンポーネントをダブルクリックして、**入力は必須**チェックボックスのチェックをクリアします。デフォルトの入力の値を作成すると、この設定は重要になり、マッピングの検証に関する警告を引き起こす可能性があります。



3. 「OK」をクリックします。

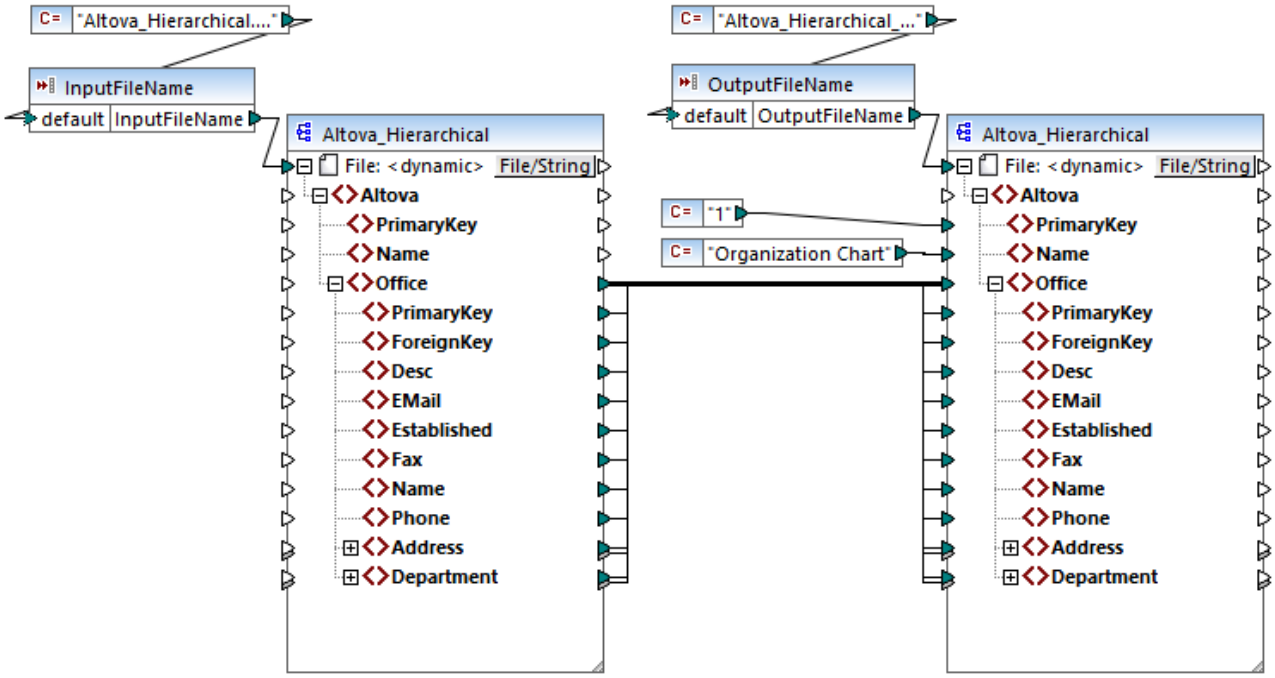
メモ　**値の指定** チェックボックスをクリックし横のボックスに値を入力した場合、入力された値は、マッピングをプレビューする際に、以前入力された(すなわち、デザイン時の実行時の)デフォルトの値を上書きします。しかしながら、デザイン時の値はMapForce Server による実行またはFlowForce Server へのデプロイ時に生成されたXSLT、XQuery、またはプログラムコードコードに影響を与えません。

4.5.4 例: ファイル名をマッピングパラメーターとして使用する

このサンプルはランタイムに入力パラメーターを取り込むマッピングの実行に必要なステップを説明します。マッピングデザインファイルは、以下のパスにあるサンプルを使用します: **<マイドキュメント >\Altova\MapForce2021\MapForceExamples\FileNamesAsParameters.mfd**.

このマッピングはXML ファイルからデータを抽出し、ターゲット XML ファイルに書き込みます。データはターゲットファイルにほぼ変更されることなく書き込まれます。属性 **PrimaryKey** と**Name** のみがマッピングからの定数の一部と共に作成されます。マッピングのメインの目的はマッピングのランタイムで呼び出し元がマッピングのパラメーターとして入力ファイルの名前と出力ファイルの名前を指定できるようになることです。

これを達成するには、マッピングには2 つの入力コンポーネントが存在します: **InputFileName** and **OutputFileName**。これらのコンポーネントはソースとターゲット XML ファイルの入力ファイル名（および出力ファイル名をそれぞれ）与えます。この理由のため、これらのコンポーネントは、**ファイル: <動的>** アイテムに接続されています。**ファイル**（ File ）ボタンをクリックして、**マッピングにより提供された動的なファイル名を使用する**を選択してコンポーネントをこのモードに切り替えることができます。



FileNamesAsParameters.mfd (MapForce Enterprise Edition)

InputFileName と**OutputFileName** コンポーネントのタイトルバーをダブルクリックすると、コンポーネントをビューすることができます。例えば、単純型入力コンポーネント設定で説明されているとおり、入力パラメーターのデータ型を指定、または、入力パラメーターの名前を変更することができます。このサンプルでは、入力と出力パラメーターは以下のように構成されています:

- **InputFileName** パラメーターは型 [string] で、同じマッピング内で定義されている定数により与えられているデフォルトの値を持っています。定数は型 [string] で、その値は [Altova_ Hierarchical.xml] です。このために、他の値がパラメーターとして与えられないと想定し、マッピングが実行されると [Altova_ Hierarchical.xml] と呼ばれるファイルからデータを読み取ろうと試みます。
- **OutputFileName** パラメーターは型 [string] で、同じマッピング内で定義されている定数により与えられているデフォルトの値を持っています。定数は型 [string] で値は [Altova_ Hierarchical_ output.xml] ですこのため、このために、他の値がパラメーターとして与えられないと想定し、マッピングは実行時に [Altova_ Hierarchical_ output.xml] と呼ばれるXML 出力ファイルを作成します。

以下のセクションはマッピングの実行方法と以下の変換言語でパラメーターを与える方法を説明しています:

- RaptorXML Server を使用したXSLT 2.0

XSLT 2.0

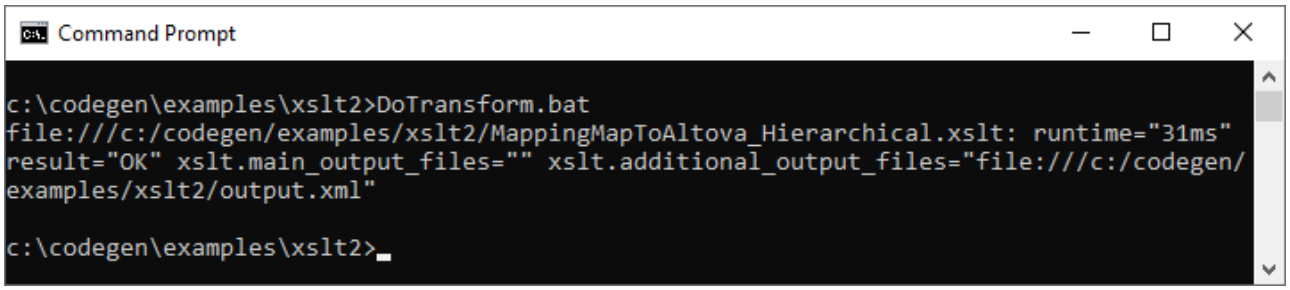
XSLT 1.0、XSLT 2.0、またはXSLT 3.0 でコードを生成する場合、XSLTファイルに加え、**DoTransform.bat** バッチファイルが選択されたターゲットディレクトリで生成されます。**DoTransform.bat** によりRaptorXML Server を使用してマッピングを実行できるようになります。 RaptorXML Server を使用した自動化を参照してください。

異なる入力（または、出力）ファイルを使用するために**DoTransform.bat** ファイルを必要とするパラメーターが含まれるように以下のように編集します:

1. 最初にXSLT コードを生成します。例えば、XSLT 2.0, を生成する場合「**ファイル | コード生成 | XSLT 2.0**」を選択します。
2. <**マイドキュメント**>¥Altova¥MapForce2021¥MapForceExamples¥から**Altova_Hierarchical.xml** ファイルをXSLT 2.0 コードが直接生成されるディレクトリにコピーします（このサンプルでは、**c:\codegen\examples\xslt2**）。**InputFileName** パラメーターにカスタム値を提供しない場合、前記の通り、マッピングはこのファイルを読み取ろうと試みます。
3. **DoTransform.bat** が`%*` の前または後にカスタム入力パラメーターを含むように編集します。パラメーターの値は一重引用符で閉じられていることに注意してください。使用することのできる入力パラメーターは**rem**（リマーク）セクションにリストされています。**output.xml** と呼ばれる出力ファイルを生成すると想定します。これを達成するには**DoTransform.bat** ファイルを以下のように変更します:

```
@echo off

RaptorXML xslt --xslt-version=2
  --input="MappingMapToAltova_Hierarchical.xslt"
  --param=OutputFileName:'output.xml' %* "MappingMapToAltova_Hierarchical.xslt"
rem --param=InputFileName:
rem --param=OutputFileName:
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

DoTransform.batファイルを実行すると、RaptorXML Server は**Altova_Hierarchical.xml** を入力パラメーターとして使用して、変換を完了します。上のステップに従うと生成されたファイルの名前は**output.xml** になります。



4.6 マッピングから文字列の値を返す

マッピングから文字列を返すことが必要な場合単純型出力コンポーネントを使用します。マッピングエリアでは、単純型出力コンポーネントは、アイテムとシーケンスの構造の代わりに文字列データ型を持つターゲットコンポーネントの役割をします。この結果、単純型出力コンポーネント（または追加して）ファイルベースのターゲットコンポーネントの代わりに作成することができます。. 例: 単純型出力コンポーネントを使用して、素早く関数の出力をテストとプレビューすることができます関数の出力（例: 関数の出力をテストするを参照）。

単純型出力 コンポーネントをユーザー定義関数の出力パラメーターと混同しないように注意してください。（ユーザー定義関数を参照）。類似点と相違点は、以下のとおりです。

出力コンポーネント	ユーザー定義関数出力パラメーター
「関数	出力の挿入」メニューから追加	「関数	出力の挿入」メニューから追加
文字列をデータ型として持ちます。	単純型と複雑型のデータ型を持つことができます
マッピング全体に適用することができます。	定義された方法で関数のコンテキストに適用することができます。

必要であれば、複数の単純型出力 コンポーネントをマッピングに追加することができます。単純型出力 コンポーネントをファイルベースのターゲットコンポーネント と共に使用することができます。マッピングが複数のターゲットコンポーネントを含む場合、特定のコンポーネントにより返されるデータをプレビューすることができます。コンポーネントタイトルバー内の「プレビュー」（ ）ボタンをクリックすることにより、マッピングウィンドウの「出力」タブをクリックします。

単純型出力 コンポーネントを MapForce 変換言語内で以下のように使用することができます:

言語	作動のしくみ
XSLT 1.0、XSLT 2.0、XSLT 3.0	生成されたXSLT ファイル内で、マッピング内で定義されている単純型出力コンポーネントは XSLT 変換の出力になります。 RaptorXML Server を使用する場合、RaptorXML Server にマッピング出力をファイルに書き込み、 `--output` パラメータに値としてパスするように命令することができます。 ファイルに出力を書き込む、**DoTransform.bat** ファイル内の `--output` パラメーターに追加または編集する場合。例: 次のDoTransform.bat ファイルは**Output.txt** ファイルへのマッピング出力に書き込むために編集されました（ハイライトされたテキスト参照してください）。 `RaptorXML xslt --xslt-version=2 --input="MappingMapToResult1.xslt" --output="Output.txt" %* "MappingMapToResult1.xslt"` `--output` パラメーターが定義されていない場合、マッピングが実行される際、マッピング出力が標準出力ストリーム(stdout) に書き込まれます。

逆のマッピングを作成する場合、（メニューコマンド「ツール | 逆マッピングの作成」を使用すると）単純型出力コンポーネントは、単純型入力コンポーネントになります。

4.6.1 単純型出力コンポーネントの追加

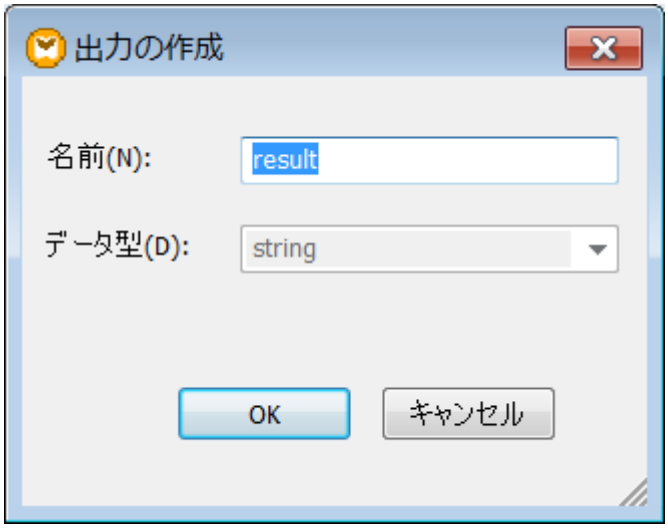
マッピングエリアに出力コンポーネントを追加する

1. ウィンドウが(ユーザー定義関数ではない)インのマッピングを表示していることを確認してください。
2. 以下の一つを行います:

   a. 「関数」メニューから、「出力の挿入」をクリックします。

   b. 「出力の挿入」  ツールバーボタンをクリックします。

3. コンポーネントのための名前を入力します。
4. 「OK」をクリックします。



出力の作成ダイアログボックス

コンポーネント名を以下の方法で後に変更することができます:

- コンポーネントを選択して、「コンポーネント」メニューから「プロパティ」をクリックします。
- コンポーネントヘッダーをダブルクリックします。
- コンポーネントヘッダーを右クリックして、「プロパティ」をクリックします。

4.6.2 例: 関数出力のプレビュー

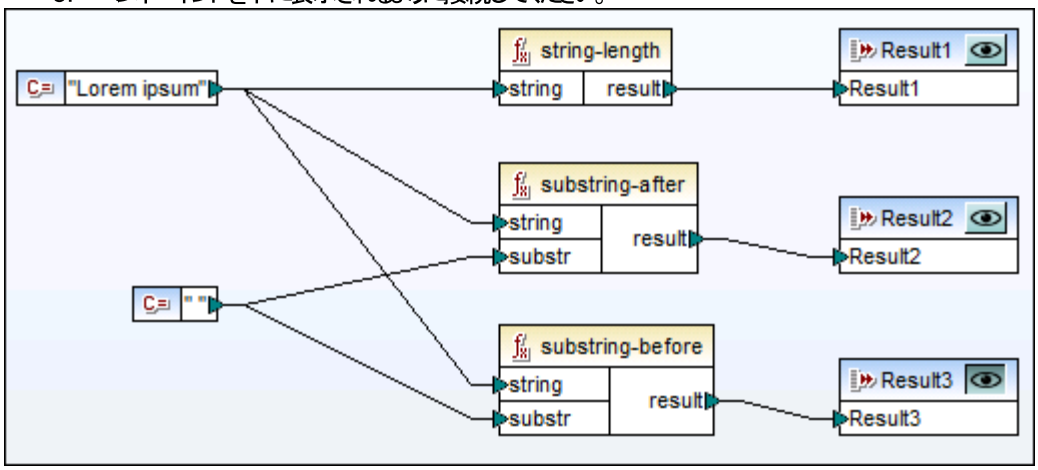
この例は、単純型出力コンポーネントを使用してMapForce 関数により返された出力を確認する方法を説明します。関数に関する基本的理解と、MapForce 関数に関する全般的な知識があると、このサンプルをよく理解することができます。MapForce 関数の基礎知識がない場合、例を開始する前に関数の使用を参照してください。

この例の目的は、マッピングエリアに複数の関数を追加することで、また単純型出力コンポーネントを使用して出力をプレビューする方法を理解することです。特に、例はcore ライブラリ内で使用することのできる複数の関数をマッピングエリアに追加する方法を説明しています。次に使用方法の概要が述べられています:

string-length	は、引数として与えられた文字列内の文字の数を返します。例: これを関数値 "Lorem ipsum" にパスすると、テキスト "Lorem ipsum" が持つ文字の数である結果は "11" です。
substring-after	は、引数として与えられたセパレーターの後に発生する文字列の一部を返します。例: これを関数値 "Lorem ipsum" とスペース文字 (" ") にパスすると、結果は "ipsum" です。
substring-before	は、引数として与えられたセパレーターの前に発生する文字列の一部を返します。例: これを関数値 "Lorem ipsum" とスペース文字 (" ") にパスすると、結果は "Lorem" です。

カスタムテキストの値に対して関数をテストするには (このサンプルでは "Lorem ipsum") 以下のステップを行います:

1. 値を持つ定数 "Lorem ipsum" マッピングエリアを追加します (メニューコマンド「**挿入 | 定数**」を使用して)。定数はテストされる関数のそれぞれの入力パラメーターになります。
2. **string-length**、**substring-after** と**substring-before** 関数を core ライブラリ、string 関数 セクションからマッピングエリアにドラッグして、マッピングエリアに追加します。
3. 空のスペース (" ") を値として持つ定数を追加します。これは**substring-after** と**substring-before** 関数により必要とされるセパレーターパラメーターとなります。
4. 単純型出力 コンポーネントを (メニューコマンド「**関数 | 出力の挿入**」を使用して) 追加します。他の名前を与えることも可能ですが、このサンプルでは、*Result1*、*Result2*、および *Result3*, と名前が付けられています。
5. コンポーネントを下に表示されるように接続してください。



単純型出力 コンポーネントを使用して関数の出力をテストする

上のサンプルで示されるとおり、"Lorem ipsum" 文字列はそれぞれの、**string-length**、**substring-after**、および **substring-before** 関数入力パラメーターとして振る舞います。更に、**substring-after** と**substring-before** 関数は、スペースの値を第2入力パラメーターとして取ります。*Result1*、*Result2*、および *Result3* コンポーネントは各関数の結果をプレビューするために使用することができます。

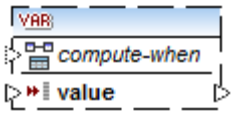
関数の出力をプレビューする方法:

- コンポーネントタイトルバー内の「プレビュー」( ) ボタンをクリックし、マッピングウィンドウの「出力」 タブをクリックします。

4.7 変数の使用

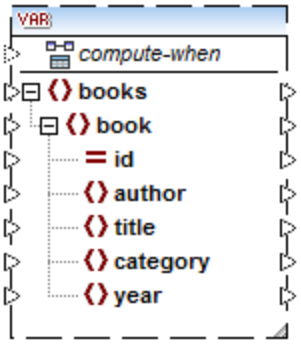
変数とは更なる処理のために中間マッピングの結果を保管するために使用される特別なコンポーネントです。マッピング上のデータを一時的に「記憶」し、処理する必要がある場合があります。例えば、ターゲットコンポーネントにコピーされる前に、フィルター、または関数を適用するなど。

変数は、単純型（例えば、文字列、整数、ブール値、など）または複合型（ツリー構造）であることができます。



単純型変数

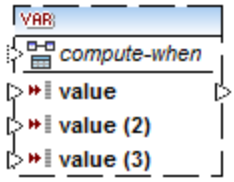
変数の構造を示すXMLスキーマを提供して複合型の変数を作成することができます。スキーマが要素をグローバルに定義すると、変数構造のルートノードになるものを選択することができます。変数に関連したインスタンスXMLファイルが存在しない場合、変数のデータはマッピングランタイムに計算されます。



XMLスキーマから作成された複合型変数

上のイメージでは、各変数には、`compute-when`という名前のアイテムが存在することに注意してください。このアイテムに接続することは任意です。これにより、マッピング上でどのように変数の値が計算されるかを管理することができます（次を参照してください：変数のコンテキストとスコープの変更）。

必要な場合、通常のコンポーネントと同様に、変数構造のアイテムを1つ以上のソース接続から受け入れられるように複製することができます（次を参照してください：入力の複製）。これは、しかしながらデータベーステーブルから作成された変数には適用されません。



複製された入力を持つ単純型変数

変数に関して最も重要な点は、変数はシーケンスであり、シーケンスを作成するために使用されるという点です。「シーケンス」という用語は、ここでは、ゼロまたは、それ以上のアイテムのリストを意味します（次も参照してください：マッピングのルールと戦略）。これにより、変数はマッピングの寿命内で複数のアイテムを処理することができるようになります。しかしながら、値を変数に与え、マッピングのその他の部分を同じに保つことも可能です（次を参照してください：変数のコンテキストとスコープの変更）。

ある程度までは、変数をチェーンマッピングの中間コンポーネントと比較することができます（チェーンマッピングを参照してください）。しかしながら、マッピング内の各段階で中間ファイルを作成する必要がない場合、変数は柔軟性があり便利です。次のテーブルの概要は変数とチェーンマッピングの違いを示しています。

チェーンマッピング	変数
チェーンマッピングには、2つの独立したステップが含まれます。例えば、A、B、とC の3つのコンポーネントがマッピングに存在するとします。マッピングの実行には2つの段階があります： A からB へのマッピングの実行、および、B からC へのマッピングの実行。	マッピングの実行中、変数は、コンテキストとスコープに従い評価されます。コンテキストとスコープを影響することはできます（変数のコンテキストとスコープの変更を参照してください）。
マッピングが実行されると、中間結果は外部でファイルに保管されます。	マッピングが実行されると、中間の結果は内部で保管されます。変数の結果を含む外部ファイルは作成されません。
中間 結果は ボタンを使用してプレビューすることができます 。	マッピング ランタイムで計算される変数の結果をプレビューすることはできません

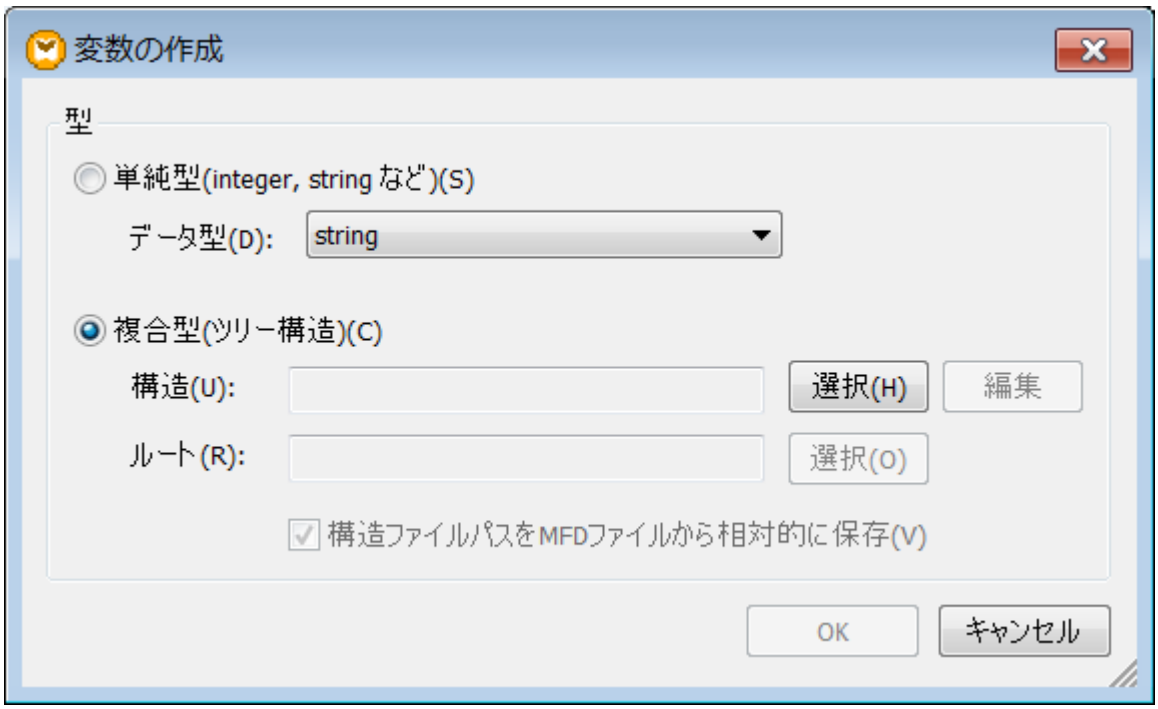
メモ 変数は、マッピング変換言語がXSLT 1.0 に設定されていない場合サポートされません。

4.7.1 変数の追加

マッピングに変数を追加するには、以下に示されているようにいくつかの方法があります。

メニュー、またはツールバーコマンドの使用

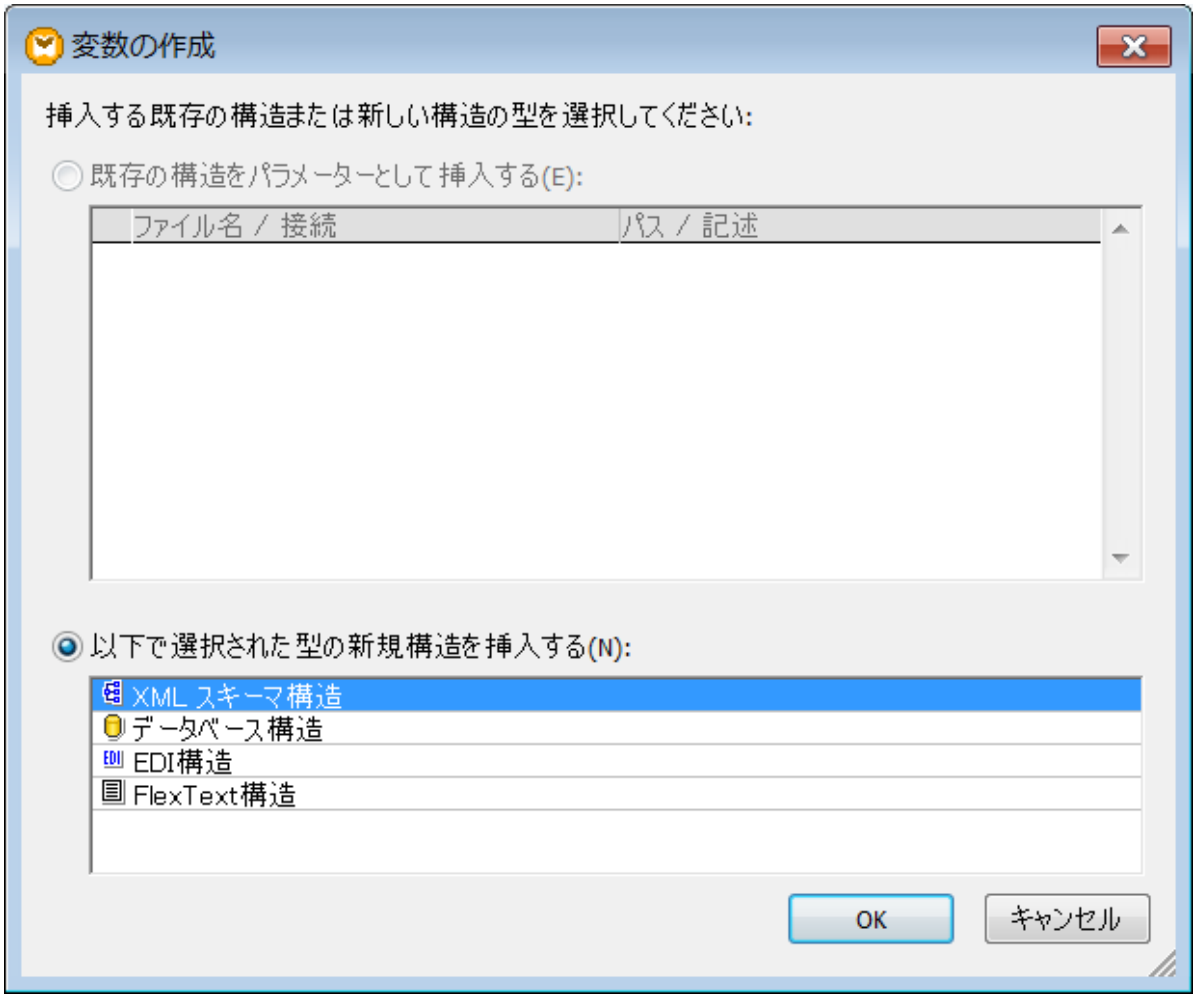
1. 「挿入」メニューから、「変数」をクリックします。（「変数」 ツールバーボタンを代わりにクリックします）。



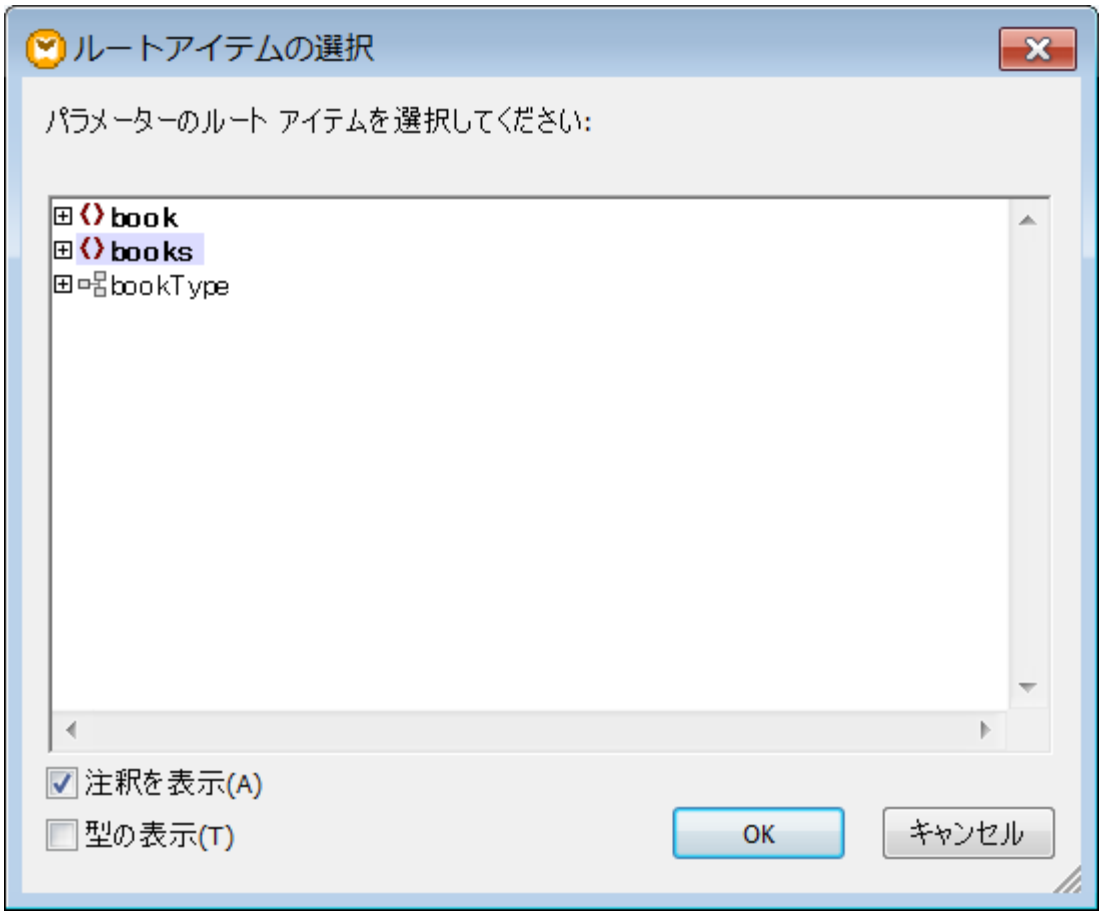
2. 挿入する変数の型を選択します（単純型 または 複合型）。

"複合型" を選択した場合、追加ステップが存在します:

3. 変数の構造を与えるソースを選択するために、「選択」をクリックします（例えば、XML スキーマ、）。

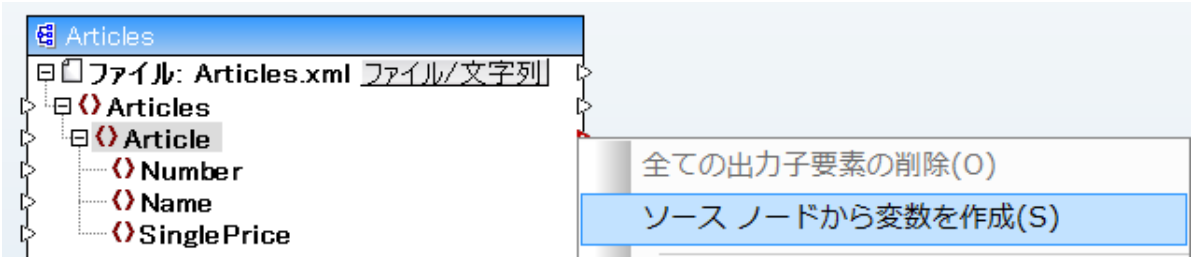


4. プロンプトされると、構造のルートアイテムを指定します。XML スキーマの場合、ルート アイテムは、グローバルに定義されているすべての要素であることができます。データベースの場合は、ルートアイテムはすべてのテーブルであることができます。

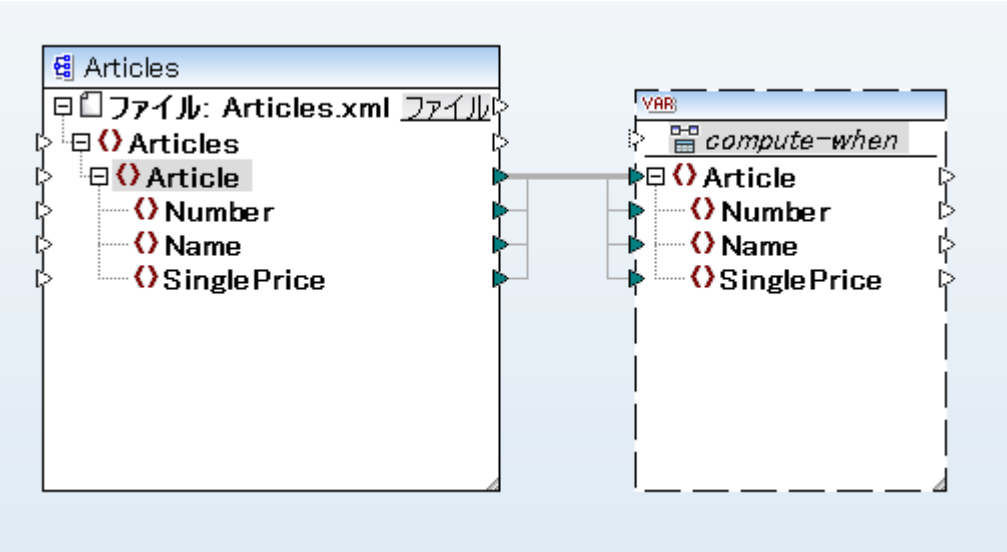


コンテキストメニューの使用

- コンポーネントの出力コネクタを右クリックします（このサンプルでは、「Article」）そして、「**ソースノードから変数を作成する**」を選択します。



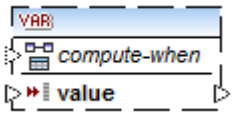
これにより、同じソーススキーマを使用する複合型の変数を作成し、自動的にすべてのアイテムをすべてをコピーする接続により接続します。



- ターゲットコンポーネントの入力コネクタを右クリックして、「ターゲットノードのために変数を作成する」を選択します。これにより、同じスキーマをターゲットとして使用し、複合型の変数を作成します。そして、すべてコピー接続を使用してすべてのアイテムを自動的に接続します。
- フィルターコンポーネント（on-true/on-false）の出力コネクタを右クリックして、「ソースノードから変数を作成する」を選択します。これにより、ソーススキーマを使用して複合型コンポーネントを作成し、フィルター入力にリンクされているアイテムを中間コンポーネントのルート要素として自動的に使用します。

4.7.2 変数のコンテキストとスコープの変更

各変数には、変数のスコープを管理することを許可する `compute-when` 入力アイテムが存在します。すなわち、マッピングが実行される際に変数の値がいつ、どの頻度で計算されるかを管理することができます。この入力は、多くの場合接続する必要がありませんが、デフォルトのコンテキストを上書きする場合、またはマッピングのパフォーマンスを最適化するには、必要です。



「Compute-when」アイテム

次のテキストでの、サブツリーは、ターゲットコンポーネント内のアイテム/ノードセットとその子孫を意味します。例えば、 `<FirstName>` と `<LastName>` 子要素を持つ、`<Person>` 要素。

変数の値 は、変数コンポーネントの出力サイドでデータが使用できることを意味します。

- 単純型の変数に関しては、コンポーネントプロパティ内で指定されているデータ型を持つ動的な値のシーケンスであることを意味します。
- 複合型の変数に関しては、それぞれが自身の子孫ノードを含む（コンポーネント プロパティ内で指定されている型の）ルートノードのシーケンスであることを意味します。

動的な値（または、ノード）のシーケンスは、1つの要素、または要素を全く含まない場合があります。これは、変数の入力サイドに何が接続されているか、および、ソースとターゲットコンポーネント内の親アイテムの存在により異なります。

「Compute-when」が接続されていない場合（デフォルト）

compute-when 入力アイテムがソースコンポーネントの出力ノードに接続されていない場合、ターゲットサブツリー内で最初に使用される際に変数の値はコネクタにより変数コンポーネントからターゲットコンポーネント内のノードに直接、または関数を使用して間接的に計算されます。同じ変数の値は、サブツリー内のターゲット子ノードすべてのために使用されます。

実際の変数の値は、ソースとターゲットコンポーネントの親アイテム間の接続により異なります。

このデフォルトの振る舞いは、正規ユーザー定義 関数とW eb サービス関数の呼び出しの複合型の出力と同じです。

変数の出力が複数の関連したターゲットノードに接続されている場合、変数の値は、それぞれのアイテムのために個別に計算されます。これにより、各ケース内で異なる結果を生成することができます。これは異なる親の接続は、変数の値が評価されるコンテキストを影響するからです。

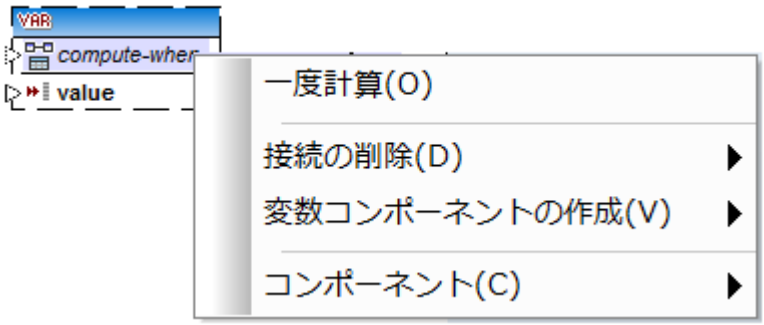
「Compute-when」が接続されている場合

ソースコンポーネントの出力コネクタを compute-when に接続すると、ソースアイテムが最初にターゲットサブツリー内で使用される都度に変数が計算されます。

変数は実際には、compute-when に接続されているアイテムの子アイテムのように振る舞います。これは、新規のアイテムがソースコンポーネント内のシーケンスから読み込まれる都度、ランタイム変数が再評価されます。これは、MapForce 内の接続を管理する一般的なルールに関連しています: 各ソースアイテムのために、1つのターゲットアイテムが作成されます。compute-when に関しては、各ソースアイテムのために、変数の値が計算されます（次を参照してください： マッピングのルールと戦略）。

「Compute-once」

必要であれば、ターゲットコンポーネントの前に1度変数の値を計算し、変数をマッピングの残りでグローバルな定数にすることを選択することができます。これをおこなうには、compute-when アイテムを右クリックして、コンテキストメニューから「1度計算する」を選択します:



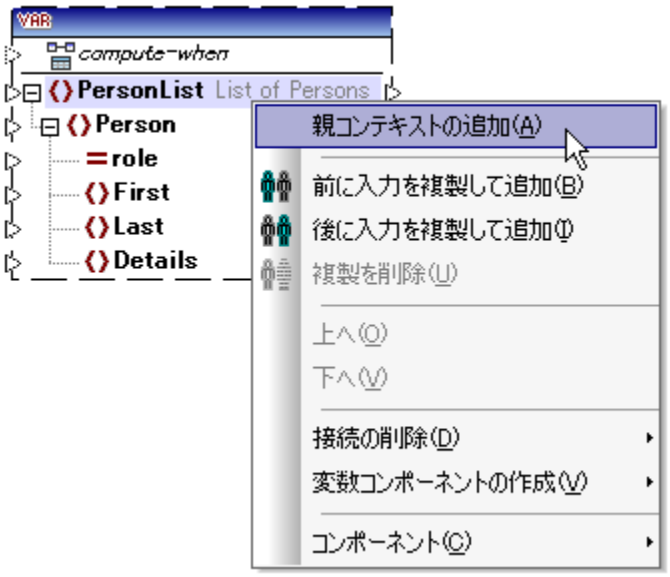
変数のスコープを compute-when=once に変更する場合、このような変数は1度のみ評価されるため、入力コネクタは、compute-when アイテムから削除されます。

実際の関数の結果が評価される前に、ユーザー定義関数 compute-when=once 変数は関数が呼び出される都度評価されます。

親コンテキスト

親コンテキストを追加する必要がある場合があります。例えば、マッピングが複数のフィルターを使用し、反復するために親ノードを追加する必要がある場合など。例: 親コンテキストの変更 を参照してください。

変数に親コンテキストを追加するには、ルートノード（このサンプルでは「PersonList」）を右クリックします、そして、コンテキストメニューから、**「親コンテキストを追加する」**を選択します。これにより新規ノード、親コンテキストを既存の階層構造に追加します。

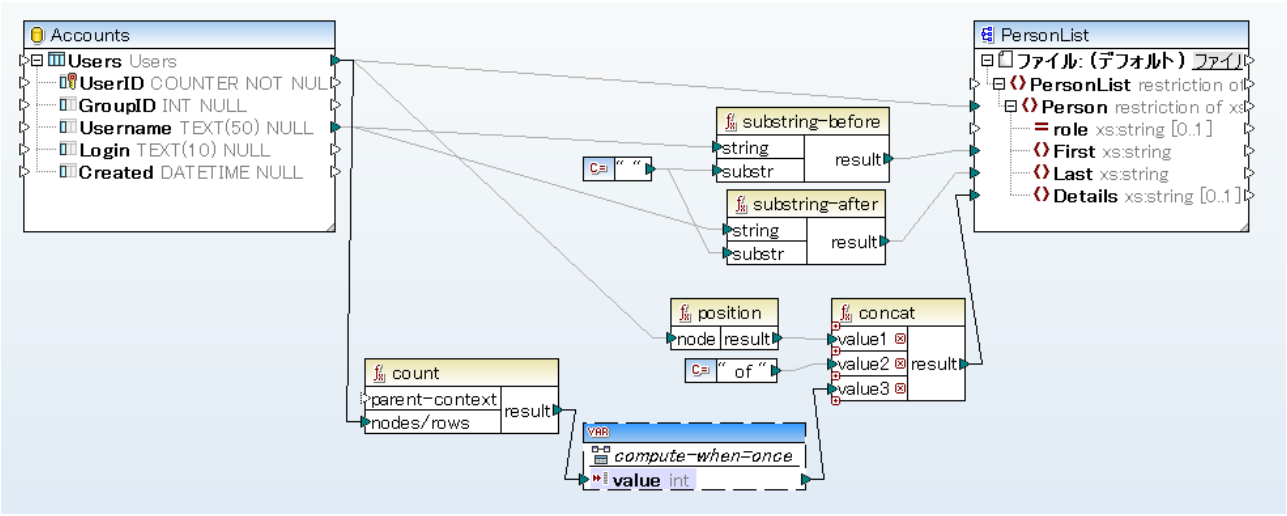


親コンテキストは、仮想の「親」ノードをコンポーネント内の階層構造に追加します。これにより、同じ、または異なるソースコンポーネント内で追加ノードを反復することができます。

4.7.3 例: データベースのテーブルの行の計算

このサンプルで説明されているマッピングは、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥フォルダー内の **DB_UserList.mfd** です。このマッピングは、ユーザーのレコードを "Users" という名前のデータベーステーブルから抽出し、XML ファイルに書き込みます。データベースの列 "Username" には、個人の姓と名の両方が含まれています（例えば、"Vernon Callaby"）。このマッピングの目的は以下のとおりです:

1. "Users" テーブル内の各レコードのために、XML ファイル内に新規の `Person` 要素を作成する。
2. データベースフィールド "Username" から抽出された値を XML ファイル内の個別のフィールド（"First" と "Last"）に分割する。
3. 各レコードのために、データベース内に存在するレコードの全体の番号に対して比較される連番を検索し（例えば、"4 件中の 1 件目" など） `Details` 要素にこの情報を書き込みます。



DB_UserList.mfd

上で説明されているとおり、最初の目的を達成するために、ソース"Users"テーブルとターゲット XML ファイルの Person 要素 の間に接続が描かれます。これにより、ソーステーブル内の各レコードのために、ターゲット内に新規の Person 要素が作成されます。

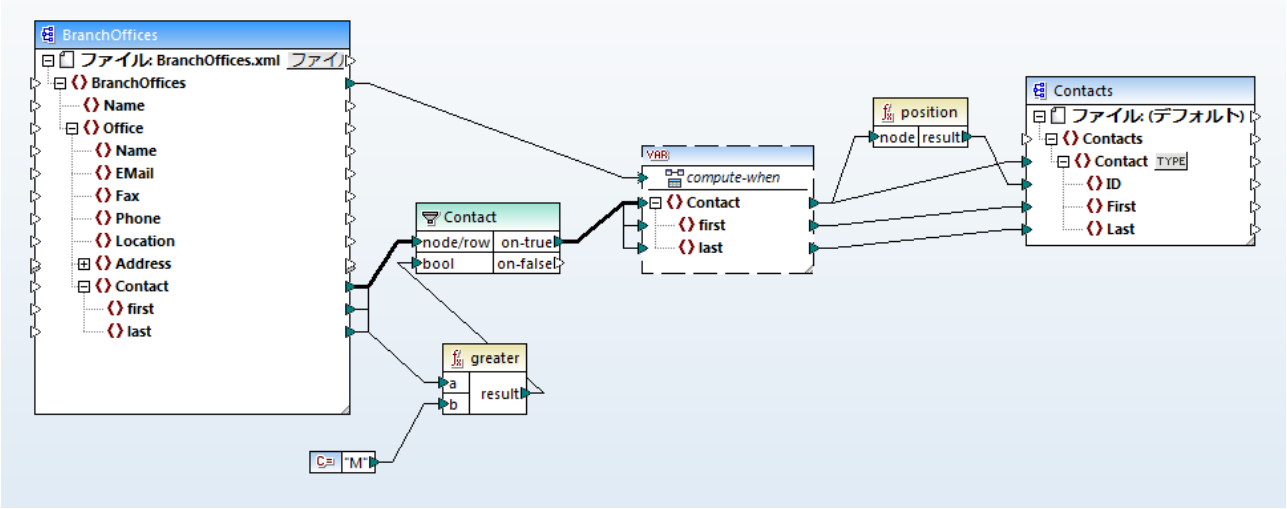
フィールド "Username" の値が substring-before と substring-after 関数に与えられています。これらの2 つの関数は、スペース文字 (" ") の前後のテキストをそれぞれ抽出し、このマッピングの目的を達成します。

3つ目の目的を達成するために、マッピングは count 関数を使用します。集計関数の結果は、変数にパスされます。変数は、マッピング上に結果が保管され、各個人の"Details" 要素 をターゲット XML に書き込む際に使用することができます。効率性のために、データベースのレコードは1 度のみ数えられるべきであり、変数のスコープは compute-when=once に設定されています (次を参照してください: 変数のコンテキストとスコープの変更)。

4.7.4 例: ノードのフィルターと番号付け

このサンプル内で説明されているマッピングのシーケンスは、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥ フォルダー内の **PositionInFiltered.mfd** で使用することができます。

このマッピングは、複数の個人の連絡先のデータを含むXML ファイルを読み取り、フィルターし、ターゲット XML ファイルに書き込みます。このマッピングの目的は、ソースXML ファイルから、「M」または、以降のアルファベットから始まる姓を持つ個人をフィルターします。抽出された連絡先には、番号が付けられている必要があります。番号は、ターゲット XML ファイル内で各連絡先の一意の識別子としての役割を果たします。



PositionInFilteredSequence.mfd

上の目的を達成するには、次のコンポーネント型 がマッピングに追加されました

- フィルター (次を参照してください: フィルターと条件)
- 複合型変数 (次を参照してください: 変数の追加)
- 関数 greater と position (次を参照してください: 関数をマッピングに追加する)
- 定数 (定数を追加するには、メニューコマンド「**挿入 | 定数**」を選択します)。

変数は、ソースコンポーネントと同じスキーマを使用します。変数を右クリックして、コンテキストメニューからプロパティを選択すると、この変数構造のためにルートノードとして ノード **BranchOffices/Office/Contact** が選択されていることを確認してください。

最初に、ソースコンポーネントのデータは、フィルターにパスされます。フィルターは、変数にフィルターの条件を満たすレコードのみをパスします。具体的には、フィルターは、最初の名前が「M」に等しい、または大きい値を持つ Contact ノードのみを取得するように構成されています。この目的を達成するには、関数 greater は、各 last アイテムを 定数値「M」と比較します。

変数には、ソースコンポーネント（BranchOffices）のルートアイテムに接続されている compute-when 入力があります。ランタイムでは、これは、ソースコンポーネント内のシーケンスから新規のアイテムが読み込まれる都度、変数を再評価するようにします。このマッピングでは、しかしながら、compute-when アイテムの接続、または未接続に違いはありません。この理由は、変数がフィルターを使用して間接的に Contact ソースアイテムに接続されており、フィルター条件を満たす Contact のインスタンスの数だけ計算がおこなわれるからです。

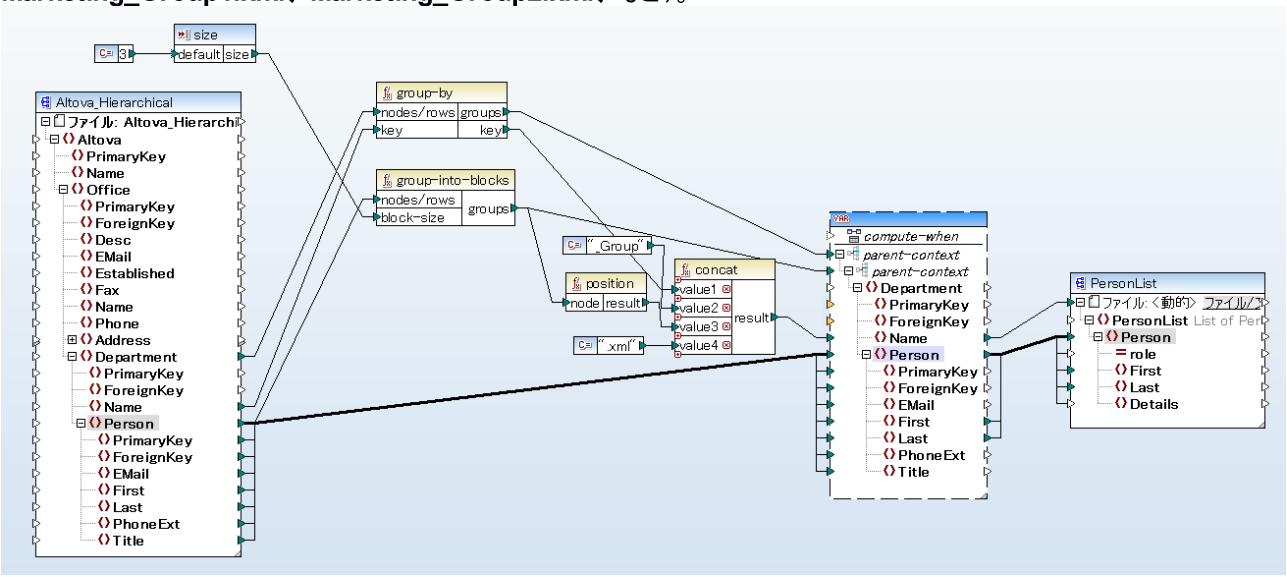
position 関数は、変数の各反復のために現在のシーケンスの番号を返します。8 件の連絡先がフィルターの条件を満たしています。ですから、マッピングをプレビューして、出力を確認すると、1 から 8 の ID が、ターゲットコンポーネントの ID 要素に書き込まれていることが確認することができます。

変数の必要性に関しては、すべてのレコードに番号付けをする必要性に応じるものです。フィルターの結果を直接ターゲットコンポーネントに接続すると、Contact の各発生に番号を付けることができません。このマッピング内の変数の目的は、ですから、Contact の各インスタンスを一時的にマッピングに保管し、ターゲットに書き込まれる前に番号を付けることです。

4.7.5 例:記録のグループ化、および、サブグループ化

このサンプルで説明されているマッピングは、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥フォルダー内の **DividePersonsByDepartmentIntoGroups.mfd** です。

このマッピングは、架空の会社の社員のレコードを含むXML ファイルを処理します。社内には2つのオフィスが存在します:「Nanonull, Inc.」と「Nanonull Partners, Inc」。各オフィスには複数の部署が存在します（例えば、「IT」、「マーケティング」など）、また、各部署には一名以上の社員が存在します。マッピングの目的は、オフィスに関わりなく各部署から3人までで構成されているグループを作成することです。グループのサイズはデフォルトでは3名です。しかしながら、必要に応じて変更することもできます。各グループは、フォーマット"<Department Name>_GroupN" の名前を持つ個別のXML ファイルとして保存される必要があります（例えば、**Marketing_Group1.xml**、**Marketing_Group2.xml**、など）。



DividePersonsByDepartmentIntoGroups.mfd

上で説明されているとおり、マッピングの目的を達成するには、複合型変数と主に関数である他のコンポーネント型をマッピングに追加します。ソースXML 内の Department アイテムと同じ構造が変数には存在します。プロパティを表示するために変数を右クリックすると、ソースコンポーネントと同じ XML スキーマが使用されていること、および、ルート要素として Department が存在していることに気が付きます。重要な点は、各部署のコンテキスト内で変数が最初に計算され、次に各グループのコンテキスト内で計算されるように、変数には、2つのネストされた親コンテキストアイテムが存在することです（次も参照してください：変数のコンテキストとスコープの変更）。

始めに、マッピングは、各部署の名前を取得するためにすべての部署を反復します（これは各グループに対応するファイル名を後に作成するために必要とされます）。これは、**group-by** 関数を Department ソースアイテムに接続すること、部署名をグループ化のキーとして与えることにより達成されます。

次に、各部署のコンテキスト内で、2番目のグループ化が発生します。具体的には必要とされる一のグループを作成するために、マッピングが **group-into-blocks** 関数を呼び出します。デフォルトの値が"3"である単純型のコンポーネントが各グループのサイズを提供します。デフォルトの値は、定数により与えられます。グループのサイズを変更するために、このサンプルでは、必要に応じて定数を簡単に変更することができます。しかしながら、"サイズ"入力コンポーネントも変更することができ、マッピングが生成されるコード、またはMapForce Server により実行される場合、グループのサイズは、マッピングへのパラメーターとして便利に提供することができます。更に詳しい情報に関しては、次を参照してください： マッピングにパラメーターを与える.

次に、変数の値はターゲット PersonList XML コンポーネントにより与えられます。作成された各グループのためのファイル名は、**concat** 関数を使用して次の部分を連結することにより計算されます:

1. 各部署の名前
2. 文字列"_Group"
3. 現在のシーケンス内のグループの番号（例えば、"1"は、この部署の最初のグループを指します）
4. 文字列".xml"

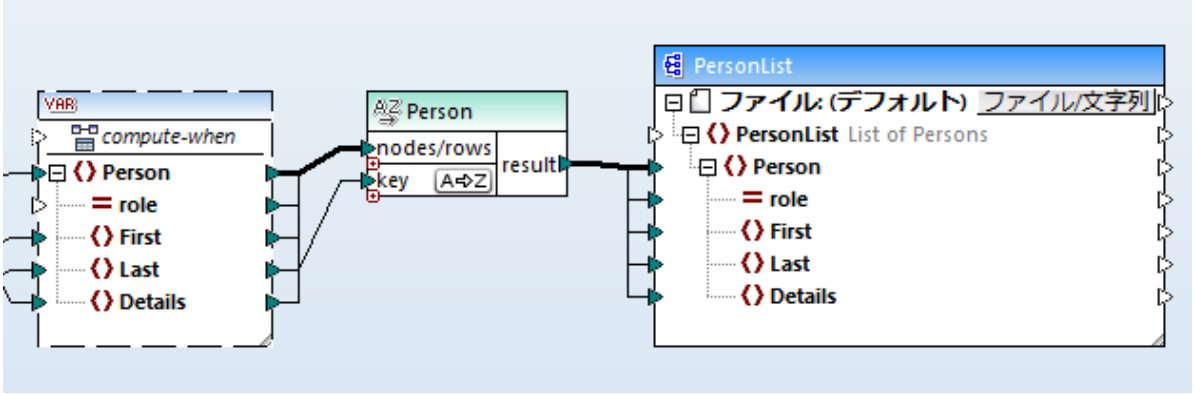
この連結の結果は、変数の Name アイテム内に保管され、ターゲットコンポーネントに動的なファイル名として与えられます。これにより、受信した値のために新規のファイルが作成されます。このサンプルでは、変数は、8個のグループを計算するため、マッピングが実行されると8件の出力ファイルが作成されます。この技術に関する更に詳しい情報は、次を参照してください： 複数の入力または出力ファイルを動的に処理.

4.8 データの並べ替え

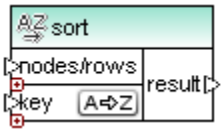
特定のソートキーをベースにした入力データの並べ替えを行う場合、ソートコンポーネントを使用してください。ソートコンポーネントは、XSLT2、XQuery、内蔵の実行エンジンにてサポートされます。

マッピングに並べ替えコンポーネントを追加するには、以下を行います:

- 既存の接続を右クリックして、コンテキストメニューから「Insert Sort: Nodes/Rows」を選択します。並べ替えコンポーネントを自動的に挿入し、ソースとターゲットコンポーネントに自動的に接続します。例えば、下のマッピングでは、並べ替えコンポーネントが変数とXML コンポーネント間に挿入されます。(並べ替えるフィールドである) 並べ替えキーのみを手動で接続します。



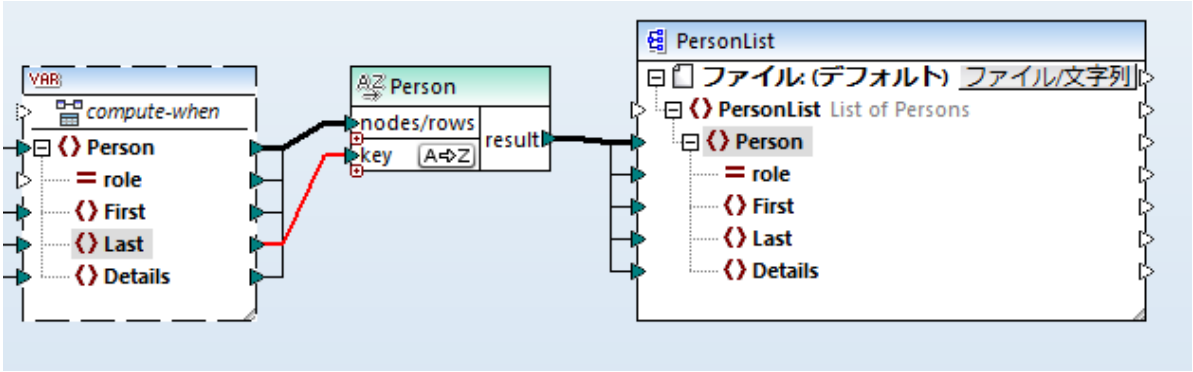
- 「挿入」メニューから「並べ替え」(または、「並べ替え」  ツールバーボタン) をクリックします。「未接続」フォーム内に並べ替えコンポーネントが挿入されます。



ソースコンポーネントに接続が構築されると、タイトルバー名は、`nodes/rows` アイテムに接続されているアイテムの名前に変更されます。

並べ替えるアイテムを定義する:

- 並べ替えコンポーネントの `key` パラメーターを並べ替えるアイテムに接続します。例えば、下のマッピングでは、`Person` nodes/rows は、フィールド `Last` に従い並べ替えられます。

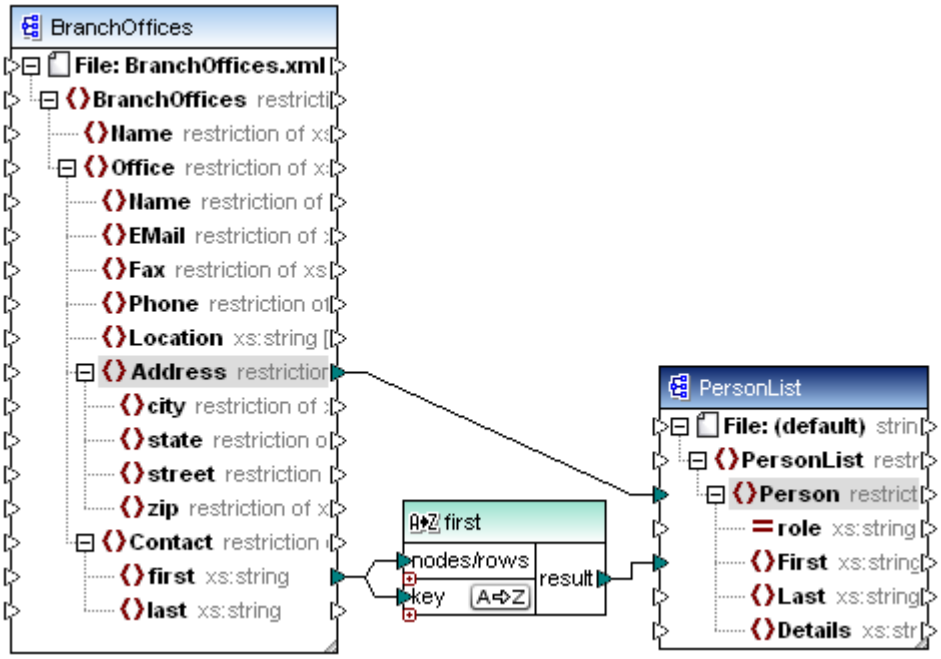


並べ替えの順序の変更:

- 並べ替えコンポーネント内の A⇨Z アイコンをクリックすると、Z➔A に変更され、並べ替えの順序は逆の順序になったことを示します。

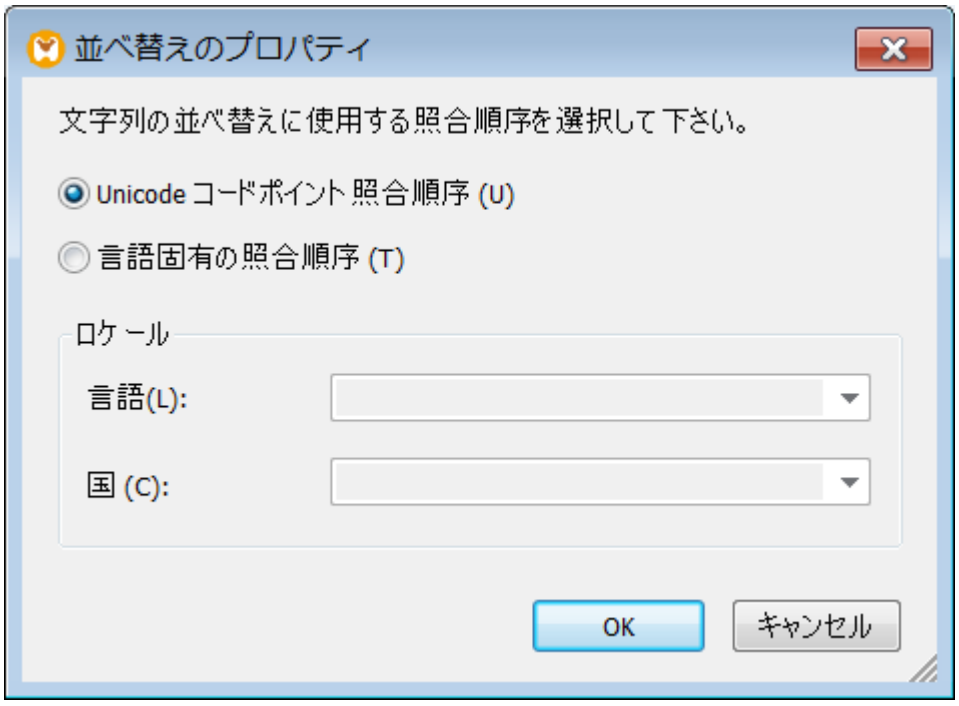
単純型アイテムにより構成される入力データの並べ替え:

- アイテムを並べ替えコンポーネントの `nodes/rows` と `key` パラメーターに接続します。下のマッピングでは、単純型 `first` 要素が並べ替えられます。



言語特有のルールを使用して文字列を並べ替える:

- 並べ替えコンポーネントのヘッダーをダブルクリックして、「並べ替え」プロパティダイアログボックスを開く。

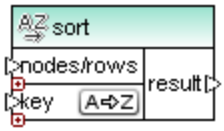


Unicode コードポイント照合: この(デフォルト)オプションにより、コードポイントの値をベースにした比較/並べ替えが行われます。コードポイント値とはUnicode コンソーシアムにより採用されたUniversal Character Set(UCS) における絶対文字に割り当てられた整数のことです。このオプションでは複数の言語やスクリプト間で並べ替えを行うことが可能になります。

言語固有の照合: このオプションにより、並べ替えを行う際にベースとなる言語や国を指定することができます。このオプションは内蔵の実行エンジン(BUILT-IN) が選択されている際にサポートされ、XSLT におけるサポートはコードの実行に使用されるエンジンに左右されます。

4.8.1　複数のキーを使用した並べ替え

マッピングに並べ替えコンポーネントを追加した後、`key` という名前の並べ替えキーがデフォルトで作成されます。

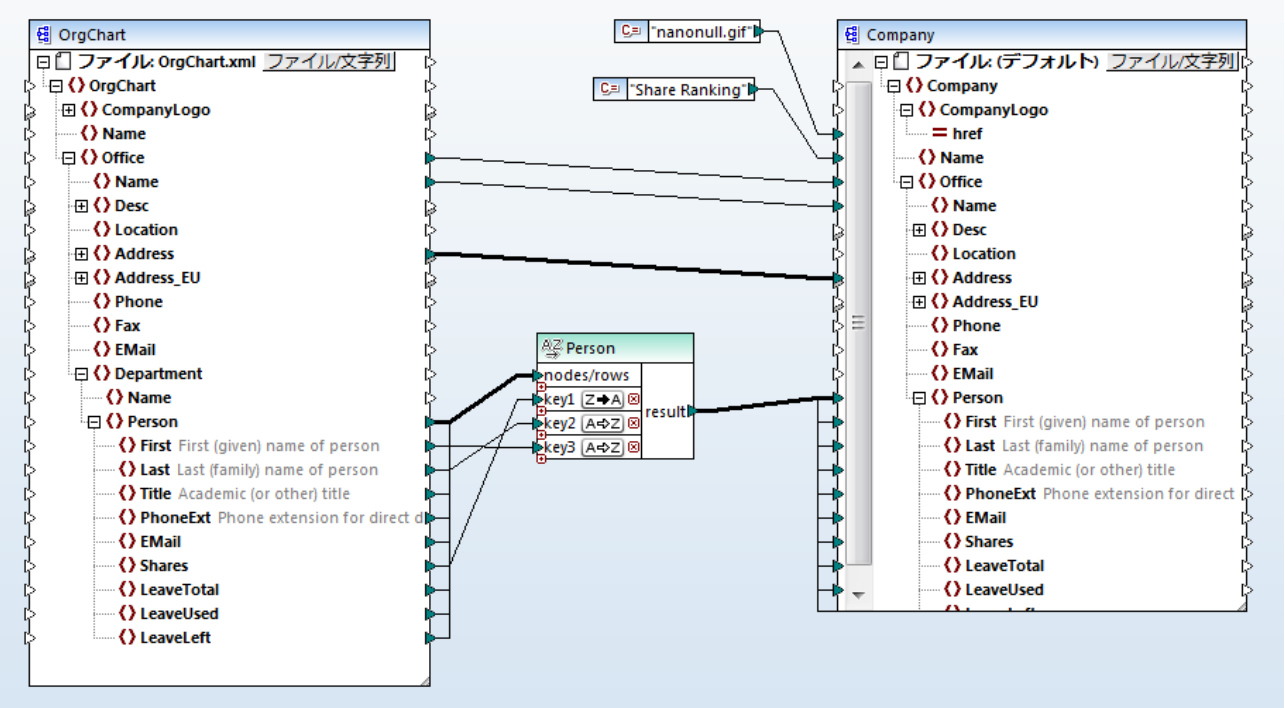


デフォルトの並べ替えコンポーネント

複数のキーを使用して並べ替える場合、並べ替えコンポーネントを以下のように調整します:

- 「**キーの追加**」(⊕) アイコンをクリックして新しいキーを追加します(例えば、下のマッピング内の`key2`)。
- 「**キーの削除**」(⊗) アイコンをクリックしてキーを削除します。
- ⊕ アイコンに接続をドロップして、キーを追加し、接続します。

複数のキーによる並べ替えを示すマッピングは、次のパスで検索することができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\SortByMultipleKeys.mfd**.



SortByMultipleKeys.mfd

上のマッピングでは、`Person` レコードは3つの並べ替えキーにより並べ替えられています:

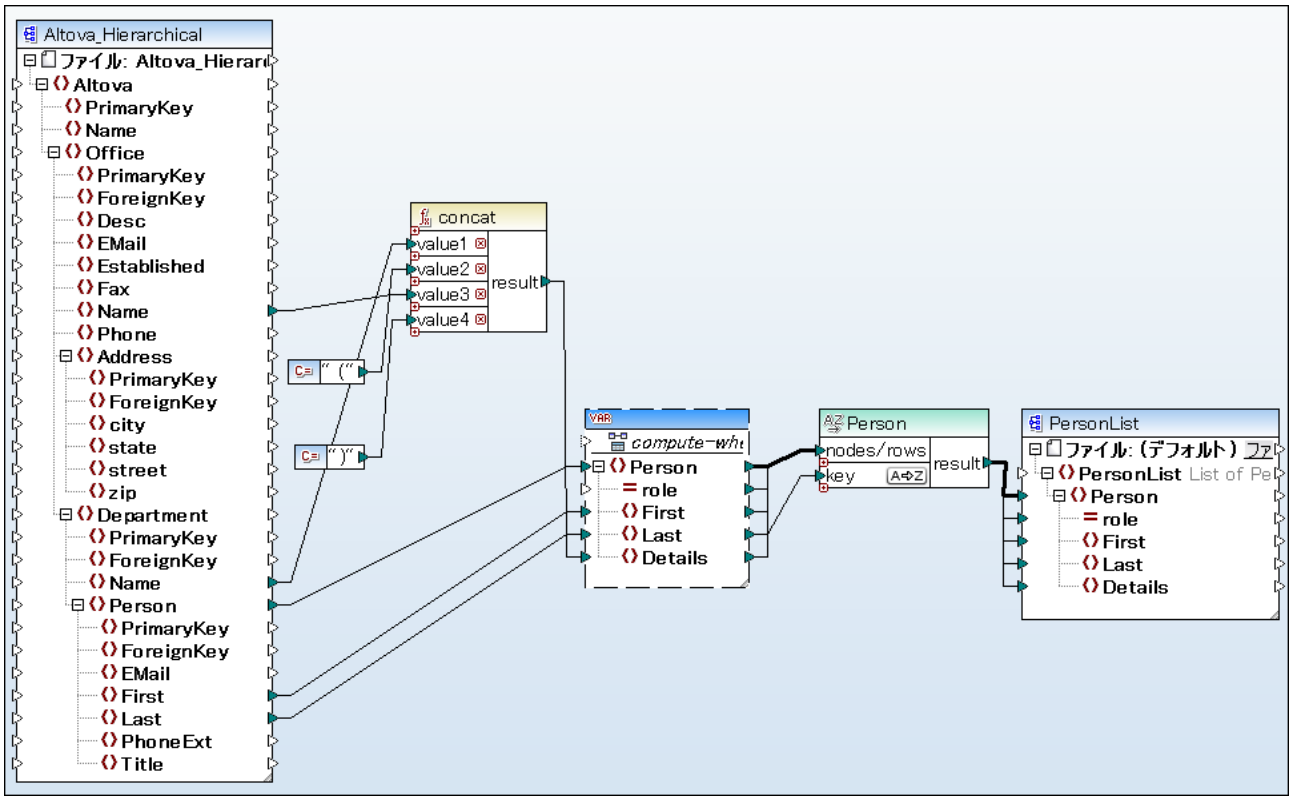
1. `Shares` (個人が所有するシェア数)
2. `Last` (姓)
3. `First` (名)

並べ替えコンポーネント内の並べ替えキーの位置は、並べ替えの優先順序を決定します。例えば、上のマッピングでは、所有するシェア数により並べ替えられています。これは、最も高い優先順位を持つ並べ替えキーです。所有するシェア数が同じの場合も同様で、姓により並べ替えられます。最後に、同数のシェアを持つ同じ性を持つ個人は、名を考慮して並べ替えられます。

各キーの並べ替えは異なる順番であることができます。上のマッピングでは、キー `Shares` は降順の並べ替え順序(Z-A)が与えられていますが、その他2つのキーには昇順の並べ替え順序 (A-Z) が与えられています。

4.8.2 変数を使用して並べ替える

希望する結果を達成するためにマッピングに中間変数を追加する必要がある場合があります。このサンプルでは、XML ファイルから記録を抽出し、中間変数を使用して並べ替える方法について説明されています。このサンプルには、次のパスで検索することのできるマッピングのサンプルが存在します: <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Altova_Hierarchical_Sort.mfd。



Altova_Hierarchical_Sort.mfd

このマッピングは、**Altova_Hierarchical.xml** という名前のソースXMLファイルからデータを読み取り、ターゲットXMLファイルに書き込みます。上に示されるとおり、ソースXMLには架空の企業の情報が含まれています。企業はオフィスに分岐されています。オフィスは部署に分岐されており、部署は社員に分岐されています。

ターゲットXMLコンポーネント、`PersonList`、は`Person`レコードのリストが含まれています。`Details`アイテムは、社員が属するオフィスと部署の情報が保管されています。

ソースXMLから全ての社員を抽出し、姓をアルファベット順に並べ替えることが目的です。また、オフィスと部署名は、`Details`アイテムに書き込まれる必要があります。

この目的を達成するために、このサンプルは、次のコンポーネントの型を使用します:

1. **`concat`** 関数: このマッピング内では、この関数は、フォーマット`Office(Department)`で文字列を返します。オフィス名、部署名、かっこ開始し終了する2つの定数を入力として取ります。関数をマッピングに追加するも参照してください。
2. 中間変数: 変数の役割は同じマッピングコンテキストに個人に関連する全てのデータを運ぶことです。変数によりマッピングがそれぞれの個人のコンテキストで、個人の部署とオフィスを探します。すなわち、変数は、個人が所属するオフィス名と部署名を「記憶」することを意味します。変数が不在の場合、コンテキストは、(同じXMLスキーマを使用して)正確ではなくマッピングは希望しない結果を生成します。(マッピングの実行方法の詳細に関しては、次を参照してください: マッピングルールと戦略)。変数はXMLファイルの構造を複製することに注意してください。これにより、全てコピー接続を使用して、ターゲットに並べ替えの結果を接続することができます。次も参照してください: 変数の使用 と 全てコピー接続.
3. 実際の並べ替えを行う、並べ替えコンポーネント: `Sort` コンポーネントのキー入力は、姓により個人の記録を並べ替える変数の`Last`アイテムに接続されていることに注意してください。

4.9 データのグループ分け

マッピングがソート、またはお行をグループ分けする必要がある場合、次のMapForce 内蔵関数を使用してこの目的を達成することができます:

- `group-by`
- `group-adjacent`
- `group-into-blocks`
- `group-starting-with`
- `group-ending-with`

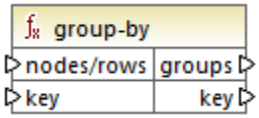
マッピング上でこれらの関数を使用する場合、ライブラリウィンドウからドラッグしてマッピングエリアにドロップします。関数をマッピングに追加するも参照してください。

メモ グループ関数は次の変換言語内で使用することができます: XSLT 2.0、XSLT 3.0、C++、C#、Java、Built-In。

次のセクションではグループ関数のために使用することのできる一般的なサンプルが挙げられています。これらのサンプルは次のデモマッピングが伴います: <**マイドキュメント**>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥**GroupingFunctions.mfd**。デモマッピングには各関数のために1つずつ与えるための複数の変換が含まれています。一度にプレビュー可能な出力は1つであるため、希望する変換に適用するプレビュー ボタンを「出力」タブをクリックする前に、にクリックしてください。

group-by

`group-by` 関数は、指定するグループキーに従いレコードのグループを作成します。



例えば、下で示される抽象的な変換では、グループキーは、「Department」です。トータルで3つの一意の部署が存在するため、`group-by` を適用すると3つのグループが作成されます:

Department	Name
Administration	Vernon Callaby
Marketing	Susi Sanna
Engineering	Michelle Butler
Engineering	Fred Landis
Administration	Frank Further

group-by()

Department	Name
Administration	Vernon Callaby
Administration	Frank Further

Department	Name
Marketing	Susi Sanna

Department	Name
Engineering	Michelle Butler
Engineering	Fred Landis

詳細に関しては、group-by 関数を参照してください。

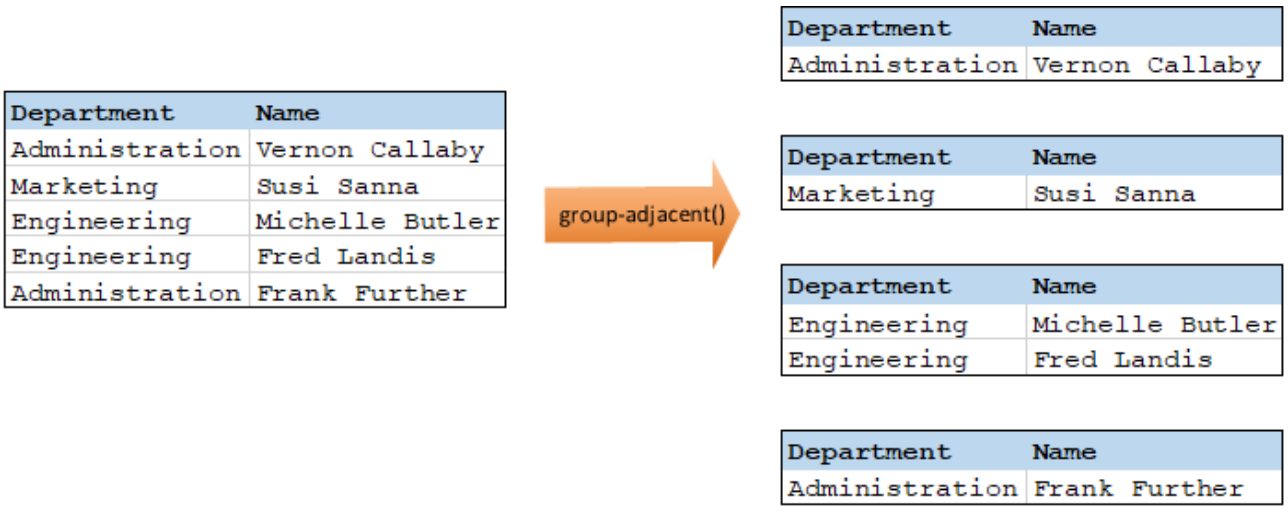
group-adjacent

group-adjacent 関数は**group-by** 関数と同様、グループキーを引数としてグループ分けすることを必要とします。**group-by** とは異なり、この関数は次のキーが異なる都度新規のグループを作成します。2つの隣接するレコードが同じキーを持つ場合、これらのアイテムは同じグループに配置されます。

例えば、下記の抽象的な変換に示されるように、グループキーは「Department」です。このダイアグラムの左側は入力データを示し、右側はグループ分けの後の出力データを示しています。次は、変換の実行中に発生します:

- 最初は、最初のキー「Administration」は新規のグループを作成します。
- 次のキーは異なるため、2番目のグループである「Marketing」が作成されます。
- 3番目のキーもまた異なるための、「Engineering」と呼ばれる他のグループが作成されます。
- 4番目のキーは3番目と同じキーです。このため、レコードは既存のグループに配置されます。
- 最後に、5番目のキーは4番目のキーと異なるため、最後のグループを作成します。

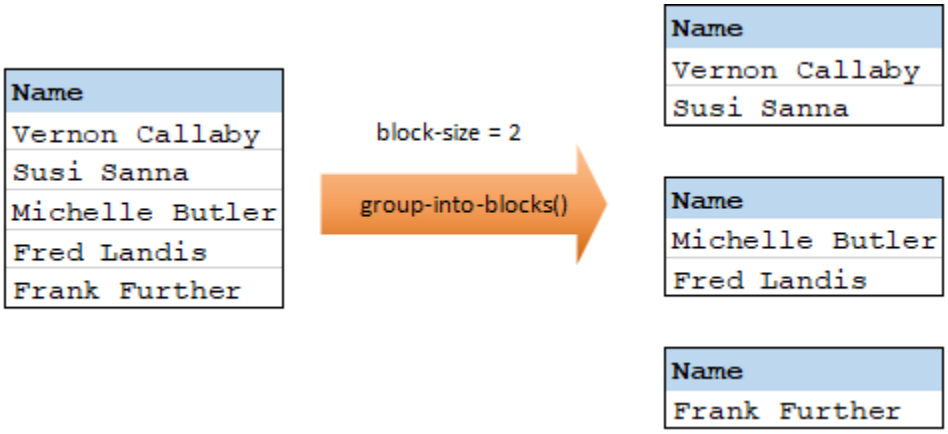
下に説明されるとおり、「Michelle Butler」と「Fred Landis」は隣接する同じキーを持つために一緒にグループ分けされています。しかしながら、「Vernon Callaby」と「Frank Further」は同じキーを所有しているにもかかわらず隣接していないため、異なるグループに分けれています。



詳細に関しては、group-adjacent 関数を参照してください。

group-into-blocks

group-into-blocks 関数は、N が`block-size` 引数に与えられる値である、N アイテムを含む等価のグループを作成します。最後のグループは、ソース内のアイテムの数に従い、N つのアイテム、または少ない数のアイテムを含むことができます。下のサンプルでは、`block-size` は2 です。総数で5つのアイテムが存在し、最後のアイテムを除き、それぞれのグループは2つのアイテムを含んでいます。



詳細に関しては、group-into-blocks 関数を参照してください。

group-starting-with

group-starting-with 関数はブール式の条件を引数として取ります。関数はブール式の条件がtrue の場合、条件を満たすレコードから、新規のグループが作成されます。下のサンプルでは、「Key」が「heading」と等価であることが条件です。最初と4番目のレコードのために、この条件がtrue の場合、2つのグループが結果として作成されます:

メモ 条件を満たす最初のレコードの前にレコードが存在する場合、追加のグループが作成されます。例えば、「line」レコードが最初の「heading」レコードよりも多く存在する場合、これらは新しいグループ内に配置されます。

詳細に関しては、group-starting-with 関数を参照してください。

group-ending-with

group-ending-with 関数はブール式の条件を引数として取ります。関数はブール式の条件がtrue の場合、条件を満たすレコードが最後になるように、新規のグループが作成されます。下のサンプルでは、「Key」が「trailing」と等価であることが条件です。3番目と5番目のレコードのために、この条件がtrue の場合、2つのグループが結果として作成されます:

メモ　　条件を満たす最後のレコードの前にレコードが存在する場合、追加のグループが作成されます。例えば、「line」レコードが最後の「heading」レコードよりも多く存在する場合、これらは新しいグループ内に配置されます。

詳細に関しては、group-ending-with 関数を参照してください。

4.9.1　例: キー別にレコードをグループ分けする方法

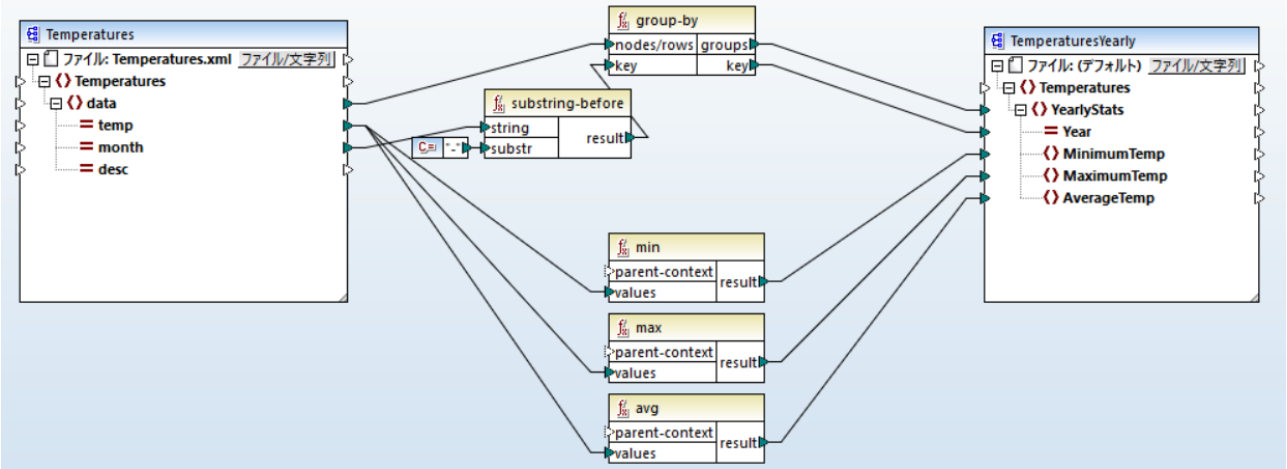
このサンプルは、`group-by` 関数の手助けと共にグループ分けする方法、および、データを集計する方法を示しています。このサンプルには次のパスで使用することのできるデモマッピングが付随しています: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\GroupTemperaturesByYear.mfd**。このマッピングは、月ごとの気温を含むXMLファイルからデータを読み取ります:

```
<Temperatures>
   <data temp="-3.6" month="2006-01" />
   <data temp="-0.7" month="2006-02" />
   <data temp="7.5" month="2006-03" />
   <data temp="12.4" month="2006-04" />
   <data temp="16.2" month="2006-05" />
   <data temp="19" month="2006-06" />
   <data temp="22.7" month="2006-07" />
   <data temp="23.2" month="2006-08" />
   <data temp="18.7" month="2006-09" />
   <data temp="11.2" month="2006-10" />
   <data temp="9.1" month="2006-11" />
   <data temp="0.8" month="2006-12" />
   <data temp="-3.2" month="2007-01" />
   <data temp="-0.3" month="2007-02" />
   <data temp="6.5" month="2007-03" />
   <data temp="10.6" month="2007-04" />
   <data temp="19" month="2007-05" />
   <data temp="20.3" month="2007-06" />
   <data temp="22.3" month="2007-07" />
   <data temp="20.7" month="2007-08" />
   <data temp="19.2" month="2007-09" />
   <data temp="12.9" month="2007-10" />
   <data temp="8.1" month="2007-11" />
   <data temp="1.9" month="2007-12" />
</Temperatures>
```

このマッピングのビジネスの条件は2段階で構成されています:

1. 各年の気温をグループ分けします。
2. 各年度の最低、最高、および平均機能を検索します。

最初の目的を達成するには、マッピングは**group-by** 関数を呼び出します。2番目のゴールを達成するには、**min**、**max** および**avg** 集計関数を呼び出します。これらすべての関数はMapForce ビルトイン関数で、ライブラリウィンドウからマッピングにドラッグして追加することができます。関数をマッピングに追加する を参照してください。



GroupTemperaturesByYear.mfd

MapForce がマッピングを実行する方法は（そして読み取りを開始するために奨励されるアプローチ）ターゲットコンポーネントの一番上のアイテムを確認することです。このサンプルでは、**YearlyStats** アイテムが**group-by** 関数により返される各グループのために作成されます。**group-by** 関数は**key** 入力に接続が作成されると、最初の引数として、すべての**data** アイテムをソースとグループから取ります。条件が気温を年度ごとにグループ分けすることであり、年度が最初に取得される必要があります。これを達成するには、**substring-before** 関数は、各 **data** 要素の**month** 属性から年度の部分を抽出します。具体的には、**month** の値を引数としてとり、の最初の**substr** の発生の前の部分に返します。上記の通り、このサンプルでは、**substr** は点線の文字のセットであるため、値「2006-01」を与えられると、関数は「2006」を返します。

最後に、**MinimumTemp**, **MaximumTemp**、と**AverageTemp** の値は、これらのアイテムを対応する集計関数に接続することにより取得することができます: **min**、**max**、と**avg**。全ての3つの関数は、ソースコンポーネントから読み取られた入力として温度のシーケンスを取ります。
これらの関数は、各グループのコンテキストと既に作業しているため**parent-context** 引数を必要としません。すなわち、作業する各集計関数のためのコンテキストを提供する**data** から**YearlyStats** への親コネクションが存在します。

マッピングの出力をプレビューするには、**Output** タブをクリックしてください。ソースファイルの読み取りで取得された年数とグループの数量が一致することに注意してください。例えば、

```
<Temperatures>
   <YearlyStats Year="2006">
      <MinimumTemp>-3.6</MinimumTemp>
      <MaximumTemp>23.2</MaximumTemp>
      <AverageTemp>11.375</AverageTemp>
   </YearlyStats>
   <YearlyStats Year="2007">
      <MinimumTemp>-3.2</MinimumTemp>
      <MaximumTemp>22.3</MaximumTemp>
      <AverageTemp>11.5</AverageTemp>
   </YearlyStats>
</Temperatures>
```

メモ　　簡単化のために、上記のコードリストには、デモマッピングで使用されている実際の入力と出力より少ないデータを含んでいます。

4.10 フィルターと条件

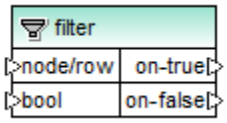
データをフィルターする場合、または値を条件に対して取得する場合、以下のうち1つのコンポーネント型を使用することができます:

- フィルター: Nodes/Rows ( )
- If-Else 条件 ( )

「挿入」メニューから、または「コンポーネントの挿入」ツールバーからこれらのコンポーネントをマッピングに追加することができます。上記のそれぞれのコンポーネントには特定の振る舞いと必要条件があることにご注意してください。差異は下のセクションで説明されています。

ノードと行をフィルターする

XML ノードを含むデータをフィルターする場合、Nodes/Rows をフィルター コンポーネントを使用します。このNodes/Rows をフィルター コンポーネントにより、true または false 条件をベースにし、大きなデータセットからノードのサブセットを抽出することができます。マッピングエリアでのこの構造は以下のように表されます:



上の構造では、に接続されている条件は、**bool** に接続された**node/row** が、**on-true** または**on-false** 出力に接続されているかを決定します。具体的には、条件がtrue の場合、**node/row** は、**on-true** 出力にリダイレクトされます。一方、条件がfalse の場合、**node/row** は、**on-false** 出力にリダイレクトされます。

マッピングがフィルターの条件を満たすアイテムのみを消費する必要がある場合、**on-false** 出力を未接続のままにできます。フィルターの条件を満たさないアイテムを処理する必要がある場合、このようなアイテムがリダイレクトされる、**on-false** 出力をターゲットに接続します。

手順を追ってのマッピングのサンプルは、例: ノードのフィルターを参照してください。.

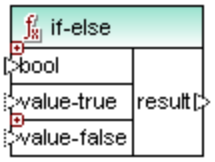
値を条件付きで返す

(ノードまたは行ではない) 単一の値を条件付きで取得する必要がある場合、**If-Else 条件** を使用します。If-Else 条件は、ノードまたは行をフィルターするには、適していません。Nodes/Rows をフィルター コンポーネントとは異なり、**If-Else 条件** は、(文字列または整数などの) 単純型を返します。ですから、**If-Else 条件** は、単純型を条件付きで処理するシナリオのみに適しています。例えば、各月の平均気温のリストがあると仮定します:

```
<Temperatures>
   <data temp="19.2" month="2010-06" />
   <data temp="22.3" month="2010-07" />
   <data temp="19.5" month="2010-08" />
   <data temp="14.2" month="2010-09" />
   <data temp="7.8" month="2010-10" />
   <data temp="6.9" month="2010-11" />
   <data temp="-1.0" month="2010-12" />
</Temperatures>
```

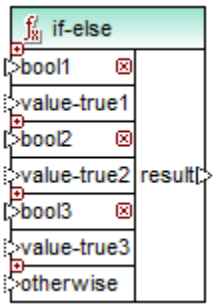
If-Else 条件 により、リスト内の各アイテムのために気温が摂氏20度を超える場合は、値 "高" を、気温が摂氏5度以下の場合は、値 "低" を返すように設定することができます。

マッピング上で**If-Else 条件** の構造は、以下のようになります:



bool に接続された条件がtrue の場合、**value-true** に接続された**result** としての出力になり、条件がfalse の場合、**value-false** に接続された**result** としての出力になります。**result** のデータ型を事前に知ることはできず、これは値のデータ型、により異なり、**value-true** または**value-false**. に接続された値により異なります。重要な点は、常に(文字列および整数などの)。単純型であるということです。入力値を(ノードまたは行などの)複合型に接続することはIf-Else 条件によりサポートされていません。

If-Else 条件は拡張することが可能です。これは、**追加** ( )ボタンをクリックすることにより、コンポーネントに複数の条件を追加できることを意味します。以前に追加された条件を削除するには、ボタンを**削除** ( ) をクリックします。この機能により、条件がtrue の場合は、複数の条件をチェックして、各条件のために異なる値を返すことができます。



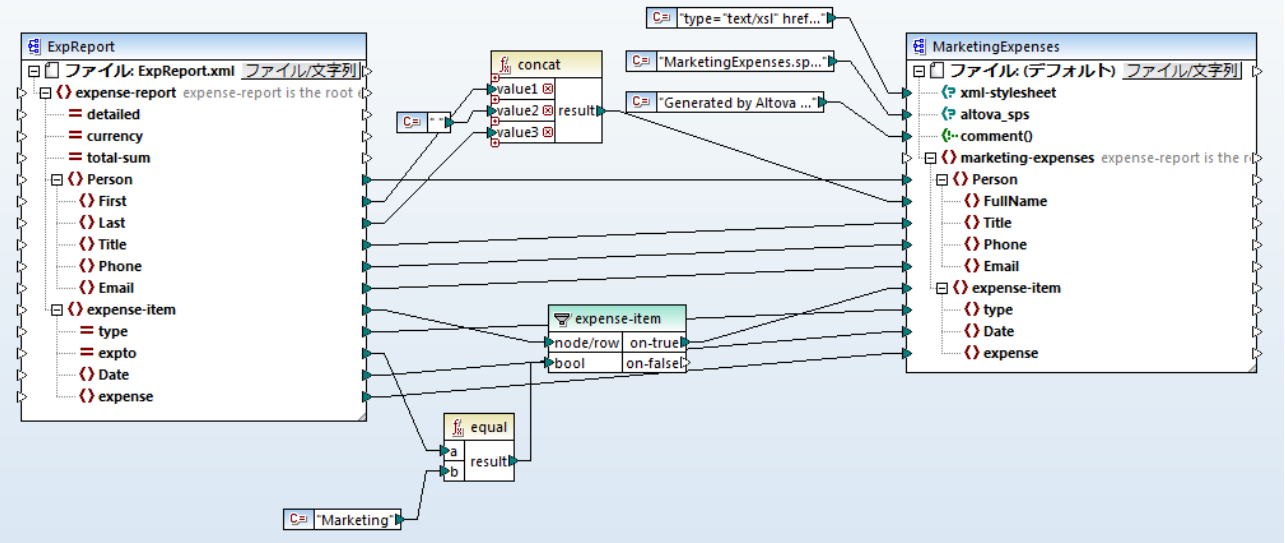
展開されたIf-Else 条件 は上から下へと評価されます(最初の条件が最初にチェックされ、2番目が次にチェックされます)。条件の全てが真ではない場合に値を返す場合は、**otherwise** に接続してください。

手順を追ってのマッピングのサンプルは、例: 値を条件付きで返すを参照してください。

4.10.1 例: ノードのフィルター

この例は、true/false 条件を基にしてノードをフィルターする方法について説明しています。**Filter: Nodes/Rows** ( ) コンポーネント がこの目的を達成するために使用されています。

この例で説明されているマッピングは以下のパスで検索することができます: **<マイドキュメント >\Altova\MapForce2021\MapForceExamples\MarketingExpenses.mfd**。



上に示されるように、このマッピングは、経費の方向k（"ExpReport"）のデータを含むソースXML からデータを読み取り、ターゲット XML（"MarketingExpenses"）に書き込みます。ターゲットとソースの間には複数の他のコンポーネントが存在します。最も関連性の高いコンポーネントは、このトピックの主題を表す **expense-item** フィルター（ ）です。

このマッピングのゴールは、マーケティング部署に属する経費アイテムのみをフィルターすることです。 この目的を達成するために、フィルターコンポーネント がマッピングに追加されました。（フィルターを追加するには、「挿入」メニューをクリックして、「Filter: Nodes/Rows」をクリックします）。

各経費がマーケティング部署に属するかを識別するために、このマッピングは、ソース内の"expto" 属性の値を確認します。この属性は経費がマーケティングの経費である場合、値 "Marketing" を有します。例えば、 下にリストされるコードでは、最初と3番目のアイテムは、マーケティングに属し、2番目のアイテムは開発部署に、4番目は販売部署に属することが示されています:

```
...
   <expense-item type="Meal" expto="Marketing">
      <Date>2003-01-01</Date>
      <expense>122.11</expense>
   </expense-item>
   <expense-item type="Lodging" expto="Development">
      <Date>2003-01-02</Date>
      <expense>122.12</expense>
   </expense-item>
   <expense-item type="Lodging" expto="Marketing">
      <Date>2003-01-02</Date>
      <expense>299.45</expense>
   </expense-item>
   <expense-item type="Entertainment" expto="Sales">
      <Date>2003-01-02</Date>
      <expense>13.22</expense>
   </expense-item>
...
```

マッピングが実行される前のXML 入力

マッピングエリアで、フィルターの**node/row** 入力が、ソースコンポーネント内の**expense-item** ノードに接続されています。これにより、フィルターコンポーネントが処理されなければならないノードのリストを取得することを保証します。

フィルタリングが発生する条件を追加するには、MapForce core ライブラリから **equal** 関数 を追加しました（詳細に関しては、以下を参照してください：関数をマッピングに追加する）。**equal** 関数は、"type" 属性を値 "Marketing" を持つ定数と比較します（定数を追加するには、「**挿入**」メニューをクリックして、**定数**をクリックします）。

条件を満たすアイテムのみをフィルターする必要があるため、フィルターの **on-true** 出力のみをターゲットコンポーネントに接続します。

マッピングの結果をプレビューする準備が整うと、「**出力**」タブをクリックします。MapForce は、フィルターの **bool** 入力に接続されている条件である、それぞれの経費アイテムノードを評価します。条件が true の場合、経費アイテムノードはターゲットにパスされます。それ以外の場合は、無視されます。結果、条件を満たす経費アイテムのみが出力内に表示されます:

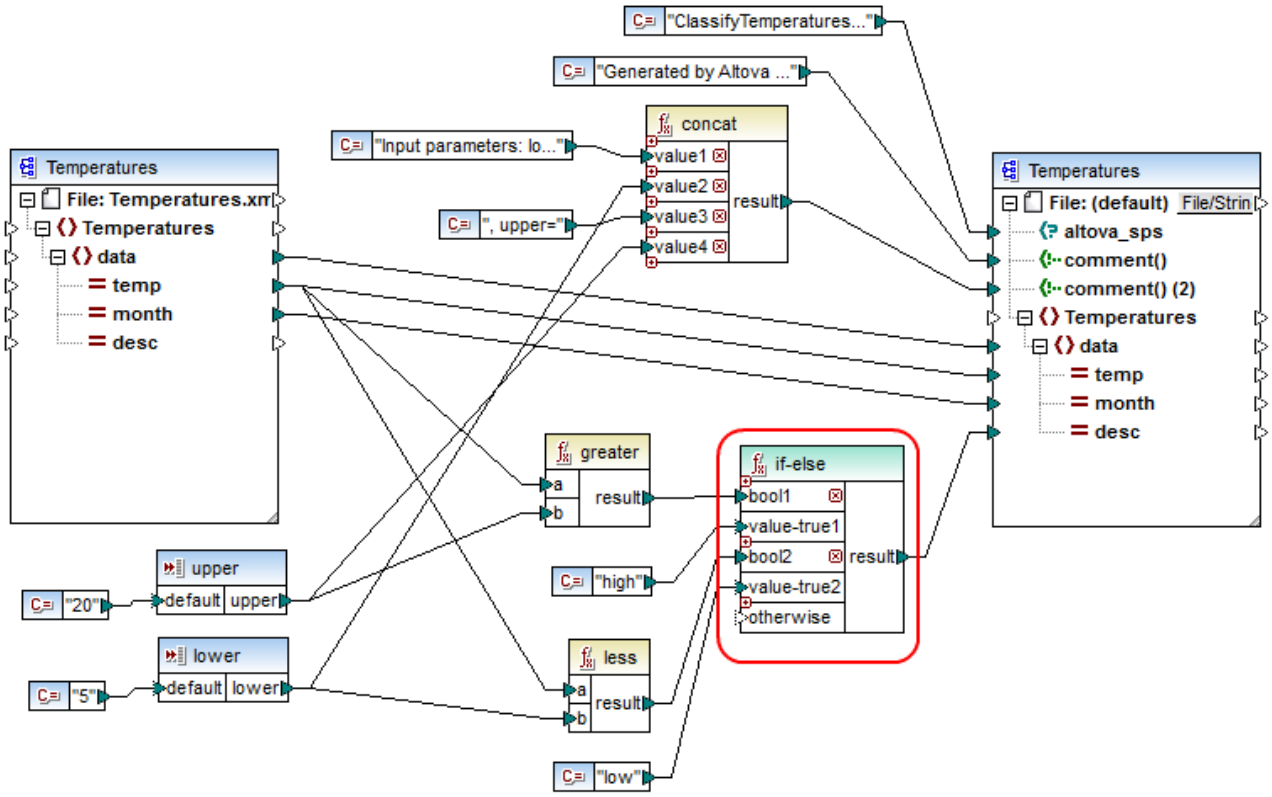
```
...
   <expense-item>
      <type>Meal</type>
      <Date>2003-01-01</Date>
      <expense>122.11</expense>
   </expense-item>
   <expense-item>
      <type>Lodging</type>
      <Date>2003-01-02</Date>
      <expense>299.45</expense>
   </expense-item>
...
```

マッピングの実行後の XML 出力

4.10.2 例: 条件付で値を返す

この例では、true/false 条件を基にして、コンポーネントから単純型の値を返す方法を説明しています。「**If-Else 条件**」( ) がこの目的を達成するために使用されています。**If-Else 条件** とフィルターコンポーネントを混同しないように注意してください。「**If-Else 条件**」は、（文字列、整数などの）単純な値を条件付きで処理する場合のみに適しています。ノードなどの複合値をフィルターする場合、代わりにフィルターを使用してください（以下を参照してください：例: ノードのフィルター）。

この例で説明されるマッピングは以下のパスで検索することができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\ClassifyTemperatures.mfd**。



このマッピングは、気温データを含むソースXML（"Temperatures"）からデータを読み取り、データを同じスキーマに準ずるターゲットXMLに書き込みます。ターゲットとソースの間には複数の他のコンポーネントが存在します。そのうちの1つが、このトピックの主題である（赤でハイライトされている）**if-else** 条件です。

このマッピングのゴールは、ターゲット内の各気温の記録に短い説明を追加することです。具体的には、摂氏20度以上の気温の説明は、"high" と表示され、摂氏5度以下の気温は、"low" と表示されるように設定します。上記以外の場合は、説明が表示されません。

この目的を達成するために、条件付き処理が必要とされます。ですから、If-Else 条件 がマッピングに追加されました。（If-Else 条件を追加するには、**「挿入」** メニューをクリックして、**If-Else 条件**をクリックします）。このマッピングでは、以下の2つの条件を受け入れるために（ボタンを使用して）If-Else 条件が展開されています: **bool1** と**bool2**。

条件自身は、MapForce core ライブラリから追加された`greater` と`less` 関数により追加されます。詳細に関しては、関数をマッピングに追加するを参照してください。関数は、"upper" と"lower" と呼ばれる2つの入力コンポーネントより与えられた値を評価します（入力コンポーネントを追加するには、**「挿入」** メニューをクリックして、**「入力の挿入」**をクリックします。入力コンポーネントに関する詳細については、マッピングにパラメーターを提供するを参照してください

`greater` と`less` 関数は、true または false のどちらかを返します。関数の結果は、ターゲットインスタンスに何が書き込まれるかを決定します。具体的には、ソース内の"temp" 属性の値が、20 よりも大きい場合、定数値 "high" が**if-else** 条件にパスされます。ソース内の"temp" 属性の値が5 よりも小さい場合、定数値 "low" は、**if-else** 条件にパスされます。**otherwise** 入力が接続されていませんので、上記の条件が満たされない場合は、**result** 出力コネクターにパスされる値はありません。

最後に、**result** 出力コネクタは、（各気温の記録に1度ずつ与えられる）この値をターゲット内の"desc" 属性に与えます。
マッピングの結果をプレビューする準備が整うと、**「出力」** タブをクリックします。気温が20度より高い、または5度より低い場合、結果 XML 出力は、"desc" 属性を含むことに注意してください。

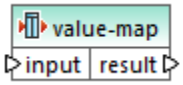
```
...
```

```
   <data temp="-3.6" month="2006-01" desc="low"/>
   <data temp="-0.7" month="2006-02" desc="low"/>
   <data temp="7.5" month="2006-03"/>
   <data temp="12.4" month="2006-04"/>
   <data temp="16.2" month="2006-05"/>
   <data temp="19" month="2006-06"/>
   <data temp="22.7" month="2006-07" desc="high"/>
   <data temp="23.2" month="2006-08" desc="high"/>
...
```

マッピングの実行後の XML 出力

4.11 Value-Map の使用

Value-Map コンポーネントにより *ルックアップ* を使用して値を他の値に置き換えることができます。このようなコンポーネントは1度に2つの値を処理します。このためにマッピング上に2つの **入力** と1つの **結果** が存在します。



Value-Map はアイテムを置き換えるために2つのセット内の個別のアイテムをマップする場合役に立ちます。例えば、数値（1、2、3、4、5、6 および 7）で表示された曜日をそれぞれの曜日（「月曜」、「火曜」など）と置き換えます。同様に月の名前（「1月」、「2月」、「3月」など）を各月の数値表記（1、2、3, etc）. に置き換えます。マッピングのランタイムでは、一致する値はカスタム *ルックアップ* テーブルに従い置き換えられます。両方のセット内の値は異なる型であることができますが、各セットは同じデータ型の値を保管する必要があります。

Value-Map コンポーネントは最初のセット内の各値が2番目のセット内の単一の値に対応する単純な *ルックアップ* に適しています。*ルックアップ* テーブル内で値が見つからない場合、カスタム値、または空の値と置き換える、またはそのままパスすることができます。更に複雑な条件をベースにした値を *ルックアップ* または *フィルター* する場合、コンポーネントのフィルターを代わりに使用します。

重要な点は、コードを生成または MapForce Server 実行可能ファイルをマッピングからコンパイルする場合、*ルックアップ* テーブルデータは生成されたコード、またはファイル内に埋め込まれる点です。この結果、*ルックアップ* テーブルを直接マッピング上で定義することは、データが頻繁に変更されず、（例えば、数百エントリより少ないなど）データの総量が大きくない場合のみ良い選択でしょう。*ルックアップ* データが定期的に変更する場合、マッピング、および生成されたコードを定期的に管理することは難しく、*ルックアップ* データをテキスト、XML、データベース、または Excel として管理する方法のほうが簡単です。

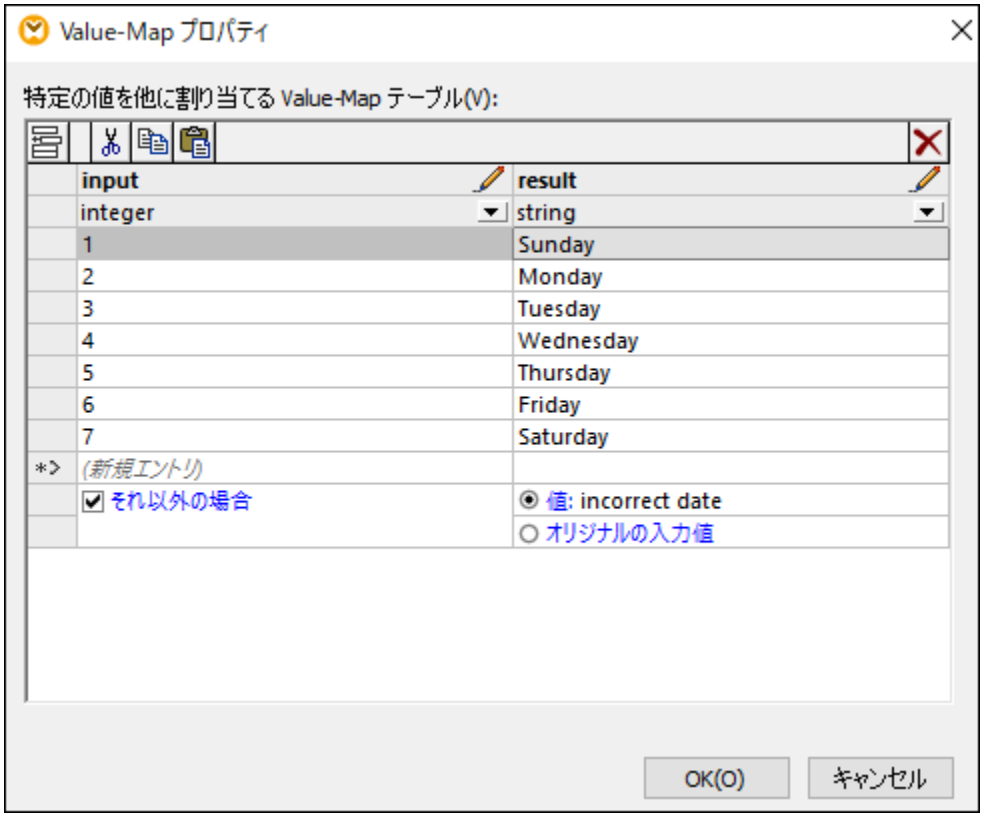
ルックアップ テーブルが大きい場合、*ルックアップ* テーブルによりマッピングの実行が遅くなる場合があります。この場合、代わりに SQL-W here を使用するデータベースコンポーネントを使用することが奨励されます。 MapForce Professional と Enterprise エディションでデータベースコンポーネントを使用することができます。ポータビリティを考慮すると、SQLite データベースは良い候補です。他方、MapForce Server または MapForce Server Advanced Edition を使用してマッピングを実行し *ルックアップ* テーブルの パフォーマンスを向上することもできます。

Value-Map の作成方法

Value-Map コンポーネントをマッピングに追加するには次のいずれかの操作を実行してください：

- 「**Value-Map の挿入**」  ツールバーボタンをクリックします。
- 「**挿入**」メニューから「**Value-Map**」をクリックします。
- 接続を右クリックし「**Value-Map の挿入**」をコンテキストメニューから選択します。

この操作は新規の Value-Map コンポーネントをマッピングに追加します。*ルックアップ* テーブルにアイテムのペアを追加することから開始します。これを行うにはコンポーネントのタイトルバーをダブルクリック、または右クリックし、「**プロパティ**」をコンテキストメニューから選択します。



マッピングのランタイムで、MapForce は Value-Map の **入力** に各値が到達するかをチェックします。*ルックアップ テーブル内の左側の列* に一致する値が存在する場合、元の入力値を*右側の列*からの値と置き換えます。それ以外の場合、任意で以下のオプションの一つに戻るように構成することができます:

- 置換の値。下のサンプル内では、置換の値はテキスト「incorrect date」です。置換の値をテキストを入力せずに空に設定することもできます。
- 元の入力の値。これはテーブル内で一致が存在しない場合、元の入力値がマッピングにパスされることを意味します。

それ以外の場合の条件を構成しない場合、Value-Map は一致が見つからない場合**空のノード**を返します。この場合、ターゲットコンポーネントには何もパスされず、出力には空のフィールドが含まれます。このような場合が発生することを回避するために、それ以外の場合の条件を構成、または、[substitute-missing 関数] を使用します。

空の置換値の設定と、それ以外の場合の条件を設定しない事は異なります。最初のケースは、出力でフィールドは生成されますが、空の値が存在します。後者のケースは値を囲むフィールド（またはXML 要素）が作成されません。詳細に関しては、[例: 職位の置換] を参照してください。

Value-Map の設定方法

ルックアップ テーブル内で、必要な数の値のペアを定義することができます。値を手動で入力する、または、テキスト、CSV、またはExcel ファイルからテーブルのデータをコピーして貼り付けることができます。一般的なブラウザーを使用してHTML ページからテーブルをコピーして貼り付けることができます。 テキスト ファイルからデータをコピーするとフィールドはタブ文字で区切られる必要があります。更に、MapForce は多くの場合コンマ、またはセミコロンにより区切られているテキストを認識することができます。

ルックアップ テーブルを作成する際に以下の点を考慮してください:

1. 左の列内のすべてのアイテムは一意である必要があります。それ以外の場合、具体的に一致するアイテムを決定することは不可能です。
2. 同じ列に属するアイテムは同じデータ型である必要があります。ルックアップテーブル内の各列の上のドロップダウンリストからデータ型を選択することができます。ブール型を変換する必要がある場合、テキスト「true」または「false」を文字通り入力してください。このシナリオの説明は例: 曜日の置換を参照してください。

MapForce がルックアップテーブル内で無効なデータを検知すると、ピンク色でハイライトされたエラーメッセージが表示されます。例:

Value-Map コンポーネントに外部ソースからデータをインポートする方法:

1. (例えば、Excel などの) ソースプログラム内の注目するセルを選択します。これは単一のデータ列、または2つの隣接した列であることができます。
2. 外部プログラムの「コピー」コマンドを使用してクリップボードにデータをコピーします。
3. Value-Map コンポーネント上で、データを張り付ける行の前をクリックします。
4. Value-Map コンポーネント上の「クリップボードからテーブルを張り付ける」ボタンをクリック、または「Ctrl+V」または「Shift+Insert」を押します。

メモ 「クリップボードからテーブルを張り付ける」ボタンはソースからデータがコピーされている場合のみ有効化されています(すなわち、クリップボード上にデータが存在します)。

クリップボードデータに複数の列が含まれる場合、最初の2列の行のデータのみがルックアップテーブルに挿入されます。以降の行は無視されます。既存の値のトップの単一行からデータを張り付ける場合、コンテキストメニューが表示され、新規の行としてクリップボード上のデータが挿入されるか、または既存の行が上書きされるかが問われます。このため、テーブル内の既存の値を上書きする必要がある場合、クリップボードには最低限ではなく1行のみのデータが含まれていることを確認してください。

既存の行の前に行を手動で挿入するには、最初に注目する行をクリックし、「挿入」ボタンをクリックします。

既存の行を削除するには、左側にマウスのボタンを押したまま行をドラッグして(上または下の)新しい位置に移動します。

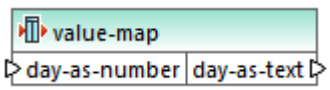
他の箇所で後に張り付けるためのコピーまたは切り取りを行うには、行を選択し「コピー」ボタン(または「切り取り」ボタンをそれぞれ)クリックします。必ずしも連続していない複数の行をコピーまたは切り取ることもできます。複数の行を選択するには、行を選択しながら「Ctrl」キーを押し続けます。切り取られた、またはコピーされたテキストは両方の列からの値を含むことに注意してください。1つの行からの値のみを切り取る、またはコピーすることはできません。

行を削除するには「削除」ボタンをクリックします。

左と右の列をスワップするには、「スワップ」ボタンをクリックします。

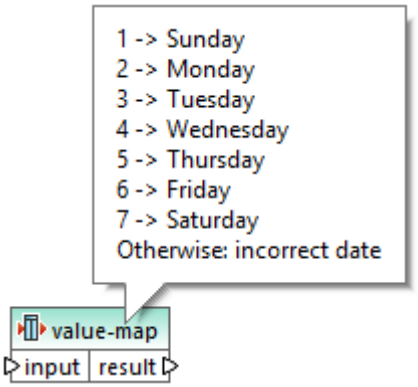
Value-Map パラメーターの名前を変更する方法

デフォルトではValue-Map コンポーネントの入力パラメータは「入力」と呼ばれ、出力パラメーターは「結果」と呼ばれます。マッピングをより簡潔にするために、それぞれの名前の横の「編集」ボタンをクリックしてこれらのパラメーターの名前を任意で変更することができます。以下は、カスタムパラメーター名を持つValue-Map のサンプルです:



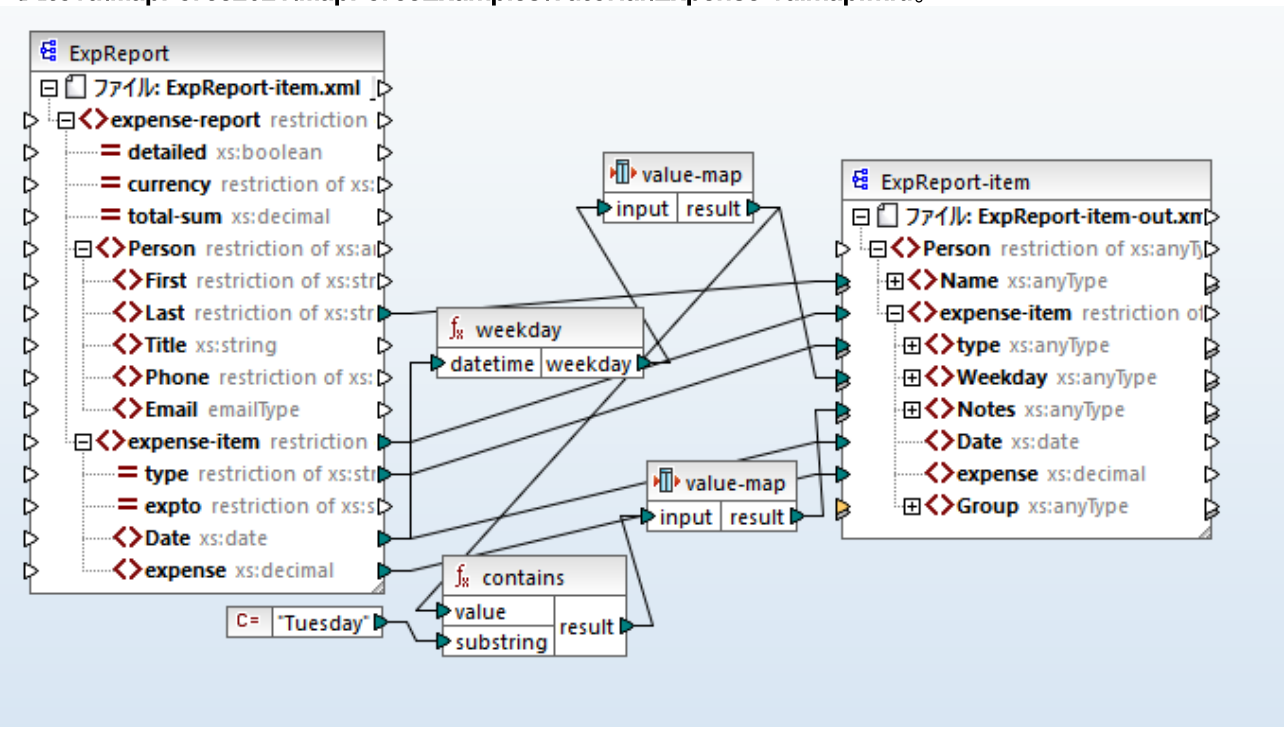
Value-Map をプレビューする方法

Value-Map の作成後、コンポーネントのタイトルバーにマウスをかざすことによりマッピングから直接実装をプレビューすることができます:



4.11.1 例: 曜日の置換

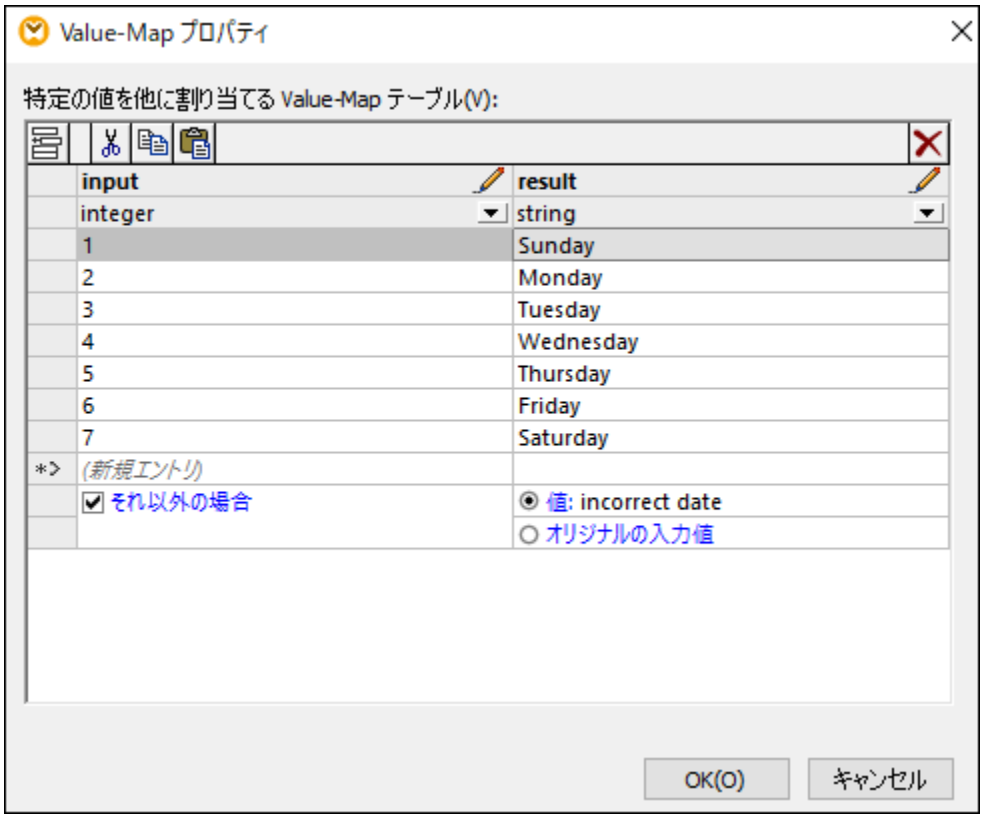
このサンプルは整数の値を曜日の名前 (1 = 日曜日、2 = 月曜日 など) と置き換える Value-Map を説明しています。このサンプルは以下のパスで見つけることのできるマッピングが存在します: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\Expense-valmap.mfd**。



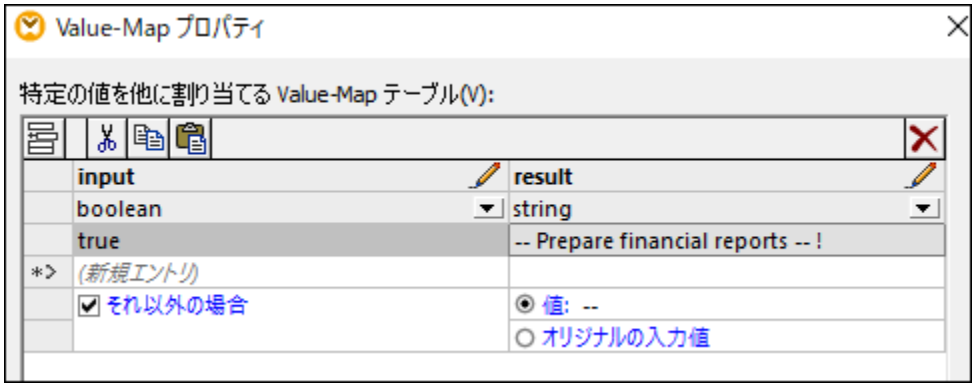
Expense-valmap.mfd

このマッピングはソースファイル内の**Date** アイテムから曜日を抽出し、テキスト内で数値に変換し、ターゲットコンポーネントの**Weekday** アイテムに書き込みます。具体的には、以下が行われます:

- `weekday` 関数はソースファイル内の **Date** アイテムから曜日の数値を抽出します。この関数の結果は 1 から 7 の範囲の整数です。
- 最初の Value-Map コンポーネントは整数を曜日（1 = 日曜日、2 = 月曜日 など）に変換します。コンポーネントは 1-7 範囲外の無効な整数が発生するとテキスト「incorrect date」（無効な日付）を返します。



- 曜日に「Tuesday」が含まれている場合、テキスト「Prepare Financial Reports」はターゲットコンポーネント内で **Notes** アイテムに書き込まれます。これはブール型 **true** または **false** 値を2番目の Value-Map コンポーネントに渡す `contains` 関数を使用して達成することができます。2番目の Value-Map には次の構成が存在します:



上で説明されている Value-Map は以下のように理解されるべきです:

- ブール値 **true** が発生すると、テキスト「— Prepare financial reports — ! 」に変換されます。それ以外の場合、テキスト「—」が返されます。

最初の列のデータ型が「boolean」に設定されていることにご注意してください。これは入力ブールの値 **true** が認識されていることを保証しています。

4.11.2 例: 職位の置換

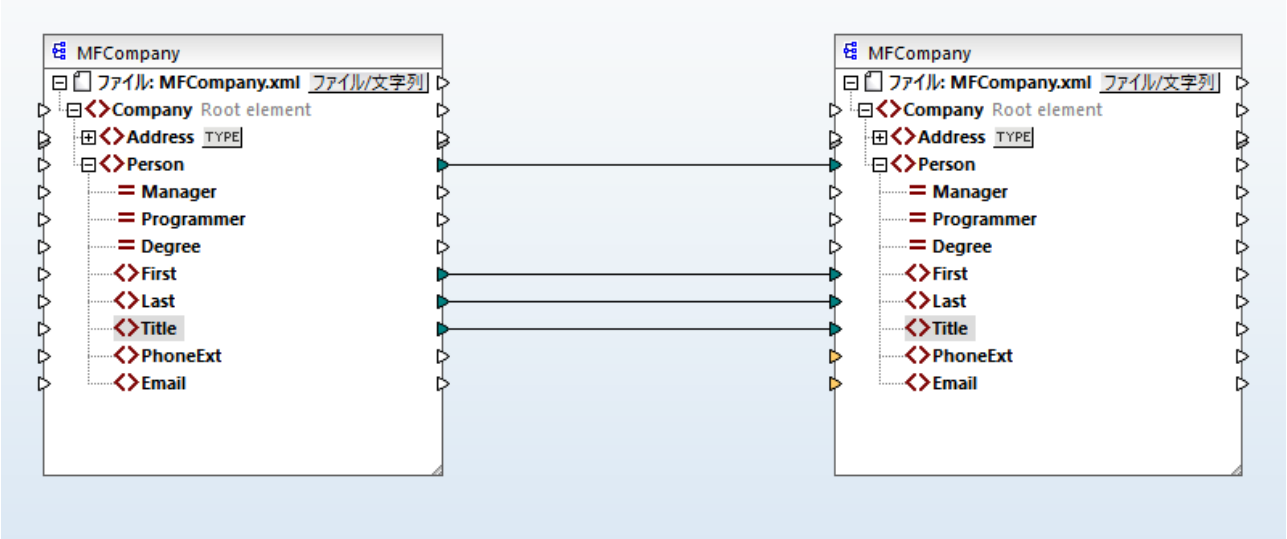
このサンプルはValue-Map コンポーネントを使用してXML ファイル内の特定の要素の値を置き換える方法(すなわち定義済みのルックアップテーブルを使用する方法)を示しています。

このサンプルで必要とされるXML ファイルは以下のパスで見つけることができます: <**マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\MFCompany.xml**。他のデータ以外に、企業の従業員と職位などの情報を保管します。例:

```
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Software Engineer</Title>
</Person>
<Person>
   <First>Lui</First>
   <Last>King</Last>
   <Title>Support Engineer</Title>
</Person>
<Person>
   <First>Steve</First>
   <Last>Meier</Last>
   <Title>Office Manager</Title>
</Person>
```

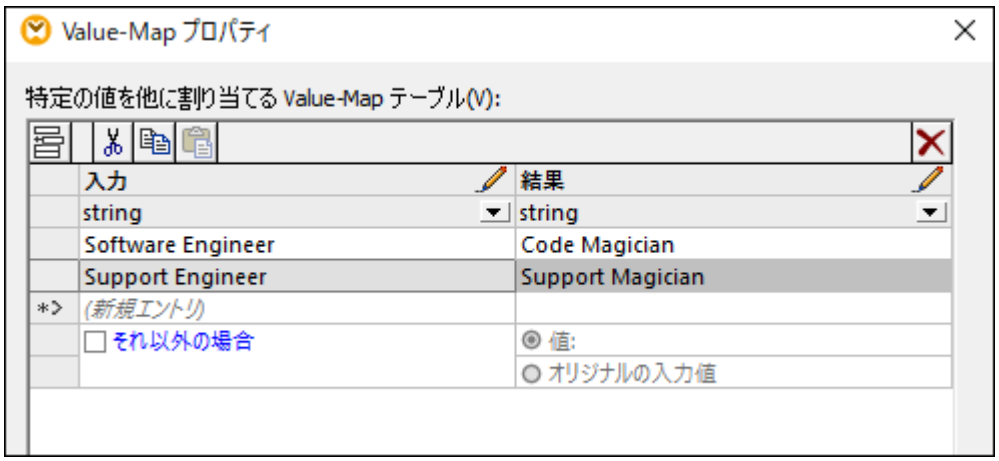
XML ファイル内の職位の一部を置き換える必要があると仮定します。具体的には職位「Software Engineer」は「Code Magician」と置き換えられます。また、職位「Support Engineer」は「Support Magician」と置き換えられます。その他の職位はすべて変更されません。

この目的を達成するために、「**XML スキーマ/ファイルの挿入**」  ツールバーボタンをクリック、または「**挿入 | XML スキーマ/ファイル**」メニューコマンドを実行してマッピングエリアにファイルを追加します。次に、マッピング上のXML コンポーネントをコピーして貼り付け、以下に示される通りに接続を作成します。自動的に不必要な接続の作成を防ぐために  「**子要素の自動的な接続の切り替え**」ツールバーオプションを最初にオフに切り替える必要がある場合があります。



First、**Last**、および**Title** 要素に変更を加えることなく作成されたマッピングは**Person** 要素をターゲットファイルにコピーします。

必要とされる職位を置き換えるために、Value-Map コンポーネントを追加します。2つの**Title** 要素間の接続を右クリックし、コンテキストメニューから「**Value-Map の選択**」を選択します。Value-Map プロパティを下に表示される通りセットアップします:

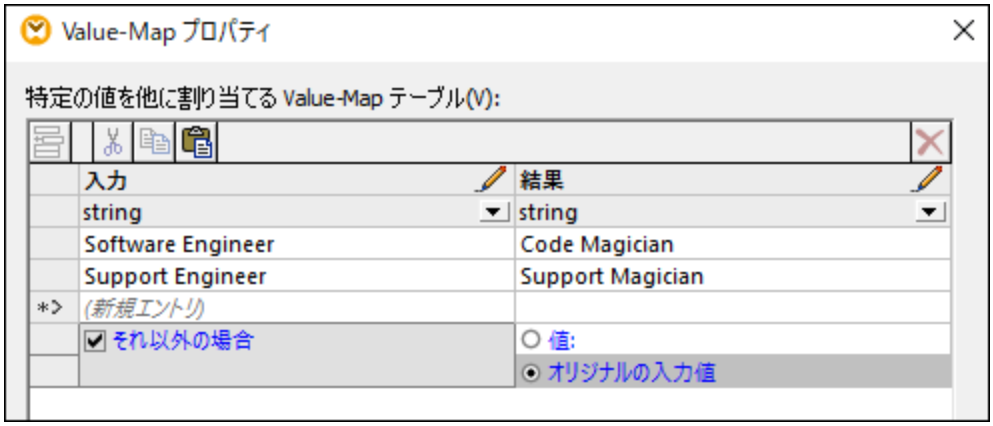


上記のセットアップに従い、「Software Engineer」は「Code Magician」と置き換えられます。そして、「Support Engineer」は「Support Magician」と置き換えられます。**それ以外の場合** 条件はまだ指定されていません。このため、Value-Map は職位が「Software Engineer」と「Support Engineer」以外の場合は *空のノード* を返します。この結果、出力タブをクリックし、マッピングをプレビューすると、**Person** 要素の一部には不足している**Title** が存在します。例:

```
<Person>
   <First>Vernon</First>
   <Last>Callaby</Last>
</Person>
<Person>
   <First>Frank</First>
   <Last>Further</Last>
</Person>
```

```
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Code Magician</Title>
</Person>
```

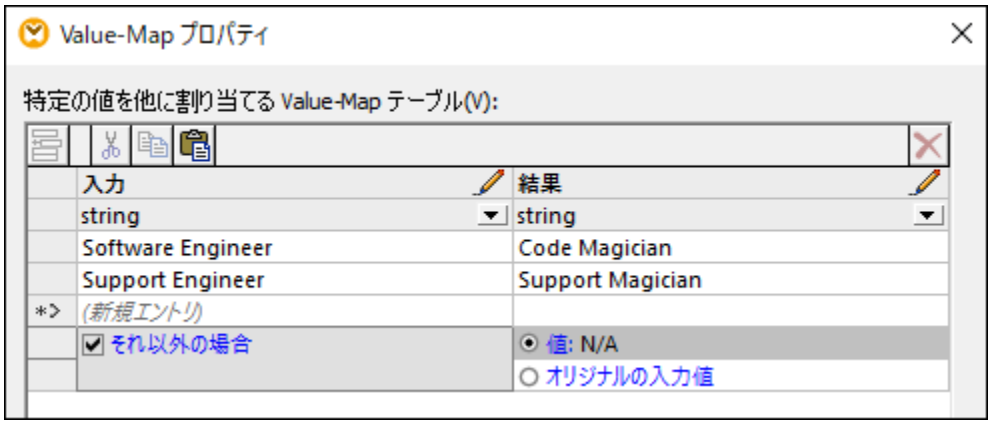
上記の通り、空のノードは生成される出力内で不足するエントリを引き起こします。このため、上記のXML フラグメント内では、ルックアップテーブル内で職位が存在するためMichelle Butler の職位のみが置き換えられています。ここまでに作成されている構成はオリジナルの必要条件を満たしていません。正しいセットアップは以下の通りです:



上記の構成を使用すると、マッピングのランタイムに以下が発生します:

- 「Software Engineer」は「Code Magician」と置き換えらえます。
- 「Support Engineer」は「Support Magician」と置き換えらえます。
- ルックアップテーブルでオリジナルのタイトルが見つからない場合、Value-Map は変換しないままタイトルを返します。

説明のために、「Software Engineer」と「Support Engineer」以外の職位を「N/A」などのカスタム値に変えることができます。これを達成するには、Value-Map プロパティを以下に示される通り設定します:



マッピングをプレビューする場合、各職位が出力に存在しますが、一致しない項目には「N/A」値が与えられます。例:

```
<Person>
   <First>Vernon</First>
```

```
   <Last>Callaby</Last>
   <Title>N/A</Title>
</Person>
<Person>
   <First>Frank</First>
   <Last>Further</Last>
   <Title>N/A</Title>
</Person>
<Person>
   <First>Michelle</First>
   <Last>Butler</Last>
   <Title>Code Magician</Title>
</Person>
```

Value-Map サンプルは終了です。上記のロジックを適用することにより、他のマッピング内で希望する結果を得ることができます。

4.12　ノード名のマッピング

MapForce を使用して、マッピングを作成すると、多くの目的はソースから値を読み取り、ターゲットに値を書き込むことです。しかしながら、ソースからだけではなく、ノード名から値を取得することを希望する場合があります。例えば、ソース XML から(値ではなく) 要素または属性の名前を読み取り、ターゲット XML内で(名前では無く)要素または属性の値 に変換することを希望する場合があります。

以下の例を考慮してください： XML ファイルが製品のリストを含む場合、各製品は次のフォーマットを有します:

```
<product>
   <id>1</id>
   <color>red</color>
   <size>10</size>
</product>
```

目的は、各製品の情報を名前と値のペアに変換することです。例:

```
<product>
   <attribute name="id" value="1" />
   <attribute name="color" value="red" />
   <attribute name="size" value="10" />
</product>
```

このようなシナリオには、マッピングからのノード名へのアクセスが必要になります。このトピックの主題であるノード名への動的なアクセスにより、このようなデータ変換を行うことができます。

メモ　node-name と static-node-name コアライブラリ関数を使用して、上記の変換を行うことができます。しかしながら、この場合は、ソースから期待する正確な要素名が必要であり、ターゲットに手動で要素を接続する必要があります。また、これらの関数は、十分ではない場合があります。例えば、名前別にフィルター、またはグループ分けを行う場合、また、ノードマッピングからデータ型を操作する場合などが挙げられます。

ノード名への動的なアクセスは、ノード名を読み取る場合だけではなく、書き込む必要がある場合にも使用することができます。標準マッピングでは、基となるコンポーネントのスキーマからくる、ターゲット内の属性または要素の名前は、マッピングの実行前に既知です。動的なノード名を使用して、マッピングの実行前に既知ではない、新規の属性または要素を作成することができます。具体的には、属性または要素の名前が、MapForce によりサポートされるソースからマッピング自身により与えられます。

> ノードの子要素、または属性への動的なアクセスを可能にするには、ノードが実際に子要素または属性を持つ必要があり、XML ルートノードであってはなりません。

動的なノード名は、次のコンポーネント型に、または型からマッピングをする際にサポートされています:

- XML
- CSV/FLF*

* MapForce Professional または Enterprise Edition が必要です。

ノード名への動的なアクセスは次のマッピング言語によりサポートされています: BUILT-IN、XSLT 2.0、XSLT 3.0 、XQuery、C#、C++、Java。*

*これらの言語は MapForce Professional または Enterprise Edition を必要とします。

ノードへの動的なアクセスに関する情報は、以下を参照してください: ノード名へのアクセスを取得する。順序を追ったマッピングの例に関しては、例: 要素名を属性値にマップするを参照してください。

4.12.1 ノード名へのアクセスを取得する

XML コンポーネント 内のノードに子ノードが存在する場合、マッピングの名前と各子ノードの値をマッピングで直接取得することができます。このテクニックは、「動的なノード名」へのアクセスと呼ばれます。「動的」とは、処理がランタイム中に素早く行われることを意味し、マッピングが実行される前に既知の静的なスキーマ情報をベースに行われます。このトピックは、ノード名への動的なアクセスの有効化の方法と、その使用方法について説明します。

ソースからデータを読み取る場合、「動的なノード名」とは以下を行うことができることを意味します:

- ノードの全ての子ノード(または属性) のリストをシーケンスとして取得します。MapForce では、"シーケンス" は、ゼロのリスト、またはターゲットに接続することのできるアイテムであり、ソース内にアイテムが存在すると、ターゲットに同じ数のアイテムを作成します。ですから、例えば、ノードがソース内に5つの属性 を持つ場合、ターゲット内に属性に対応する5つの新規の要素を作成します。
- (標準マッピングが行う) 子ノードの値を読み取るだけではなく、名前も読み込みます。

ターゲットにデータを書き込む場合、「動的なノード名」は、以下を行うことができることを意味します:

- コンポーネント設定 (いわゆる「静的」名前)により与えられた名前とは対照的な、マッピング (いわゆる「動的」名前) により与えられる名前を使用して、新しいノードを作成します。

動的なノード名を説明するために、このトピックは次のXML スキーマを使用しています: **<マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial\Products.xsd**。このスキーマは、サンプルインスタンスドキュメント **Products.xml** を伴っています。スキーマとインスタンスファイルをマッピングエリアに追加するには、**「挿入」XML スキーマ/ファイル」** メニューコマンドを選択して、**<マイドキュメント>**¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥Products.xml を参照してください。ルート要素を選択するように問われると、`products` を選択します。

`product` ノードのための動的なノード名を有効化するには、右クリックし、次のコンテキストメニューコマンドの1つを選択します:

- ノードの属性にアクセスする場合、**「動的な名前を持つ属性を表示」**
- ノードの子要素にアクセスする場合、**「動的な名前を持つ子要素を表示」**

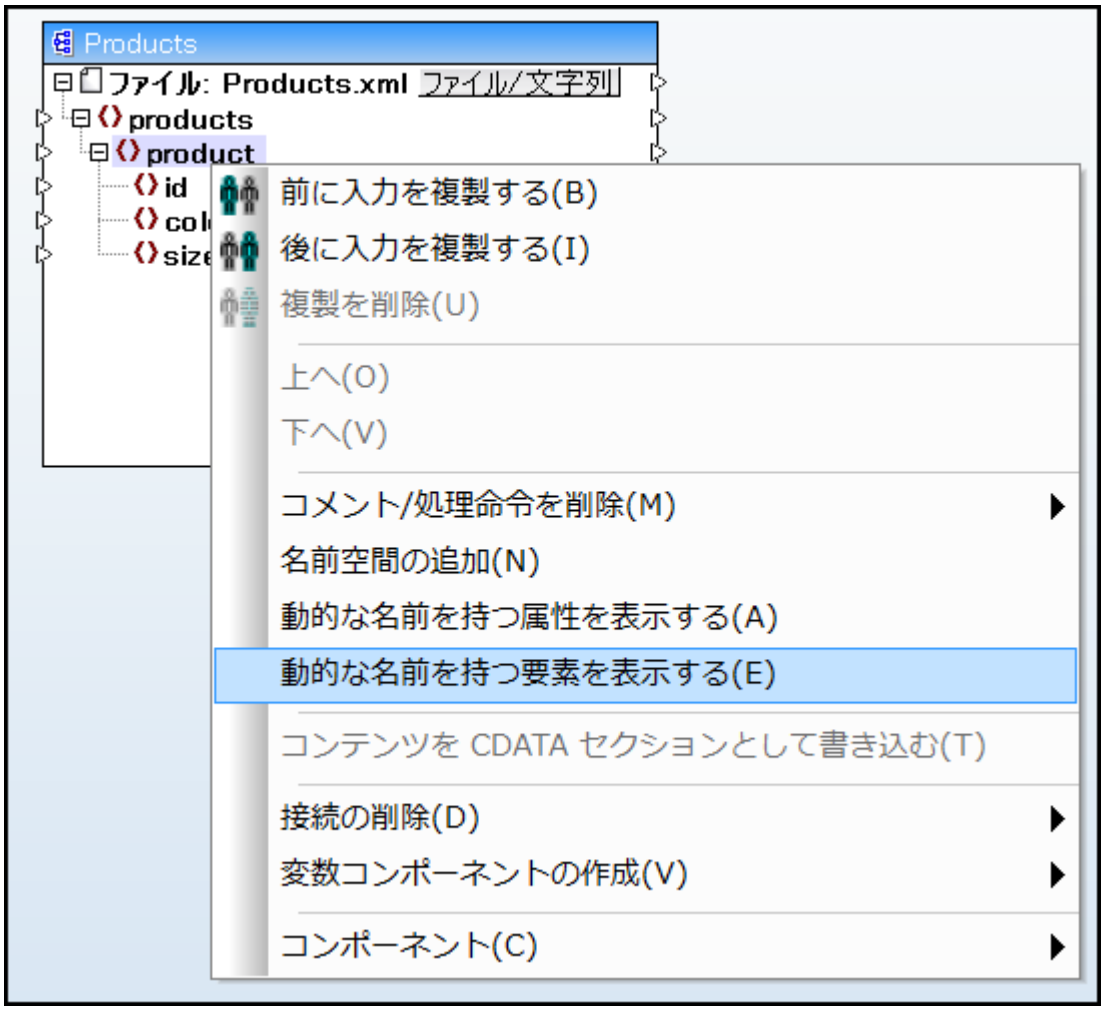


図. 1　(子要素のための) 動的なノード名の有効化

メモ　上のコマンドは、子ノードを持つノードのみに対して使用することができます。また、コマンドはアトミックノードには使用することができません。

ノードを動的なモードに切り替えると、下に表示されるダイアログボックスに類似したダイアログボックスが表示されます。このトピックの目的のために、オプションを下に表示されるように設定します。これらのオプションは特定の型のノードにアクセスする内で説明されています。

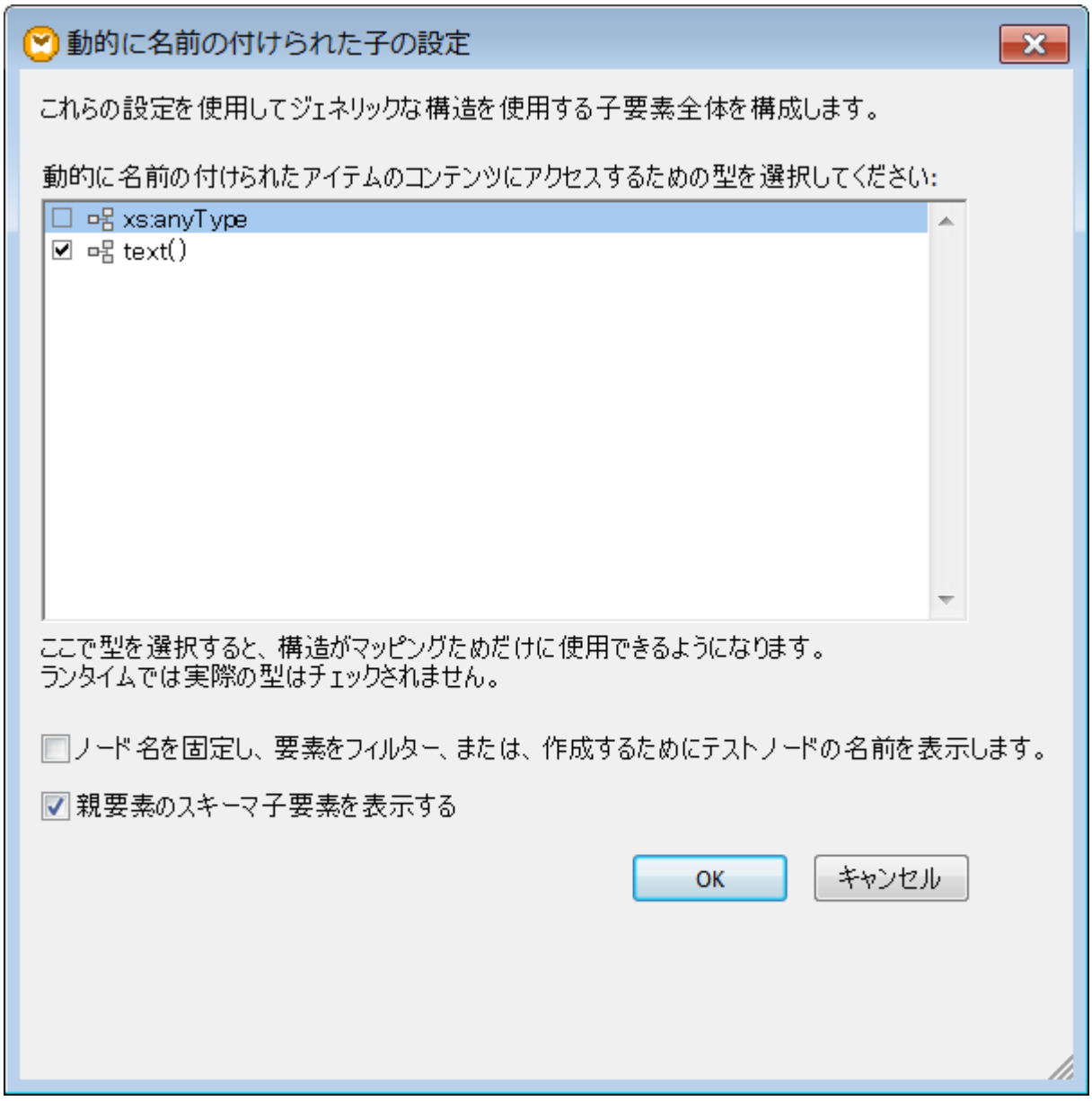


図 2　「動的に名前がつけられた子の設定」ダイアログボックス

図 3 は、`product` ノードのために動的なノード名が有効化されている場合、どのようにコンポーネントが表示されるかを示しています。コンポーネントの外観が大幅に変更されていることにご注意してください。

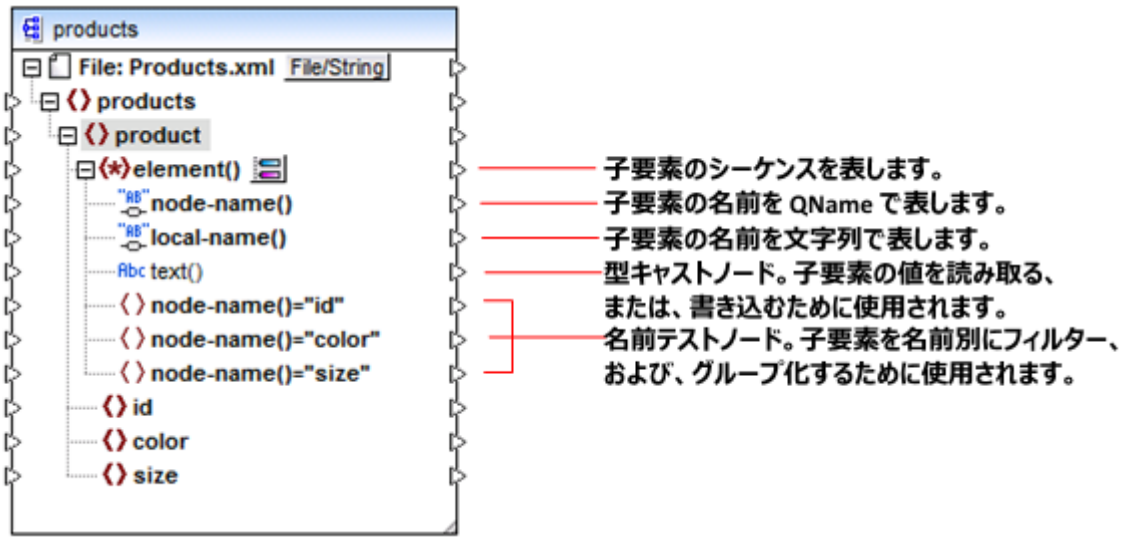


図3　（要素のために）動的なノード名が有効化されている場合

コンポーネントを標準モードに切り替えるには、`product` ノードを右クリックして、コンテキストメニューからオプション「動的な名前を持つ子要素を表示する」を無効化します。

下のイメージは、ノードの属性への動的なアクセスが有効化された時、同じコンポーネントがどのように表示されるかを示しています。`product` 要素を右クリックして、コンポーネントが取得され、コンテキストメニューから「動的な名前を持つ属性を表示する」を選択します。

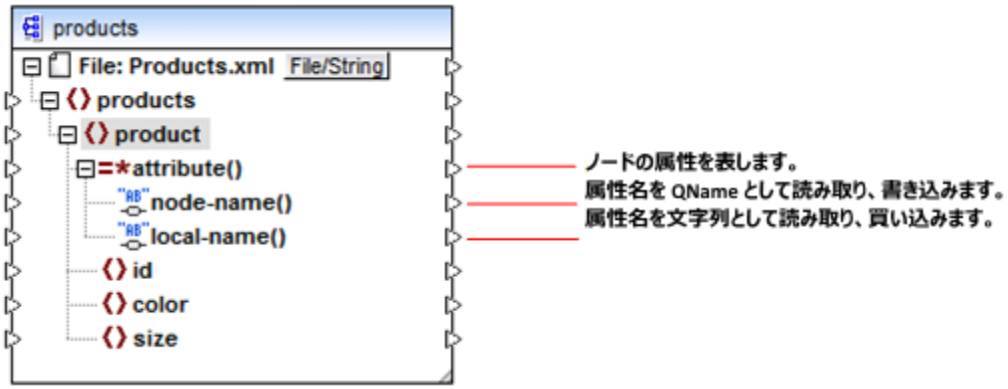


図 4　（属性のために）動的なノード名が有効化されている場合

コンポーネントを標準モードに切り替えるには、`product` ノードを右クリックして、コンテキストメニューから、「動的な名前を持つ属性を表示する」オプションを無効化します。

図 3 と図 4 に示されているように、コンポーネントの外観はノード（この場合は、`product`）が「動的なノード名」なモードに切り替えられると変更されます。新しい外観は、次のアクションを取る可能性を広げます:

- すべてのノードの子要素または属性のリストを読み取り、または書き込みます。これらは、`element()` または `attribute()` アイテムによりそれぞれ提供されます。
- 各子要素または属性の名前を読み取り、または書き込むます。名前は、`node-name()` と `local-name()` アイテムにより提供されます。

- 要素の場合、各子要素の値を特有のデータ型として読み取る、または書き込みます。この値は、型キャストノード（この場合は、`xs:string` アイテム）により与えられます。要素のみが型キャストノードを持つことができることに注意してください。属性は、常に、「文字列」型として処理されます。
- 名前別に子要素をグループ化します。

「動的なノード名」モード内で作業することのできるノード型は、下で説明されています。

element()

ターゲットコンポーネントと比較すると、このノードはソースコンポーネント内で異なる振る舞いをします。ソースコンポーネント内で、ノードの子要素をシーケンスとして提供します。図 3 `element()` は、全ての `product` の子要素のリスト（シーケンス）を与えています。例えば、次の XML から作成されたシーケンスは、3 つのアイテムを含みます（これは、`product` 3 つのつの子要素が存在するからです）:

```
<product>
   <id>1</id>
   <color>red</color>
   <size>10</size>
</product>
```

シーケンス内の各アイテムの実際の名前 と型は、`node-name()` ノード と型キャストノードによりそれぞれ与えられています 上記を理解するには、ソースからのデータ XML を ターゲット XML に次のように変換する必要があると仮定してください：

図 6 シーケンス内の各アイテムの実際の名前 と型は、`node-name()` ノード と型キャストノードによりそれぞれ与えられています 上記を理解するには、ソースからのデータ XML を ターゲット XML に次のように変換する必要があると仮定してください：

マッピングの目的を達成するマッピングは以下のようになりますマッピング:

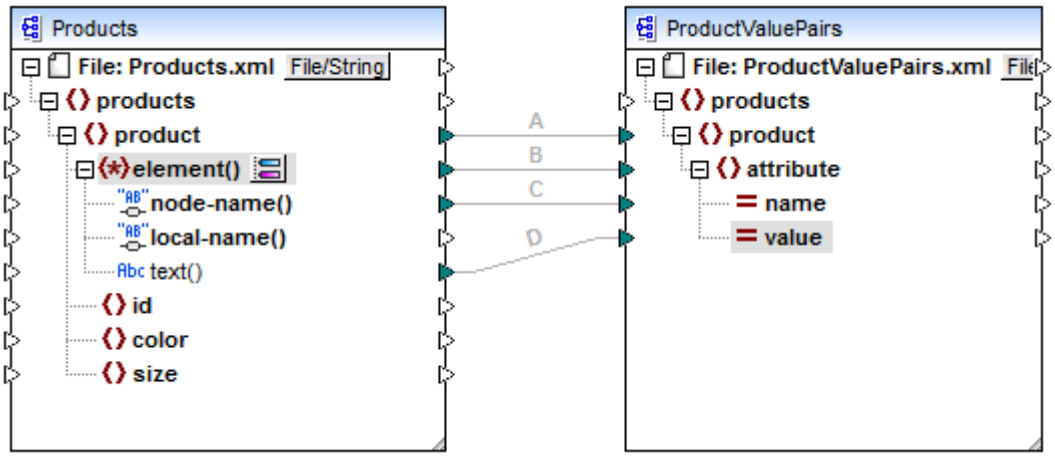


図 7　(MapForce 内で) XML 要素名 を属性値 にマッピングする

node-name() と text() は、シーケンス内の各アイテムの値と実際の名前を与えますが、element() のここでの役割は、product の子要素のシーケンスを提供します。このマッピングは、チュートリアルのサンプルが付随しており、以下で更に詳しく説明されています: 例: 要素名を属性値にマップする。

ターゲットコンポーネント内では、element() は、ソース内の各アイテムのマッピングの基本ルールの例外であるとおり自身では何も作成しませんが、1 つのターゲットアイテムを作成します。実際の要素 は、(node-name() の値を使用して) 型キャストノード と自身の名前を使用して) 名前テストノードにより作成されます。

attribute()

図 4 内で表示されているように、このアイテムは、マッピングのランタイムでノードの全ての属性へのアクセスを有効化します。ソースコンポーネント内で、接続されたソースノードの属性をシーケンスとして提供します。例えば、次の XML 内で、シーケンスは、(product には 2 つの属性が存在するため) 2 つのアイテムを含みます:

```
<product id="1" color="red" />
```

attribute() ノードは、シーケンス内の各属性の値のみを文字列の型として与えます。各属性の名前は node-name() ノードにより与えられます。

ターゲットコンポーネント内では、このノードは、接続されたシーケンスを処理し、シーケンス内の各アイテムのために属性の値を作成します。属性の名前は、node-name() により与えられます。例えば、ソースからのデータ XML を ターゲット XML を以下のように処理すると想定します:

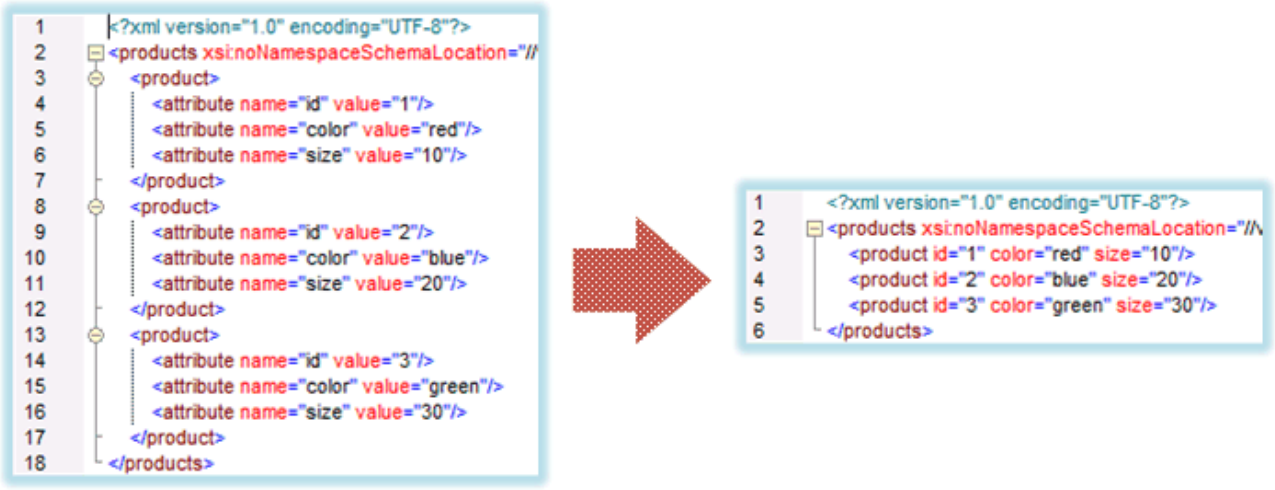


図 8 属性の値を属性名にマッピングする(必須)

この目的を達成するマッピングは以下のようになります:

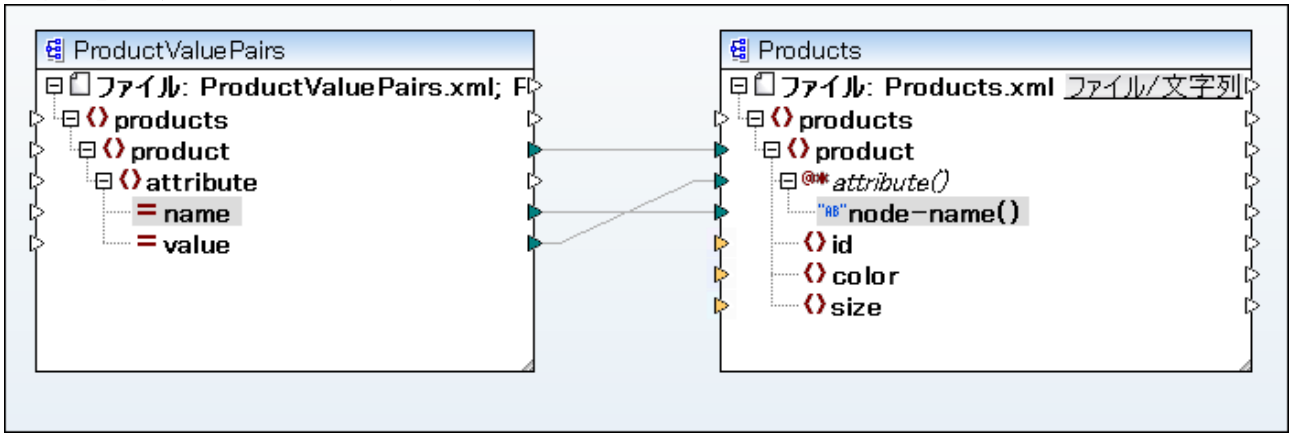


図 9 (MapForce 内で)属性の値を属性名にマッピングする

メモ この変換は、ノードの属性へのアクセスを有効化せずに行うことができます。ここでは、どのように`attribute()`がターゲットコンポーネント内で動作するかを表示しています。

このマッピングを再作成する場合は、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダー内の**ConvertProducts.mfd** マッピングと同じ XML コンポーネントを使用しています。マッピングとして使用します。唯一の違いは、ターゲットがソースになり、ソースがターゲットになることのみです。ソースコンポーネントのための入力データには、属性の値を実際に含む XML インスタンスが必要になります。例:

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
   <product>
      <attribute name="id" value="1"/>
      <attribute name="color" value="red"/>
      <attribute name="size" value="big"/>
   </product>
</products>
```

上記のコードリスティングでは、名前空間とスキーマ宣言が、ここでは簡単にするために省略されていることにご注意してください。

node-name()

ソースコンポーネント内では、node-name() は、element() の子要素の名前、または attribute() の要素の名前をそれぞれ提供します。デフォルトでは、提供される名前は、xs:QName の型です。名前を文字列として取得するには、 local-name() ノードを（図 3 を参照してください）、を使用します。

ターゲットコンポーネント内では node-name() は、element() または attribute() 内に含まれる各要素または属性の名前を書き込みます。

local-name()

このノードは、node-name() と同様の作動をしますが、異なる点は、xs:QName の代わりに xs:string が使用されていることです。

型キャストノード

ソースコンポーネント内では、型キャストノードは、element() 内の各子要素の値を提供しますこのノードの名前と構造は、「動的に名前がつけられた子の設定」ダイアログボックス（図 2）から選択された型により異なります。

ノードの型を変更する場合は、「**選択の変更**」( ) ボタンを使用して、スキーマワイルドカード（xs:any）を含む、使用できる型から希望する型を選択します。詳細に関しては、次を参照してください： 特定の型のノードにアクセスする。

ターゲットコンポーネント内で、型キャストノードは、element() 内に含まれる各子要素の値特有のデータ型として書き込みます。希望するデータ型は「**選択の変更**」( ) ボタンをクリックすることにより選択することができます

名前テストノード

ソースコンポーネント内では、名前テストノードは、ソースインスタンスから名前別に子要素をグループ化、またはフィルターする方法が提供されています。正確な型を使用してマッピングが、インスタンスデータにアクセスしていることを保証するために子要素を名前別にフィルターする必要がある場合があります。（次を参照してください： 特定の型のノードへのアクセス）。

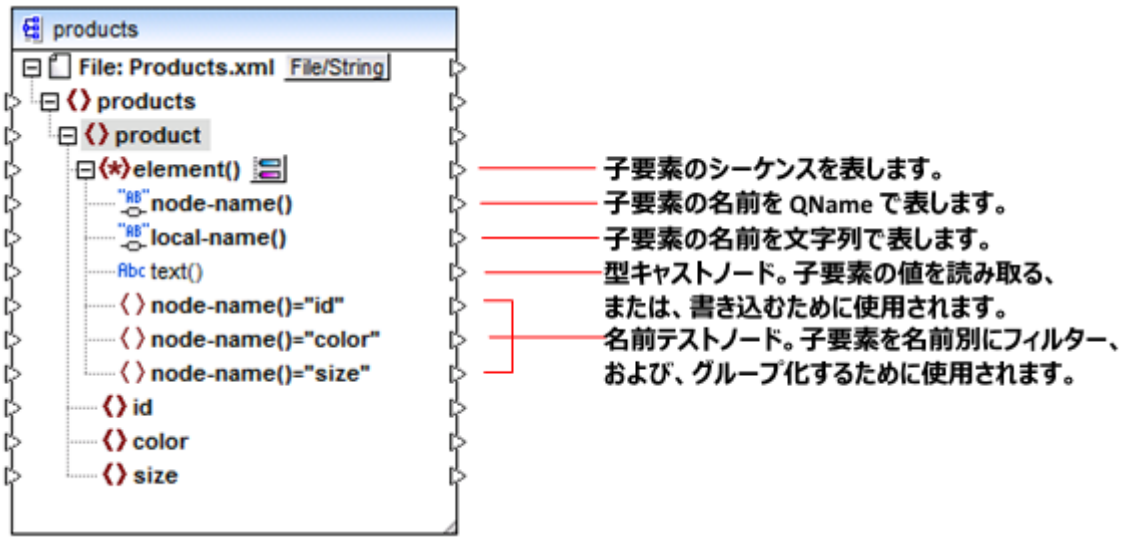
一般的には、名前テストノードは、値とサブ構造の読み取りと書き込みをする通常の要素ノードとほぼ同様の作動をします。しかしながら、動的なアクセスが有効化されていると、マッピングのセマンティクスが異なるため、制限が発生します。例えば、2 つの名前テストノードを連結することはできません。

ターゲット側では、名前テストノードは、接続されているソースシーケンスにアイテムが存在するため、出力内と同じ数量の要素を作成します。これらの名前は、node-name() にマップされている値を上書きします。

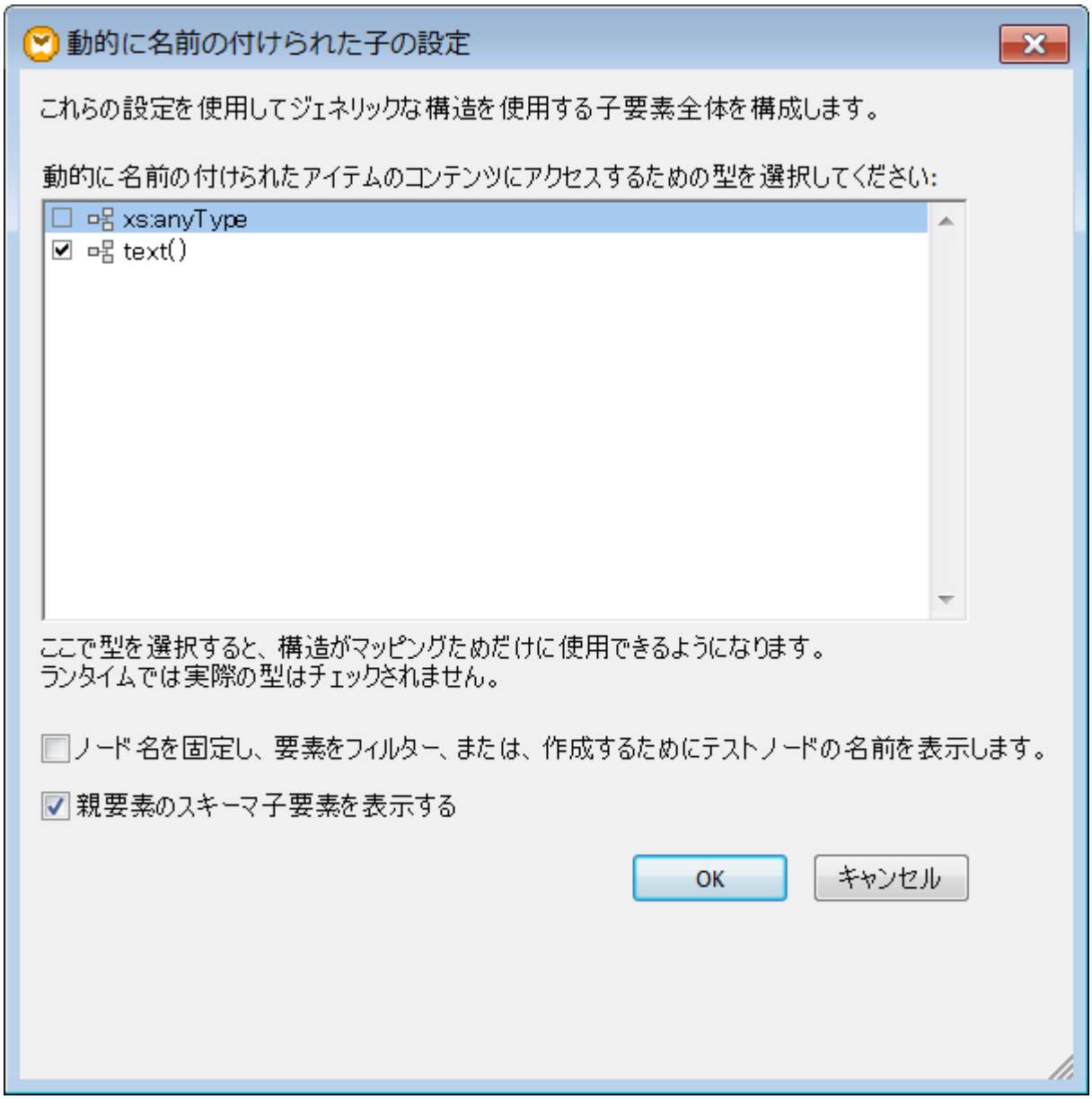
必要であれば、コンポーネントから名前テストノードを非表示にすることができます。これを行うには、 element() ノードの横の「**選択の変更**」( ) ボタン をクリックします。そして、「動的な名前を持つ子の設定」ダイアログボックスから、「**名前テストノードの表示**」チェックボックスのチェックを解除します。

4.12.2　特定の型のノードへのアクセス

前のセクションで説明されているとおり、ノード名へのアクセスの取得 は、ノードを右クリックして「**動的な名前を持つ子要素を表示する**」コンテキストメニューコマンドを選択し、ノードの全ての子要素にアクセスすることができます。特定の型のノードにより値をアクセスすることはできますが、マッピングのランタイムでは、 node-name() ノードから各子要素の名前にアクセスすることができます。下のイメージでは、 型キャストノードは、text() ノードの横にあります。



子要素のデータ型は、マッピングのランタイム前にはわかりません。 また、各子要素により異なる可能性があります。例えば、XML インスタンスファイル内の `product` ノード は、型 `xs:integer` の子要素 `id` と型 `xs:string` の子要素 `size` を持つ可能性があります。特定の型のノードコンテンツにアクセスするには、下に表示されるダイアログボックスがノードの子要素への動的なアクセスを有効化するたびに開かれます。`element()` ノードの横の「**選択の変更**」( ) ボタンをクリックすることにより、このダイアログボックスを後で開くことができます。



「動的に名前がつけられた子の設定」ダイアログボックス

マッピングのランタイムに各子要素のコンテンツにアクセスするには、いくつかのオプションがあります:

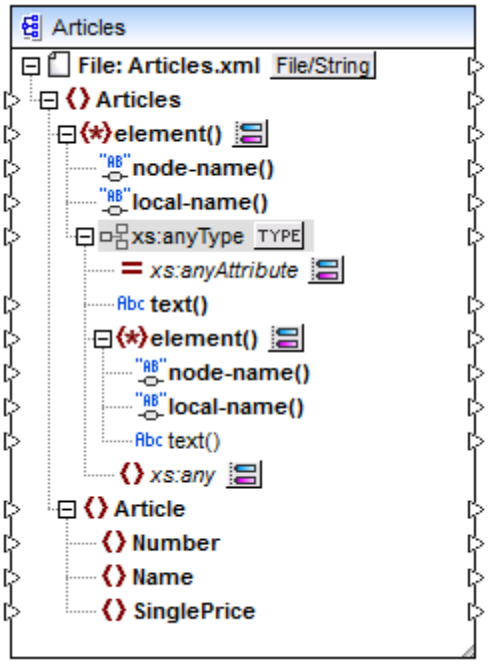
1. コンテンツを文字列としてアクセスするには、上のダイアログボックスで **text()** チェックボックスを選択します。この場合、ダイアログボックスが閉じられると、`text()` ノードがコンポーネント上に作成されます。このオプションは、コンテンツが `xs:int`、`xs:string` などの単純型 の場合、適切です。例: 要素名を属性値にマップするで詳しく説明されています。**text()** ノードは、現在のノードの子ノードがテキストを含むことができる場合表示されることに注意してください。
2. スキーマに許可されている特定の複合型としてコンテンツにアクセスすることができます。カスタム複合型が、選択されているのためにノードグローバルに許可されていると、上のダイアログボックスで使用することが可能になり、横のチェックボックスを選択することができます。上のイメージでは、グローバルに定義されている複合型は存在せず、この選択は使用することができません。
3. コンテンツを型としてアクセスする。これは、高度のマッピングのシナリオで役に立ちます（次を参照してください「更に深い構成にアクセスする」を参照してください）。これを行うには、**xs:anyType** の横のチェックボックスを選択してください。

マッピング ランタイムでは、MapForce（型 キャスト ノードを介して）には、実際のインスタンス ノード型に関する情報がないことにご注意してください。ですから、マッピングは、正確な型を使用してノード コンテンツにアクセスする必要があります。例えば、ソース XML インスタンスが多種の複合型の子ノードを持つことを希望する場合、以下を行います:
a）型キャスト ノードを一致させる必要が有る複合型に設定します（上のリストの2番目のアイテムを参照してください）。
b）一致する必要のあるインスタンスのみから読み込むためにフィルターを追加します。詳細に関しては以下を参照してください: フィルターと条件。

更に深い構成にアクセスする

スキーマ ワイルドカードの選択により、ノードをノードの直下の子よりもスキーマ内の更に深いレベルでアクセスすることができます。高度なマッピングのシナリオではとても役に立ちます。マッピングは、XML ノードの直下の子しかアクセスすることができないため、例: 要素名を属性値にマップする 等の簡単なマッピングでは、このテクニックを必要としません。しかしながら、動的に更に深い構造にアクセスする必要がある場合、例えば「孫」にアクセスが必要な場合、以下の方法でアクセスすることが可能です。

1. 新規マッピングを作成する。
2. 挿入メニューから「XML スキーマ/ファイルを挿入する」をクリックし、XML インスタンス ファイルを参照します。（子の例では、＜マイ ドキュメント＞¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥ フォルダーからの **Articles.xml** ファイルです）。
3. `Articles` ノードを右クリックして、「**動的な名前を持つ子要素を表示する**」 コンテキスト コマンドを選択します。
4. 「動的に名前がつけられた子の設定」ダイアログ ボックスから **xs:anyType** を選択します。
5. `xs:anyType` ノードを右クリックして、「**動的な名前を持つ子要素を表示する**」コンテキスト コマンドに対して選択します。
6. 「動的に名前がつけられた子の設定」ダイアログ ボックスから **text()** を選択します。



上記のコンポーネントで、2つの `element()` ノードが存在することにご注目してください。2番目の `element()` ノードは、**Articles.xml** インスタンス内の `<Articles>` ノードの孫に動的なアクセスを与えます。

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

```
   <Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Number>2</Number>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
   <Article>
      <Number>3</Number>
      <Name>Pants</Name>
      <SinglePrice>34</SinglePrice>
   </Article>
   <Article>
      <Number>4</Number>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

Articles.xml

例えば、「孫」要素名（Number、Name、SinglePrice）を取得するには、2番目のelement()ノードの下のlocal-name()ノードからターゲットノードに接続を描きます。同様に、「孫」要素の値（1、T-Shirt、25）を取得するには、text()ノードから接続を描きます。

この例には適用することはできませんが、実際のシナリオでは、更に深いレベルにアクセスするために、動的なノード名を後のxs:anyTypeノードに対して有効化することができます。

以下の点に注意してください：

- TYPE ボタンを使用して、生成された型を現在のスキーマから選択し、異なるノード内に表示することができます。この方法は、生成されたスキーマの型から、またはスキーマの型へマップする必要がある場合のみに役に立ちます（次を参照してください：生成されたXML スキーマ型）。
- element()ノードの横の「選択の変更」（ ）ボタンは、このトピックで説明されている「動的に名前の付けられた子の設定」ダイアログボックスを開きます。
- xs:anyAttribute 属性の横の「選択の変更」（ ）ボタンにより、スキーマ内でグローバルに定義された属性を選択することができます。同様に、 xs:any 要素の横の「選択の変更」（ ）によりスキーマ内でグローバルに定義されている要素を選択することができます。これにより、スキーマワイルドカードへ、またはスキーマワイルドカードからのマッピングと同じように動作します（次も参照してください：ワイルドカード – xs:any / xs:anyAttribute）。このオプションを使用する場合、選択された属性または要素がスキーマに従い特定のレベルで存在できることを確認してください。

4.12.3　例: 要素名を属性値にマップする

この例は、XML ドキュメントから、要素名をターゲット XML ドキュメント内の属性の値にマップする方法について説明しています。この例には、付随するサンプルマッピングがあり、次のパスを使用して検索することができます：**<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\ConvertProducts.mfd**。

例を理解するために、XML ファイルは製品のリストを含むと想定しましょう。各製品は次のフォーマットを有します：

```
   <product>
```

```
    <id>1</id>
    <color>red</color>
    <size>10</size>
  </product>
```

目的は、各製品の情報を名前と値ペアに変換することです。例:

```
  <product>
    <attribute name="id" value="1" />
    <attribute name="color" value="red" />
    <attribute name="size" value="10" />
  </product>
```

上記のようなデータマッピングを行うには、この例でも使用されていますが、"ノード名への動的なアクセス"と呼ばれる MapForce 機能を使用しています。マッピングを実行する際、「動的」とは、(値だけではなく)ノード名も読み取ることができ、これらの名前を値として使用することを意味します。必要とするマッピングをいくつかのシンプルなステップを以下に示されるように作成することができます。

ステップ 1: ソース XML コンポーネントをマッピングに追加する

- 「挿入」メニューから、「XML スキーマ/ファイル」をクリックして、<マイドキュメント>**\Altova\MapForce2021\MapForceExamples\Tutorial\Products.xml** を参照します。この XML ファイルは、同じフォルダー内にある **Products.xsd** スキーマを指します。

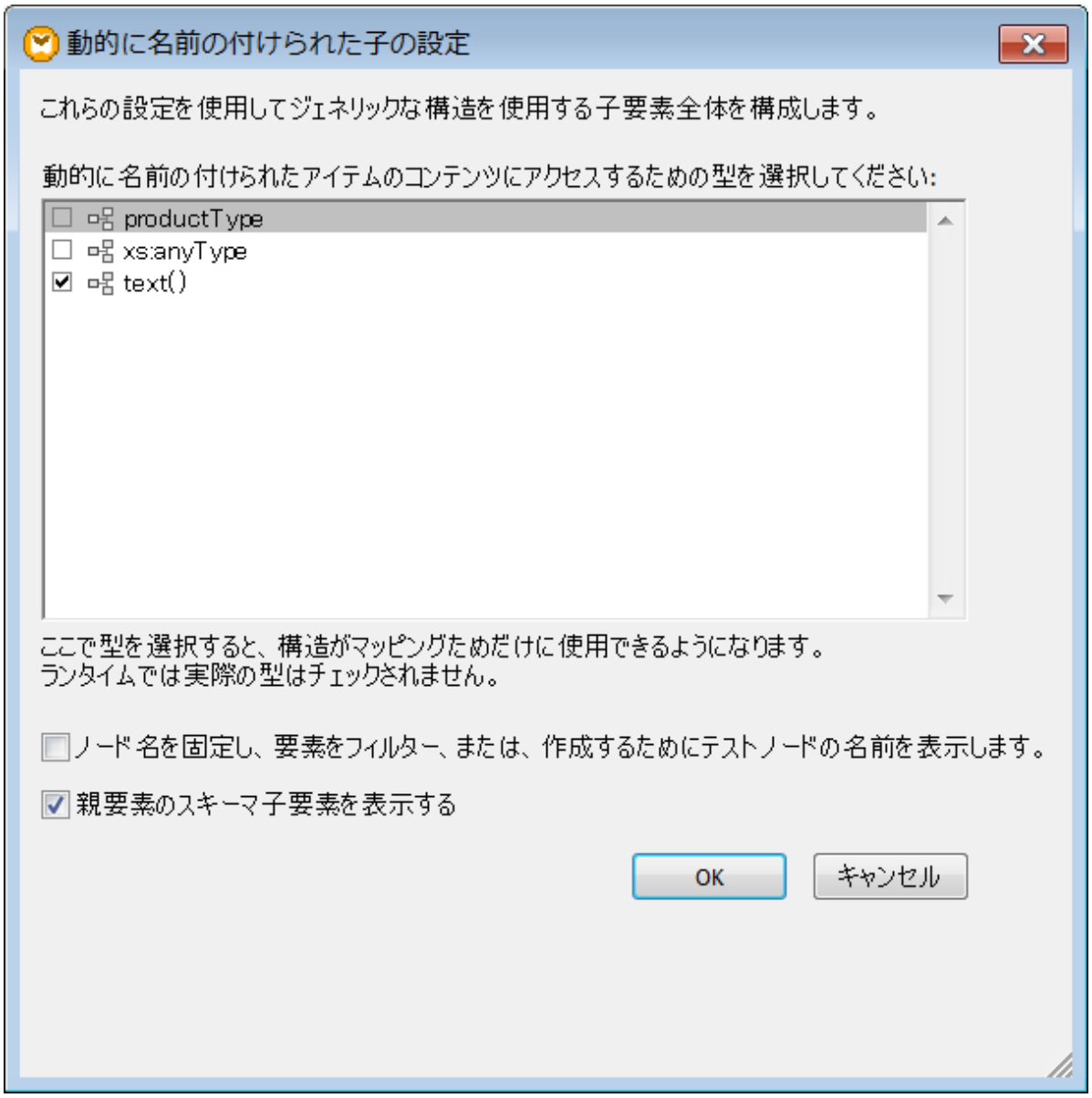
ステップ 2: ターゲット XML コンポーネントをマッピングに追加する

- 「挿入」メニューから、「XML スキーマ/ファイル」をクリックして、次のファイルを参照します: <マイドキュメント>**\Altova\MapForce2021\MapForceExamples\Tutorial\ProductValuePairs.xsd**。*インスタンスファイルを与えるように問われると、「スキップ」をクリックします。*ルート要素を選択するようにプロンプトされると、ルート要素として、`products` を選択します。

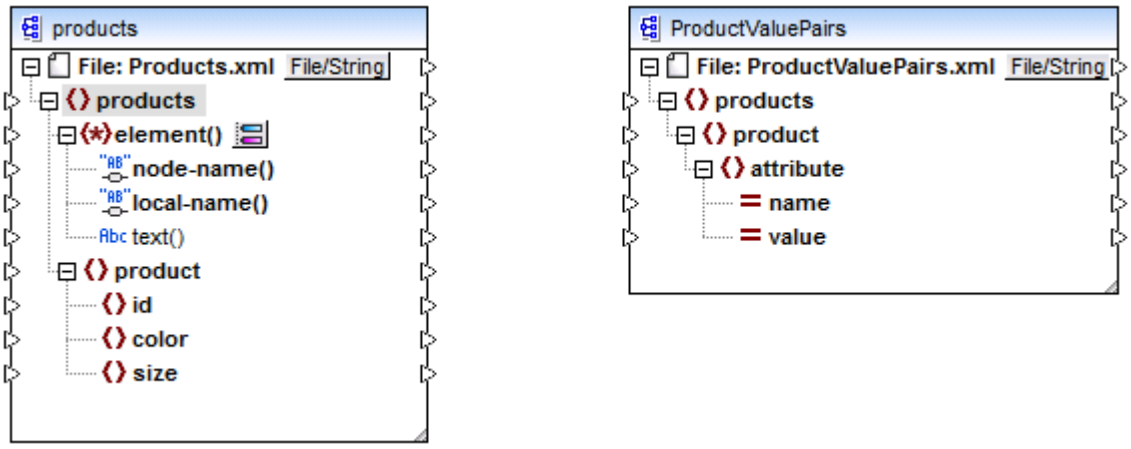
この時点では、マッピングは、以下のようになります:

ステップ 3: 子ノードへの動的なアクセスを有効化する

1. `products` ノードを右クリックして、コンテキストメニューから、「**動的な名前を持つ子要素を表示する**」を選択します。
2. 開かれるダイアログボックス内から、**text()** を型として選択します。他のオプションをそのままにします。

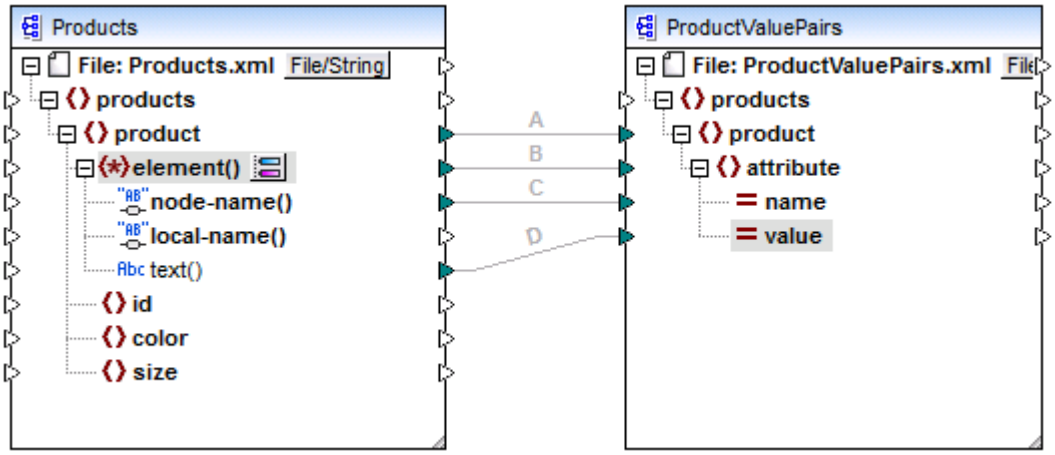


`text()` ノードがソースコンポーネントに追加されていることにご注意してください。このノードは、それぞれの子アイテムをマッピングに与えます（この場合は、「id」、「color」、「size」の値です）。



ステップ 4: マッピング 接続を描く

マッピング 接続 A、B、C、D 下に表示されるとおり、描きます。任意で、各接続上からをダブルクリックし、テキスト「A」、「B」「C」、および「D」をそれぞれ、詳細ボックスに入力します。



ConvertProducts.mfd

上で説明されているマッピングでは、接続 A は、ソース内の各製品のために、ターゲット内に製品を作成します。これまでは、これはノード名を指さない標準的な MapForce 接続でした。しかしながら、接続 B は、`product` の各対応する子要素のために、新規の要素を `attribute` と呼ばれるターゲット内に作成します。

> 接続 B は、マッピング内でとても重要です。この接続の目的である `product` の子要素のシーケンスを、ソースからターゲットに運びます。実際の名前または値を運ばないため、以下として理解される必要があります: ソース **element()** に N 子要素が存在する場合、ターゲット内のアイテムの N インスタンスを作成します。この特定の場合は、ソース内の `product` には、3 つの子要素 (`id`、`color` と `size`) が存在します。これは、ターゲット内の各 `product` が `attribute` という名前の 3 つの子要素を持つことを意味します。
>
> この例で説明されていませんが、同じルールが、**attribute()** の子要素をマップするために使用されます。ソース **attribute()** アイテムが、N 子属性を持つ場合、ターゲット内のそのアイテムの N インスタンスを作成します。

次に、接続 C は、`product` の各子要素の実際の名前をターゲットにコピーします (文字通り、「id」、「color」、と「size」)。

最後に、接続 D は、製品の各子要素の値を文字列の型として、ターゲットにコピーします。

マッピングの出力をプレビューするには、「出力」タブをクリックして、生成された XML を確認してください。マッピングの意図とする目的である通り、出力は、データが名前と値のペアとして保管されている、複数の製品を含んでいます。

```
<?xml version="1.0" encoding="UTF-8"?>
<products xsi:noNamespaceSchemaLocation="ProductValuePairs.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <product>
      <attribute name="id" value="1"/>
      <attribute name="color" value="red"/>
      <attribute name="size" value="10"/>
   </product>
   <product>
      <attribute name="id" value="2"/>
      <attribute name="color" value="blue"/>
      <attribute name="size" value="20"/>
   </product>
   <product>
      <attribute name="id" value="3"/>
      <attribute name="color" value="green"/>
      <attribute name="size" value="30"/>
   </product>
</products>
```

生成されたマッピング出力

4.13 マッピングのルールと戦略

一般的には、MapForce はデータを直観的にマップしますが、出力に多すぎる、または少なすぎるアイテムが含まれているシチュエーションが存在します。このチャプターは性格ではない接続やマッピングコンテキストが理由でマッピングが希望しない出力を生成するシチュエーションを回避するための手助けを意図としています。

マッピングのルール

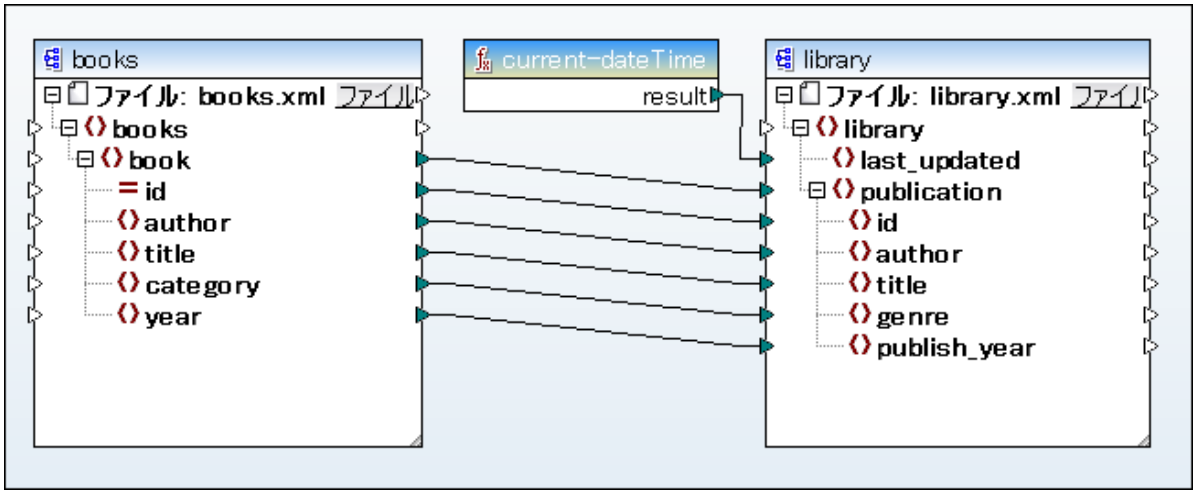
有効であるためには、マッピングには少なくとも1つのソースコンポーネントと少なくとも1つのターゲットコンポーネントが含まれている必要があります。ソースコンポーネントはデータを通常はファイルまたはデータベースから読み取ります。ターゲットコンポーネントは通常ファイルまたはデータベースでデータを書き込みます。上記が正しくない個所でマッピングを保存しようとすると、メッセージウィンドウにエラーが表示されます:「マッピングは少なくとも2つの接続された構造を必要とします。例えば、スキーマまたはデータベース構造を必要とします。」

データマッピングを作成するには、マッピング接続をソースとターゲットコンポーネント間に描きます。

描かれたマッピング接続全てはマッピングアルゴリズムを作成しています。マッピングランタイムでは、MapForce はアルゴリズムを評価し、それに基づきデータを処理します。マッピングアルゴリズムの統一性と効率性は接続により異なります。マッピング レベル、コンポーネント レベル、または 接続 レベルで設定の一部を調整することもできますが、基本的にはマッピング接続がデータの処理方法を決定します。

接続を作成する際に以下のルールが存在することに注意してください:

1. ソースアイテムから接続を描く場合、マッピングはソースファイルまたはデータベースからのアイテムと関連したデータを読み取ります。データは0、1、または複数の発生が存在する場合があります(すなわち、シーケンスである場合もあります。例えば、マッピングが書籍を含むXML ファイルからデータを読み込む場合、ソースXML ファイルは0、1、または複数の**book** 要素を含む可能性があります。下のマッピングでは、ソース(インスタンス)ファイルには複数の**book** 要素、または要素が含まれない場合がありますが、**book** アイテムはマッピングコンポーネント上で一度のみ表示されます。



2. ターゲットアイテムに接続を描く場合、マッピングはその種類のインスタンスデータを生成します。ソースアイテムが単純なコンテンツを含む場合(例えば、文字列または整数)およびターゲットアイテムが単純なコンテンツを受け入れる場合、MapForce はコンテンツをターゲットアイテムにコピーし、必要な場合は、データ型を変換します。受信するソースデータに従いゼロ、1、複数の値を生成することができます。次の点を参照してください。
3. ソース内の各(インスタンス)アイテムのために1つの(インスタンス)アイテムがターゲット内で作成されます。MapForce 内の一般的なマッピングルールです。上のマッピングを例として、ソースXML が3つの**book** 要素を含む場合、3つの**publication** 要素がターゲット側で作成されます。特別なケース シーケンス がいくつか存在することにも注意してください。を参照してください。
4. 各接続は*現在のマッピングコンテキスト* を作成します。コンテキストはどのデータが現在のターゲットノードのためにどのデータが使用できるかを決定します。コンテキストは、ですから、子のソースアイテムが実際にソースからターゲットコンポーネントにコピーされるかを決定します。接続を作成、または削除する場合、現在のコンテキストが不注意に変更される可能性があり、マッピングの出力に

影響を与える場合があります。例えば、同じマッピングが不必要にデータベース、または、Web サービスを呼び出す場合があります。このコンセプトは下のマッピングコンテキストで詳細に説明されています。

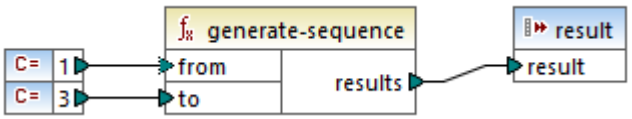
4.13.1 シーケンス

以前に記述されている通り一般的なマッピングルールは「ソース内の各アイテムのために、ターゲット内に 1 つ作成する」です。ここでは、「アイテム」は以下の 1 つを意味します:

- 入力ファイルまたはデータベースの単一のインスタンスノード
- 入力ファイルまたはデータベースのゼロから複数のインスタンスノードのシーケンス

マッピングの実行中、シーケンスがターゲットアイテムに達成すると、ソースノードの数と同じ数量のターゲットを生成するルールを作成します。いくつかの例外がこのルールには存在します:

- ターゲットアイテムが XML ルート要素の場合、1 度のみ作成されます。シーケンスを接続すると、結果はスキーマに対して有効でない場合があります。ルート要素の属性も接続されている場合 XML のシリアル化はマッピングのランタイムで失敗します。このため、ルート XML 要素にシーケンスを接続することを回避してください。
- ターゲットが 1 つの値のみ受け入れる場合、1 度のみ作成されます。アイテムのサンプルは 1 つの値にのみより受け入れられます: XML 属性、データベースフィールド、単純型出力コンポーネント。例えば、下のマッピングは `generate-sequence` 関数を使用して 3 つの整数 (1, 2, 3) のシーケンスを生成します。しかしながら、ターゲットは単一の値を受け入れる単純型出力コンポーネントのため出力は 1 個の整数を含みます。他の 2 つの値は無視されます。



- ソーススキーマが特定のアイテムが 1 度のみ発生するが、インスタンスファイルには多数のアイテムが存在する場合、MapForce は (スキーマに従うと必ず存在する必要がある) ソースから最初のアイテムを抽出し、ターゲット内で 1 個のアイテムを作成します。この振る舞いを無効化するには、チェックボックスをコンポーネント設定から解除します。この振る舞いを無効化するには、チェックボックス「min/maxOccurs をベースに入力処理最適化するを有効化する」コンポーネント設定内でクリアしてください。この振る舞いを無効化するには、コンポーネント設定からクリア、または、 XML コンポーネント設定 を参照してください。

シーケンスが空の場合、ターゲット側には何も生成されません。例えば、ターゲットが XML ドキュメントで、ソースシーケンスが空の場合、XML 要素はターゲット内に作成されません。

関数も同様な方法で作動します: 入力としてシーケンスを取得すると、シーケンス内のアイテムの数量だけ呼び出し (その数の出力を) 作成します。

> 関数が空のシーケンスを入力として取得すると、空の結果を返し、この結果出力は生成されません。

しかしながら、関数の一部のカテゴリでは、デザインのために入力として空のシーケンスを受けると空の値を返します。

- `exists`, `not-exists`, `substitute-missing`
- `is-null`, `is-not-null`, `substitute-null` (これらの 3 つの関数は前の 3 つとエイリアスです)
- 集計関数 (`sum`, `count`, など)
- (インラインではなく) 正規 関数であるユーザー定義関数

空の値を置き換えるには、add the `substitute-missing` 関数をマッピングに追加し、空の値を代替地と置き換えます。

関数には複数の入力が存在する場合があります。シーケンスが各入力に接続されている場合、通常希望される出力ではありませんが、全ての入力の連結された結果を出力することができます。これを回避するために、1つのシーケンスのみを複数のパラメーターを持つ関数に接続してください。他のすべてのパラメーターは親または他のコンポーネントからの「単純型」アイテムに接続されている必要があります。

4.13.2 マッピングコンテキスト

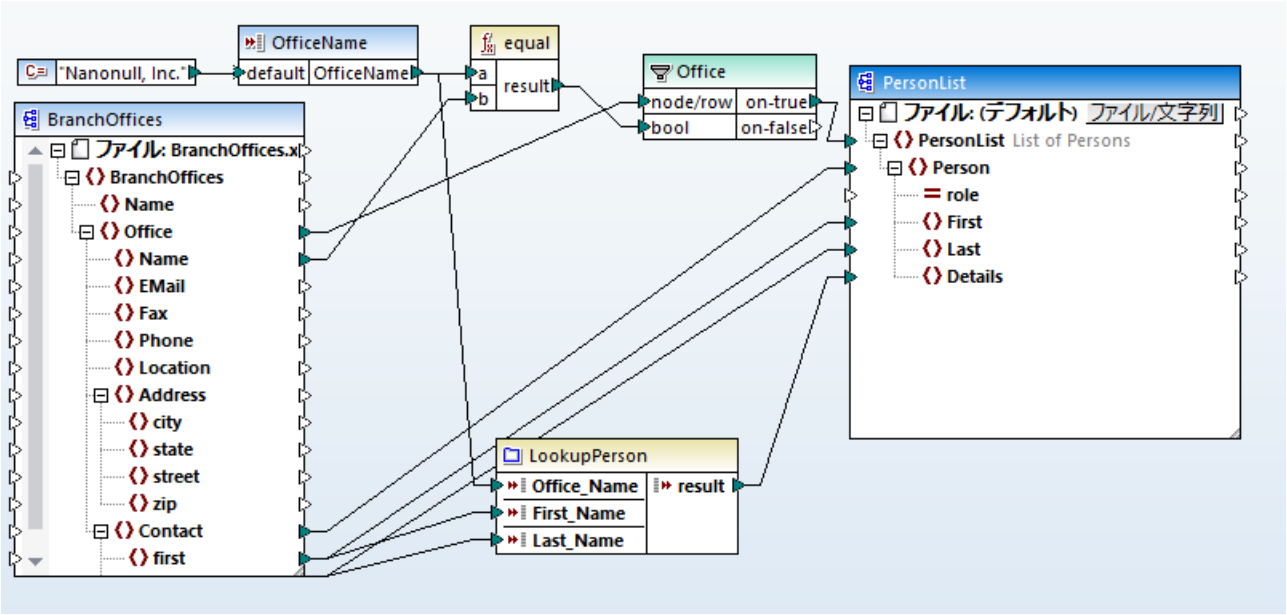
マッピングコンポーネントは深さの多くのレベルを含む階層的な構造です。他方、マッピングは複数のソースとコンポーネントおよび関数、フィルター、値マップなどの中間コンポーネントを持ちます。これは複数の関連性の内コンポーネントが接続されているとマッピングアルゴリズムに複雑性を追加します。マッピングをこのポジションで実行するには、現在のコンテキストがそれぞれの接続で確立される必要があります。

現在のコンテキストは処理済みの各接続と共に変更されるため、複数の「現在のコンテキスト」がマッピングの実行の期間に構築されていると言うことができます。

MapForce は *ターゲットルートアイテム（ノード）* 空開始する現在のコンテキストを常に構築します。これはマッピングの実行が実際に開始する箇所です。ターゲットルートアイテムへの接続は、関数または他の中間コンポーネントを含む直接または間接的に接続されている全てのソースアイテムにトレースバックされます。関数により生成された全てのソースアイテムと結果は現在のコンテキストに追加されます。

ターゲットノードの処理後、MapForce は階層的に作動します。具体的には、全てのターゲットコンポーネントのマップされたアイテムが階層的に処理されます。具体的には、上から下にターゲットコンポーネントが処理されます。新規の各アイテムのために、親コンテキストのすべてのアイテムを含む新規のコンテキストが作成されます。このために、ターゲットコンポーネント内のマップされたすべての兄弟アイテムはそれぞれに対して独立していますが、親アイテムのソースデータにアクセスすることができます。

上記が実際にどのように作動するかサンプルマッピング、**PersonListByBranchOffice.mfd** をベースに確認してみましょう。<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥ディレクトリ内でこのマッピングを見つけることができます。



上のマッピングでは、ソースとターゲットコンポーネントの両方がXML です。ソースXML ファイルには2つの**Office** 要素が含まれています。

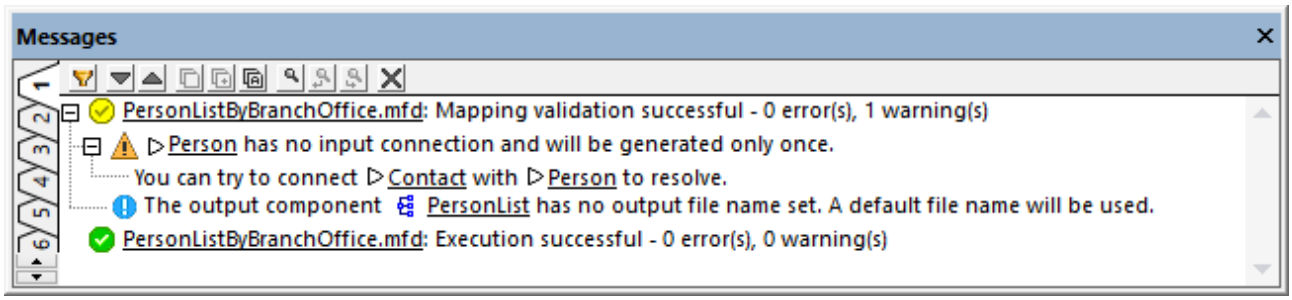
前述の通り、いつも通り、マッピングの実行はターゲットルートノード（このサンプルでは**PersonList** ）から開始されます。（フィルターと関数を介して）接続をソースアイテムにトレースバックすることで、ソースアイテムが**Office** であることが結論付けることができます。（他の接続パスは入力パラメーターへ接続されており、その目的に関しては以下で説明されています）。

Office と**PersonList** の間にわかりやすい接続が存在する場合、一般的なマッピングルールに従い、ソースファイル内の**Office** アイテムの数だけ、マッピングは**PersonList** インスタンスアイテムを作成します。しかしながら、フィルターが存在するためこれが発生しない場合があります。フィルターはフィルターの**bool** 入力に接続されているブール値の条件を満たすデータをターゲットコンポーネントに提供します。`equal` 関数は、オフィス名が「Nanonull, Inc.」に等しい場合 **true** を返します。ソースXML ファイル内のオフィス名が1つしか存在しないため、この条件は一度のみ満たされることができます。

結果的に**Office** と**PersonList** の間尾の接続はターゲットドキュメント全体のために単一のオフィスを定義します。これは**PersonList** アイテムのすべての子孫が現在のコンテキストの他のオフィスではなく "Nanonull, Inc." オフィスのデータにアクセス可能なことを意味します。

次は**Contact** と**Person** 間の接続です。一般的なマッピングルールに従うと、各ソース**Contact** のために1つのターゲット **Person** を作成します。各反復で、この接続は新規の現在のコンテキストを作成します。このため、子接続（**first** to **First**, **last** to **Last**）はデータをソースから各 **Person** のコンテキスト内でターゲットアイテムに提供します。

Contact と**Person** 間の接続をそのままにすると、マッピングにより複数の**First**、**Last** と**Details** ノードを持つ1つの**Person** が作成されます。このような場合、MapForce はメッセージウィンドウ内に警告と提案を表示します。例えば



最後に、マッピングにはユーザー定義関数 `LookupPerson` が含まれています。 **Contact** と**Person** 間の親接続のためユーザー定義関数はそれぞれの**Person** コンテスト内で実行されます。新規のPerson アイテムがターゲット側で作成される都度、関数が個人の**Details** 要素を作成するために呼び出されます。この関数は3つの入力パラメーターを取ります。最初のパラメーター（**OfficeName**）はマッピングの入力パラメーターからデータを読み取るように設定されています。マッピング出力を変更することなくこのパラメーターのための**Name** ソースデータはソースアイテムにより提供されることもできます。いずれにせよ、ソース値は親コンテキストからの値と同じです。内部では、ルックアップ関数は引数として取得された値を連結し単一の値を出力します。 詳細に関しては`LookupPerson` 関数のしくみと例: ルックアップと連結を参照してください。

4.13.2.1 ユーザー定義関数

ユーザー定義関数（UDF）はマッピング内に埋め込まれている入力、出力、および処理ロジックを定義するカスタム関数です。各ユーザー定義関数にはW eb サービスとデータベースを含むメインマッピングと同じコンポーネントが含まれている可能性があります。

> デフォルトではUDF がデータベースまたはW eb サービスコンポーネントを含む場合、そして、UDF への入力データが複数の値のシーケンスの場合、各入力値はUDF を呼び出し、データベースまたはW eb サービス呼び出しが結果として行われます。

これらの振る舞いは、代替法が無く *入力の値の数量の回数* UDF を呼び出す必要のあるマッピングの箇所で受け入れられます。

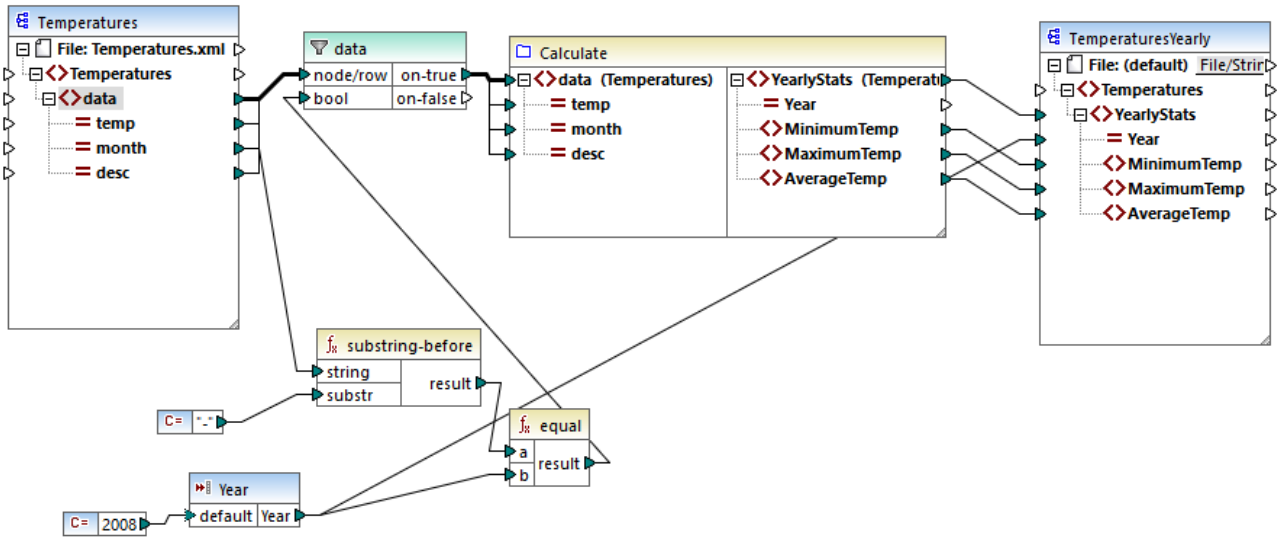
この発生を希望しない場合、UDF を構成して、入力として値のシーケンスを取得するように呼び出しを1回に制限するように構成することができます。通常これを返される前に値のセットを操作する（平均または総数を計算する）UDF のために行います。

同じ呼び出し内でUDF を複数の値を受け入れるように構成するにはUDF が型「インライン」ではなく「標準」の場合可能です。（詳細に関しては、ユーザー定義関数 チャプターを参照してください。）標準関数では、「**入力はシーケンスです**」チェックボックスを選択することによ

り入力パラメーターがシーケンスであることを指定することができます。入力パラメーターのタイトルバーをダブルクリックするとこのチェックボックスはコンポーネント設定で表示されます。このチェックボックスは以下に示される通り関数の呼び出しの頻度に影響を与えます:

- 入力データがシーケンスパラメーターに接続されていると、ユーザー定義関数は*一度のみ呼び出され*シーケンス全体がユーザー定義関数に渡されます。
- 入力データが非―シーケンスパラメーターに接続されていると、ユーザー定義関数は*シーケンス内の各単一アイテムのために*それぞれ呼び出されます。

サンプルとして以下のデモマッピングを開いてください: **<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥InputIsSequence.mfd**.

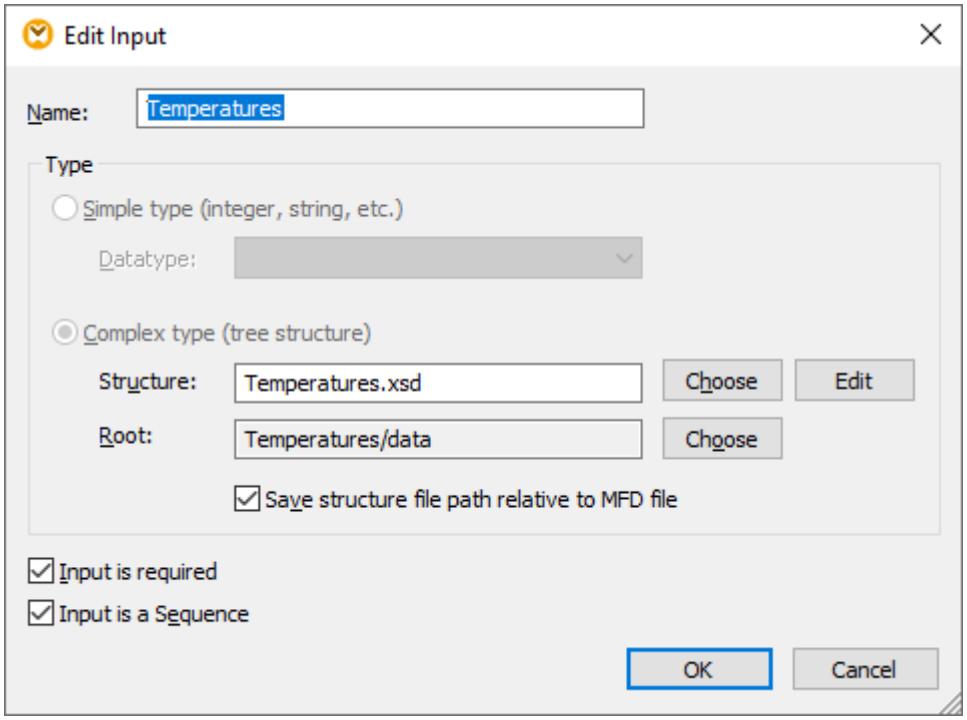


上のマッピングは値のセットを操作し、一度の呼び出し内で全ての入力値を必要とするUDF の典型的なケースを表しています。具体的には`Calculate` ユーザー定義関数は、XML ファイルから入力データを取り、最小、最大、および平均の気温をかえします。期待されるマッピングの出力は以下のとおりです:

```
<Temperatures>
    <YearlyStats Year="2008">
        <MinimumTemp>-0.5</MinimumTemp>
        <MaximumTemp>24</MaximumTemp>
        <AverageTemp>11.6</AverageTemp>
    </YearlyStats>
</Temperatures>
```

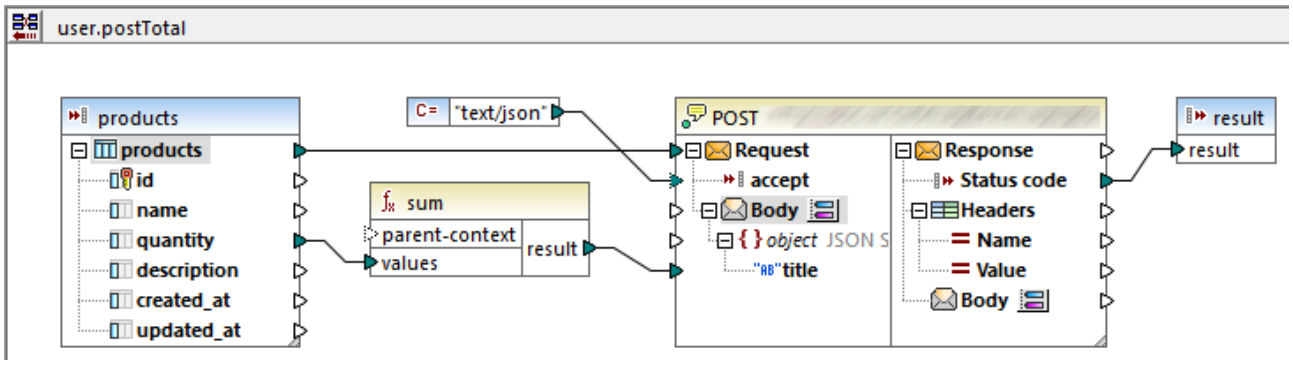
通常どおりターゲットコンポーネントのトップアイテム(このサンプルでは**YearlyStats**)からマッピングの実行は開始されます。このノードを作成するにはフィルターをトリガーするUDF からソースデータを取得しようと試みます。このマッピング内のフィルターの役割は2008 からの気温のみをUDF に渡すことです。

チェックボックス「**入力はシーケンスです**」がUDF の入力パラメーターのために選択されています(このチェックボックスを確認するには、`Calculate` 関数のタイトルバーをダブルクリックし関数のマッピングを入力し、入力パラメーターのタイトルバーをダブルクリックします)。上記の通り「**入力はシーケンスです**」オプションは関数に入力として提供された値のシーケンスを完了し、関数は一度のみ呼び出されます。



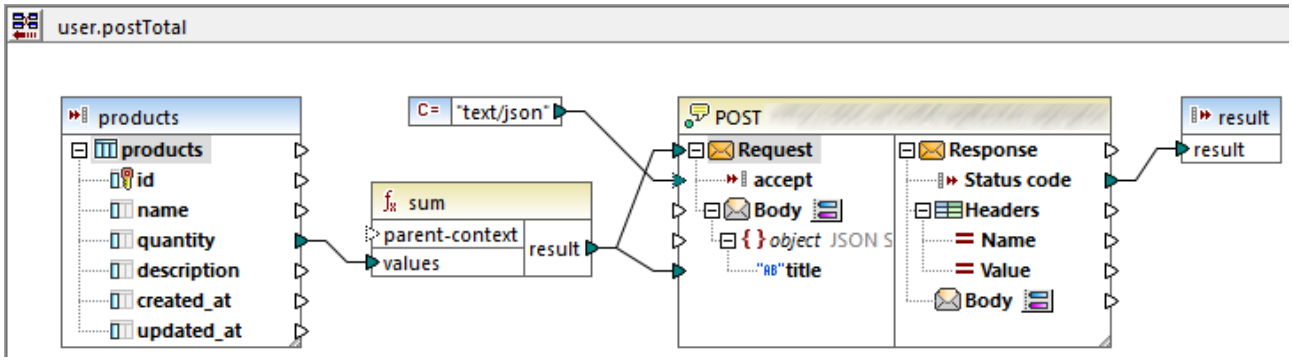
「入力はシーケンスです」チェックボックスが選択されていない場合、UDF はソース内の各値のために呼び出されます。この結果、最低、最高、および平均の値が各単一の値のために計算され、正確ではない出力が生成されます。

データベースまたは Web サービスの呼び出しを含む更に複雑な UDF に適用することで、実行を最適化し、データベースまたは Web サービスへの不必要な呼び出しを回避することができます。「入力はシーケンスです」チェックボックスは関数に入力 *後* に値のシーケンスに発生する事項を管理することはできません。すなわち、受信する値のシーケンスを Web サービスの入力に接続し複数回呼び出しが行われることを防ぐものはありません次の例を考慮してください：



上記の UDF は外部マッピングから値のシーケンスを受け取ります。具体的には、入力パラメーターに与えられたデータはデータベースから来るものです。入力パラメーターはオプション「入力がシーケンス」を選択しており、シーケンス全体が 1 つの呼び出し内で関数に提供されています。この関数は **quantity** 値を集計し、結果を Web サービスにポストします。1 つの Web サービスの呼び出しが期待されます。しかしマッピングが実行されると Web サービスは正確に複数回呼び出されます。これは Web サービスの **Request** 入力が単一の値ではなく *値のシーケンス* を受け取るからです。

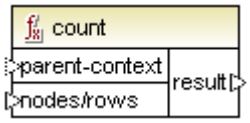
問題を解決するために Web サービスの **Request** 入力を `sum` 関数の結果に接続してください。Web サービスが一度のみ呼び出されるように、関数は 1 つの単一の値を処理します:



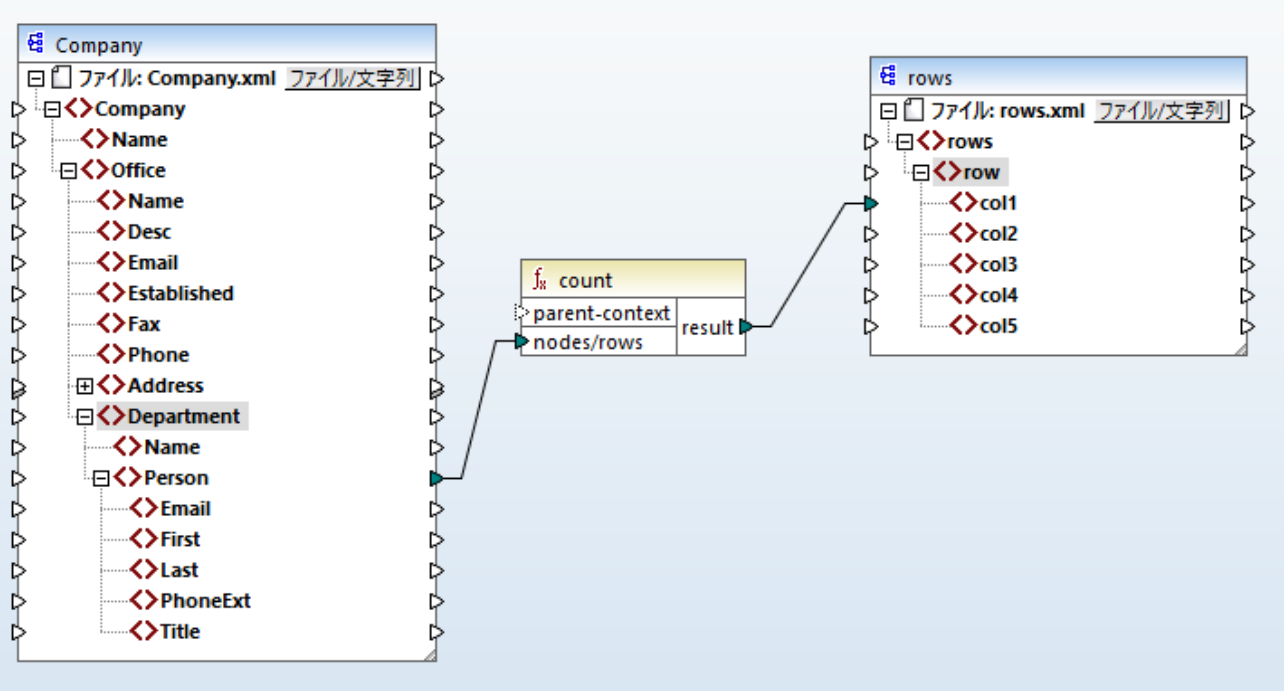
通常 sum、count などの集計関数は単一の値を生成します。しかしながら、許可する親接続が存在する場合、値のシーケンスを生成することができます。詳細に関しては以下を参照してください: 例: 親コンテキストの変更で詳細に説明されています。

4.13.2.2 例: 親コンテキストの変更

マッピングコンポーネントの一部は任意の**parent-context** アイテムを持っている場合があります。このアイテムを使用して、コンポーネントが作動し結果的にマッピングの出力を変更するようにマッピングコンテキストに影響を与えることができます。任意の**parent-context** を持つコンポーネントは以下の通りです: 集計関数、変数、ジョインコンポーネント。



親コンテキストを変更する方法に関しては次のマッピングを開いてください: <マイドキュメント>**\Altova\MapForce2021\MapForceExamples\Tutorial\ParentContext.mfd**。



上のマッピング内のソースXMLでは、2つのOfficeノードを含む単一のCompanyが存在します。各Officeノードは複数のDepartmentノードを含み、Departmentは複数のPersonノードを含んでいます。XMLエディターでXMLファイルを開くと、オフィス内の職員の配属は以下のようになっています:

オフィス	部署	職員数
Nanonull, Inc.	管理	3
	マーケティング	2
	エンジニアリング	6
	IT & 技術サポート	4
Nanonull Partners, Inc.	管理	2
	マーケティング	1
	IT & 技術サポート	3

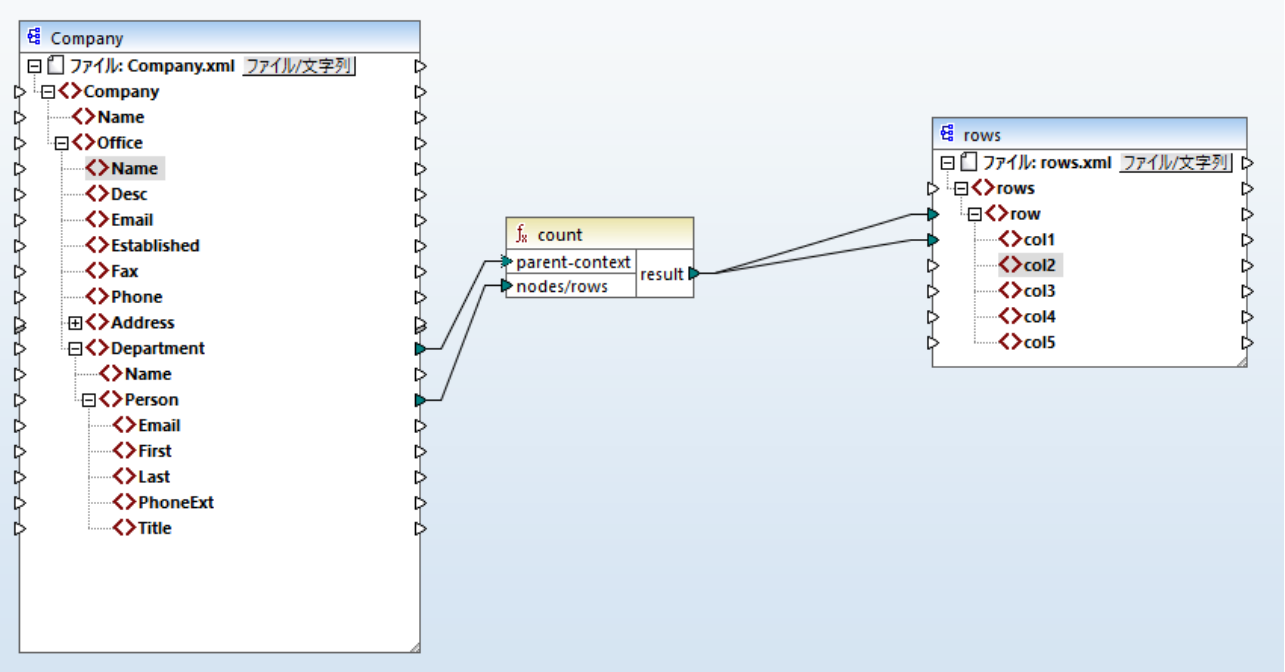
マッピングはすべての部署の職員数を計算します。この目的のために`core`ライブラリから`count`関数を使用します。マッピングをプレビューするために「出力」タブをクリックすると、ソースXMLファイル内の総職員数である単一の値**21**が生成されます。

マッピングの仕組みは以下のとおりです:

- 通常どおりターゲットコンポーネントのトップノード（このサンプルでは**rows**）からマッピングの実行は開始されます。**rows**への接続線は存在しません。この結果**Company**（ソースコンポーネントの上のアイテム）と**rows**（ターゲットコンポーネントの上のアイテム）間の明示的なマッピングコンテキストが確立されます。
- ソースファイルに1つの会社が存在するため関数の結果は単一の値です。

- **col1** ターゲットアイテムを作成するために、MapForce は、全ての部署のすべてのオフィスから **Person** ノードを数えるために、上記の *明示的な親コンテキスト* 内の `count` 関数を実行します。

関数の **parent-context** 引数によりマッピングコンテキストを変更することができます。これにより、例えば、各部署内の職員数を数えることができます。これを行うために、以下に示される通り2本の接続線を描きます:

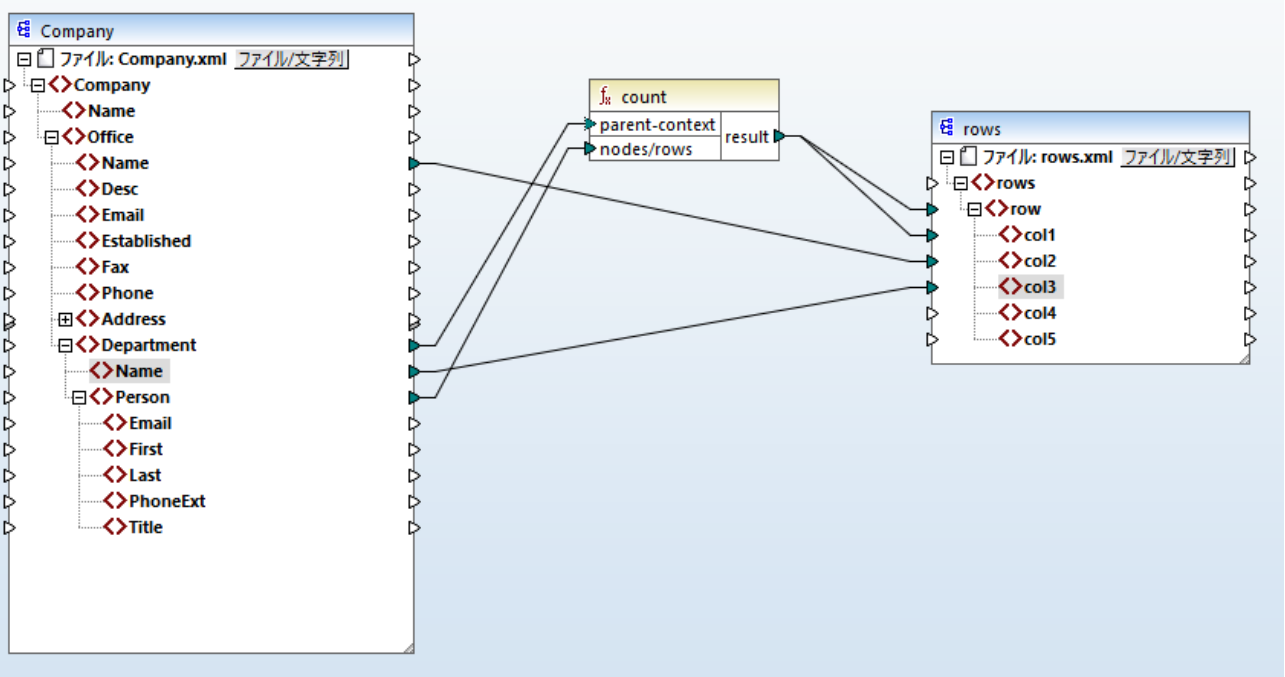


上のマッピングでは、接続線 A はthe `count` 関数の親コンテキストを **Department** に変更します。これにより、関数は各部署内の職員数を数えることができます。重要な点は、ソース内に複数の部署が存在するため関数は単一の結果の代わりに結果の *sequence* シーケンスを返します。このために接続線 B が存在します: 結果シーケンス内の各アイテムのためにターゲットファイル内に新規の行が作成されます。マッピングの出力は上記に従い変更されています（各部署内の職員数に結果が対応していることにご注意してください）:

```
<rows>
    <row>
        <col1>3</col1>
    </row>
    <row>
        <col1>2</col1>
    </row>
    <row>
        <col1>6</col1>
    </row>
    <row>
        <col1>4</col1>
    </row>
    <row>
        <col1>2</col1>
    </row>
    <row>
        <col1>1</col1>
    </row>
    <row>
        <col1>3</col1>
```

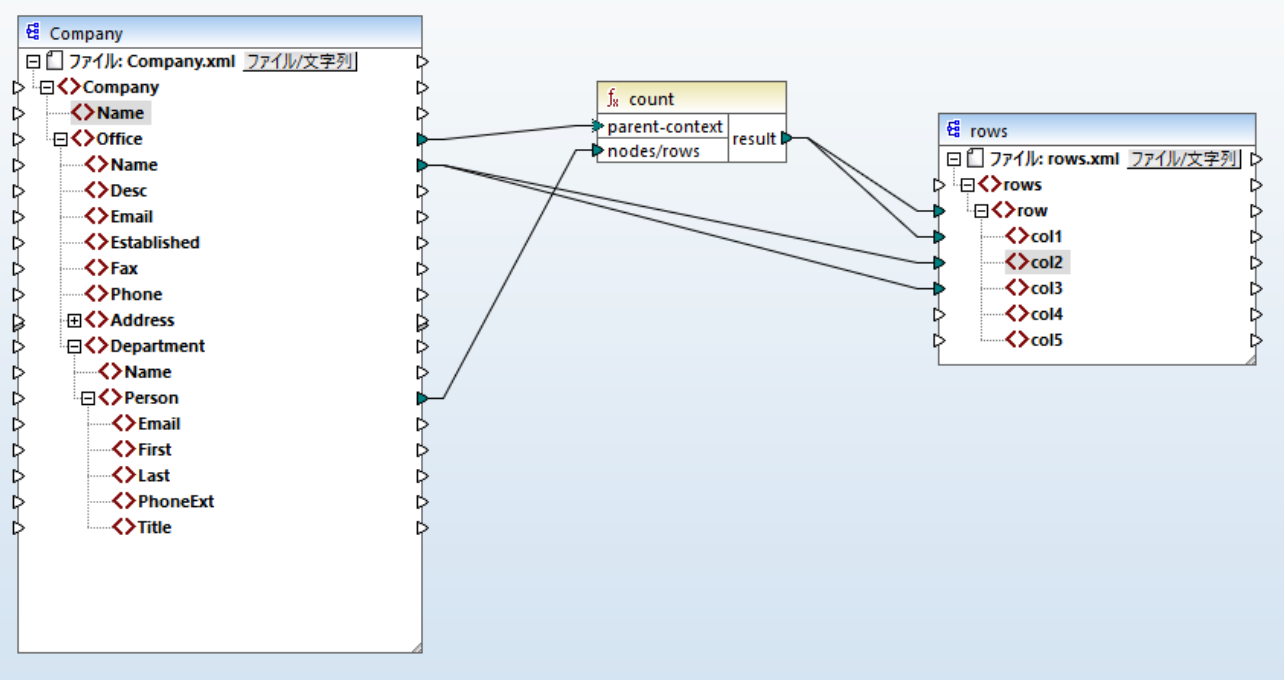
```
    </row>
</rows>
```

現在のマッピングは各部署のために行を作成することから、接続線 C と D を描くことにより、ターゲットファイル内にオフィス名と部署名もコピーすることができます:



このようにして、出力で職員数だけではなく、対応するオフィス名と部署名も表示されます。

各オフィス内の職員数を数えるためには、`count` 関数の親コンテキストをソース内の**Office** アイテムに接続します。



上記の接続では `count` 関数は各オフィスのために1つの結果を返します。ソースファイル内に2つのオフィスが存在するために、関数は2つのシーケンスを返します。この結果、出力内には各行がそのオフィス内の職員数を示す2つの行が存在します:

```
<rows>
    <row>
        <col1>15</col1>
        <col2>Nanonull, Inc.</col2>
    </row>
    <row>
        <col1>6</col1>
        <col2>Nanonull Partners, Inc.</col2>
    </row>
</rows>
```

4.13.3 優先コンテキスト

優先コンテキストは関数の入力パラメーターが評価される順序を影響する方法です。優先コンテキストの設定はマッピングがデータを2つの関連しないソースからジョインする場合必要になる場合があります。

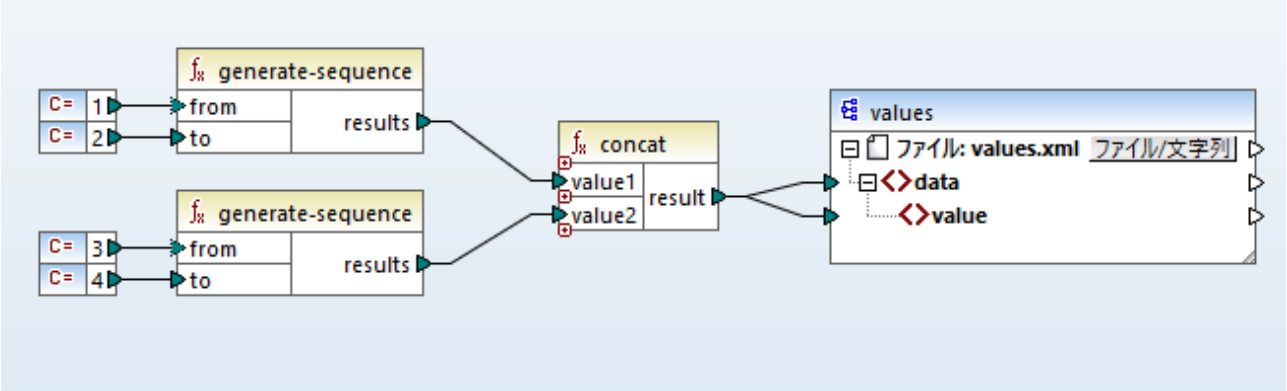
優先コンテキストのしくみを理解するには、マッピングの実行時入力アイテムへの接続が複数の値の *シーケンス* を持つ場合があることを思いだしてください。2つの入力パラメーターを持つ関数の場合、これはMapForce が1つのループが最初に処理される必要がある2つのループを作成する必要があることを意味します。最初に処理されるループは「外側の」ループです。例えば `equal` 関数は2つのパラメーターを受け取ります: *a* と *b*。*a* と *b* の両方が値のシーケンスを受け取る場合 MapForce は以下のように処理します:

- *a* の個別の発生のために
  - *b* の個別の発生のために
    - *a* は *b* と等価か?

上記で明確のように *b* は各 *a* のコンテキスト内で評価されます。優先コンテキストは処理ロジックを変更することを許可するため、各 *a* は各 *b* のコンテキストで評価されます。すなわち、内部ループと外部ループを置き換えることが可能になります。例:

- *b* の個別の発生のために
  - *a* の個別の発生のために
    - *a* は *b* と等価か?

マッピングの結果に影響を与える優先コンテキストを見てみましょう。下のマッピングでは `concat` 関数には2つの入力パラメーターが存在します。各入力パラメーターは `generate-sequence` 関数の助けを使用して生成されるシーケンスです。最初のシーケンスは「1,2」で2番目のシーケンスは「3,4」です。



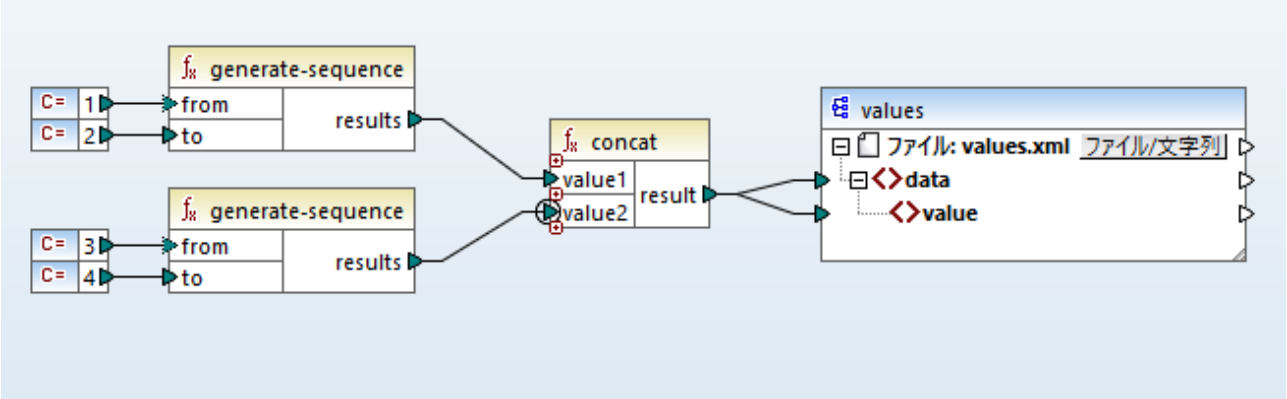
最初に、優先コンテキストを設定せずにマッピングを実行してみましょう。`Concat` 関数はトップのシーケンスの評価を開始し、次の順序で値を連結します:

- 1 と3
- 1 と4
- 2 と3
- 2 と4

これはマッピングにも反映されています:

```
<data>
    <value>13</value>
    <value>14</value>
    <value>23</value>
    <value>24</value>
</data>
```

2番目の入力パラメーターを右クリックし 優先コンテキスト をコンテキストメニューから選択すると、優先コンテキストに設定されます。下に示されているように、優先コンテキスト入力は丸で囲まれています。



今回は2番目の入力パラメーターが最初に評価されます。`concat` 関数は同じ値と連結されていますが、シーケンス `3,4` を最初に処理します。結果的に、出力は以下のとおりになります:

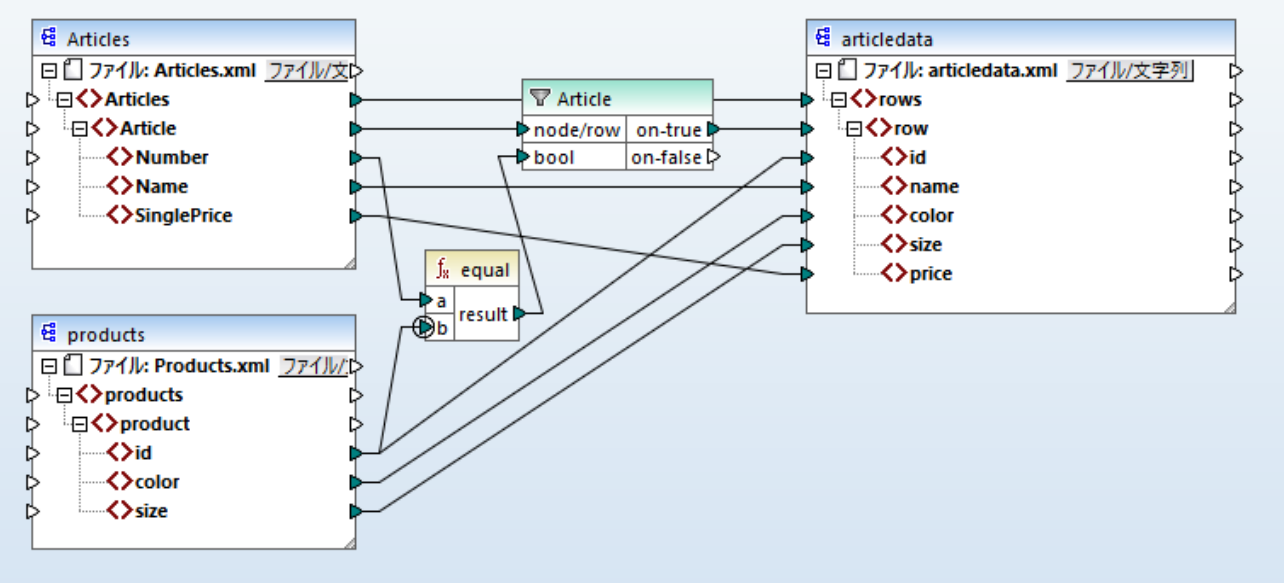
```
<data>
    <value>13</value>
    <value>23</value>
    <value>14</value>
    <value>24</value>
</data>
```

ここまでは、優先コンテキストの理論的な部分のみが紹介されています。実質的なシナリオに関しては[例: 優先コンテキストを使用してフィルターする]を参照してください。

4.13.3.1 例: 優先コンテキストを使用してフィルターする

関数がフィルターに接続されている場合、優先コンテキストは巻数のみではなく、フィルターの評価にも影響を与えます。下のマッピングは正確な出力を取得するために優先コンテキストの設定が必要とされる典型的なケースを説明しています。このマッピングを次のパスで見つけることができます: <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥**FilterWithPriority.mfd**。

メモ このマッピングはXML コンポーネントを使用しますが、下に説明されている同じロジックがMapForce 内のEDI, JSON, などを含む他のすべてのコンポーネントに適用されます。



このマッピングの目的は**Articles.xml** からデータを 異なるスキーマ**articledata.xml**を持つ新規のXML ファイルにコピーすることです。同時にマッピングは**Products.xml** ファイル内の各アーティクルの詳細をルックアップし、対応するアーティクルのレコードにジョインします。**Articles.xml** 内の各レコードは**Number** を持ち、**Products.xml** 内の各レコードは**id** を持っています。これら2 つの値が等価の場合、他のすべての値（**Name**、**SinglePrice**、**color**、**size**）はターゲット内の同じ **row** にコピーされます。

この目的はフィルターを追加することにより達成することができます。各フィルターはブール条件を入力として必要とします。条件を満たす nodes/row のみがターゲットにコピーされます。この目的のために`equal` 関数がマッピング上に存在します。`equal` 関数はアーティクル番号と製品 ID が両方のソースで等価化をチェックします。結果は入力としてフィルターに提供されます。**true** の場合 **Article** アイテムはターゲットにコピーされます。

2番目の`equal` 関数の2番目の入力パラメーター上で優先コンテキストが定義されていることに注意してください。このマッピングでは、優先コンテキストは大きな違いをもたらし、優先コンテキストを設定しない場合、間違ったマッピングの出力が結果として生成されます。

初期のマッピング: 優先コンテキストなし

優先コンテキストを持たないマッピングロジック:

- 一般的なマッピングルールに従うとフィルターの条件を満たす **Article** と新規の**row** がターゲット内で作成されます。（関数とフィルターを使用して） **Article** と**row** 間の接続線はこの部分を対応します。
- フィルターは各アーティクルのために条件をチェックします。これを処理するには、全ての製品内で反復し、現在のコンテキスト内に複数の製品をもたらします。
- ターゲット側で**id** を作成するには、MapForce は一般的なルールに従います（ソース内の核アイテムのためにターゲット内でアイテムが作成されます）。しかしながら、上記の通り**Products.xml** からのすべての製品が現在のコンテキスト内に存在します。特定の製品の**id** を読み取るための**product** と他の箇所への接続線が存在しません。この結果、複数の**id** 要素はターゲット内の各 **Article** のために作成されます。**color** と**size** にも同様な状況が発生します。

要約: **Products.xml** からのアイテムは各製品を反復する必要がある）フィルターコンテキストを有します。このため、**id**、**color**、および **size** 値は各ターゲット **row** にソースファイル内の製品の数だけコピーされ、以下のように不正確な出力を生成します:

```
<rows>
    <row>
        <id>1</id>
```

```
        <id>2</id>
        <id>3</id>
        <name>T-Shirt</name>
        <color>red</color>
        <color>blue</color>
        <color>green</color>
        <size>10</size>
        <size>20</size>
        <size>30</size>
        <price>25</price>
    </row>
</rows>
```

ソリューション A: 優先コンテキストの使用

上記の問題は関数にフィルターのブール条件を計算する優先コンテキストを追加することで解決することができます。

具体的には、`equal` 関数の2番目の入力パラメーターが優先テキストとして選択されている場合、**Products.xml** からのシーケンスが優先されます。これは次のマッピングロジックに反映されます:

- 各製品のために`equal` 関数の入力 **b** が作成されます（すなわち **b** が優先されます）。この段階では、現在の製品の詳細はコンテキスト内に存在します。
- 各アーティクルのために`equal` 関数の入力 **a** は条件がtrue かをチェックします。条件を満たしている場合、アーティクルの詳細を現在のコンテキストにも配置します。
- 次に、アーティクルと製品の詳細を現在のコンテキストからターゲット内の対応するアイテムにコピーします。

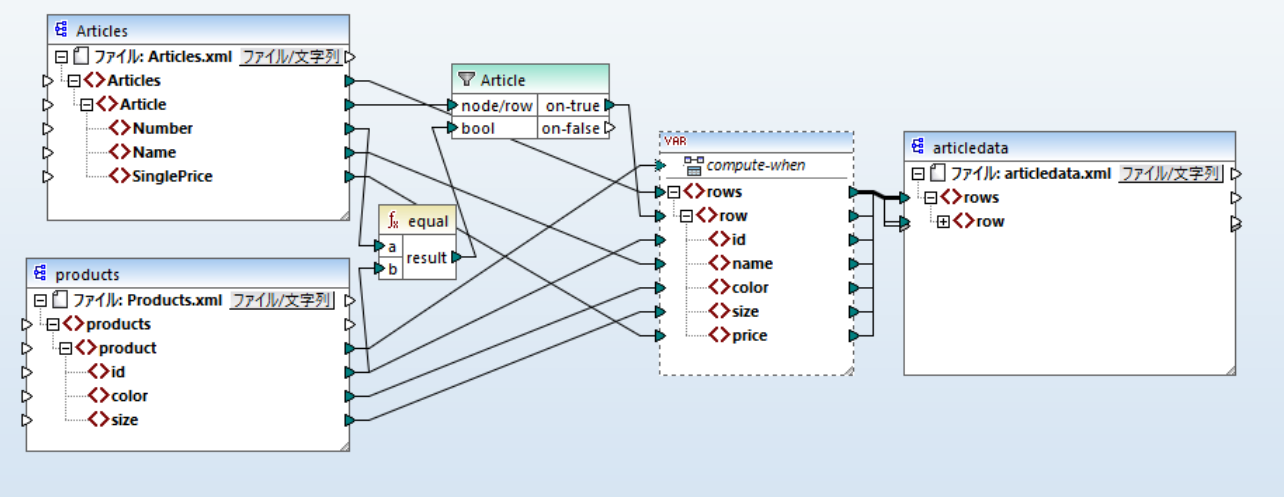
製品のマッピングロジックは正確な出力です。例:

```
<rows>
    <row>
        <id>1</id>
        <name>T-Shirt</name>
        <color>red</color>
        <size>10</size>
        <price>25</price>
    </row>
</rows>
```

ソリューション B: 変数の使用

代替ソリューションとして、中間変数を使用してフィルターの条件を満たす各アーティクルと製品を同じコンテキストに置くことができます。マッピング上にデータを一時的に保管しコンテキストを必要に応じて変更することができるため、変数はこのようなシナリオに適しています。

このようなシナリオのために、マッピングにターゲットコンポーネントと同じスキーマを持つ変数を追加することができます。「**挿入**」メニューで「**変数**」をクリックして **articledata.xsd** スキーマをプロンプトされると構造として提供します。



上記のマッピングでは次が発生します:

- 優先コンテキストはもう使用されません。代わりにターゲットコンポーネントと同じ構造を持つ変数が存在します。
- いつも通り、マッピングの実行はターゲットルートノードから開始されます。ターゲットを作成する前に、マッピングはデータを変数に収集します。
- 各製品のコンテキスト内で変数が計算されます。これが発生する理由は**product** から変数の**compute-when** 入力への接続が存在するからです。
- フィルターの条件はこのため各製品のコンテキスト内でチェックされます。条件がtrue の場合のみ変数の構造は作成され、ターゲットにパスされます。

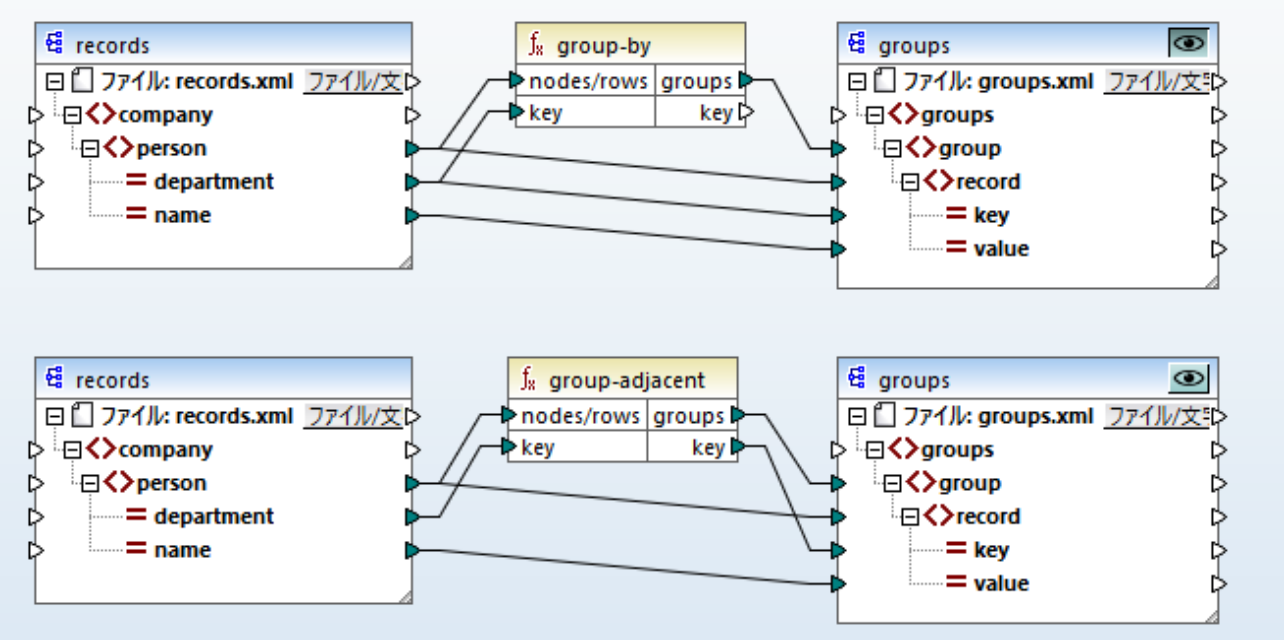
4.13.4　複数のターゲットコンポーネント

マッピングには複数のソースとターゲットコンポーネントが存在する場合があります。複数のターゲットコンポーネントが存在する場合 MapForce 内では一度に1つのコンポーネント出力のみ プレビューボタンをクリックすることによりプレビューすることができます。他の実行環境では(MapForce Server または生成されたコード) 全てのターゲットコンポーネントがシーケンスで実行され、各コンポーネントの対応する出力が生成されます。

デフォルトでは、ターゲットコンポーネントは上から下、左から右に処理されます。必要であれば、マッピングウィンドウ内でターゲットコンポーネントの位置を変更することにより処理順序に影響を与えることができます。レファレンスポイントは各コンポーネントの左角です。以下の点に注意してください:

- 2つのコンポーネントが同じ垂直方向の位置にある場合、左側が優先されます。
- 2つのコンポーネントが同じ水平方向の位置にある場合、上側が優先されます。
- コンポーネントが同じ位置に存在することは少なく、定義された順序を保証するが、変更不可能な一意の内部コンポーネント ID が自動的に使用されます。

この仕組みを説明するサンプルとしては、以下のデモマッピングを開いてください: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\GroupingFunctions.mfd**。このマッピングは複数のソースと複数のターゲットコンポーネントから構成されています。マッピングの一部のみが表示されています。



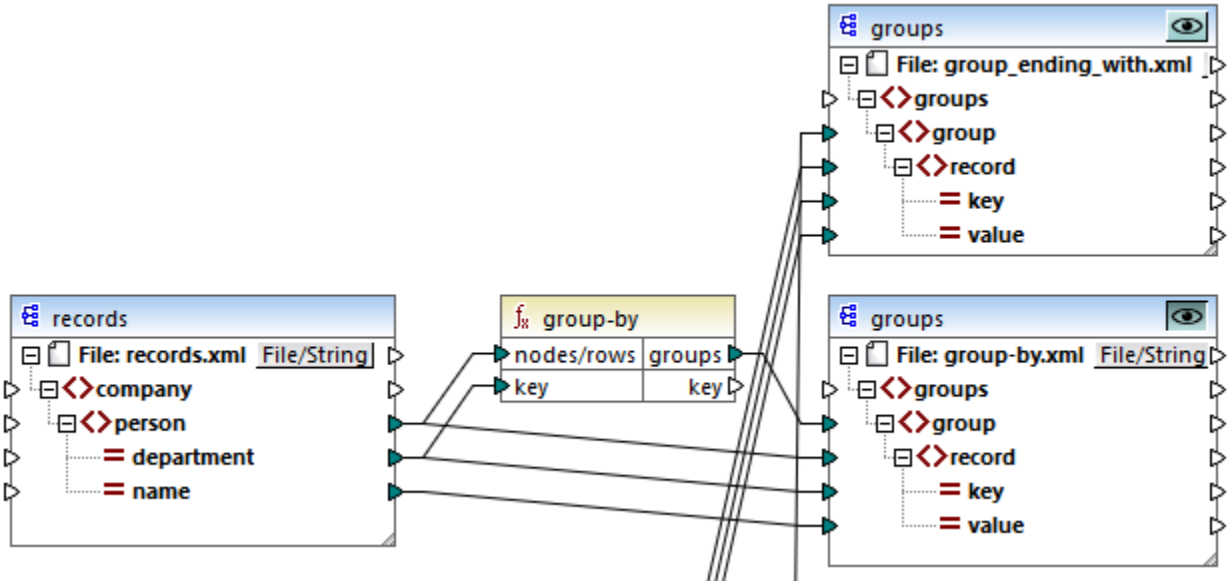
ルールに従うと、MapForce Server 内のこのマッピングと生成されたコード内のデフォルトの処理順序は上から下です。この場合 XSLT 2.0 コードを生成してチェックすることができます。

1. 「ファイル」メニューから「コード生成 | XSLT 2.0」をクリックします。
2. プロンプトされると、生成されるコードのためのターゲットディレクトリを選択します。

生成後、ターゲットディレクトリには複数の XSLT ファイルと **DoTransform.bat** ファイルが含まれています。後者は(個別のライセンスを必要とする) RaptorXML Server により実行されることができます。**DoTransform.bat** ファイルはマッピングの定義と同じ順序で上から下にコンポーネントを処理します。これは各変換の `--output` パラメーターを確認することで検証可能です。

```
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-by.xml" --xml-
validation-error-as-warning=true %* "MappingMapTogroups.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-adjacent.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-into-blocks.xml"
--xml-validation-error-as-warning=true %* "MappingMapTogroups3.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v2.xml" --output="group-starting-
with.xml" --xml-validation-error-as-warning=true %* "MappingMapTogroups4.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v3.xml" --
output="group_ending_with.xml" --xml-validation-error-as-warning=true %*
"MappingMapTogroups5.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

最後の変換は **group-ending-with.xml** と呼ばれる出力ファイルを生成します。このターゲットコンポーネントをマッピングの上の部分に移動します:



XSLT 2.0 コードをもう一度生成すると、処理順序が変更に従い変更されます:

```
RaptorXML xslt --xslt-version=2 --input="records-v3.xml" --
output="group_ending_with.xml" --xml-validation-error-as-warning=true %*
"MappingMapTogroups.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-by.xml" --xml-
validation-error-as-warning=true %* "MappingMapTogroups2.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-adjacent.xml" --
xml-validation-error-as-warning=true %* "MappingMapTogroups3.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records.xml" --output="group-into-blocks.xml"
--xml-validation-error-as-warning=true %* "MappingMapTogroups4.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
RaptorXML xslt --xslt-version=2 --input="records-v2.xml" --output="group-starting-
with.xml" --xml-validation-error-as-warning=true %* "MappingMapTogroups5.xslt"
IF ERRORLEVEL 1 EXIT/B %ERRORLEVEL%
```

上のコードリスティングでは、最初の呼び出しは **group-ending-with.xml** を生成します。

他のコード言語と生成された MapForceServer 実行ファイル(.mfx) 内で同様に処理順序を変更することができます。

チェーンされたマッピング

上記と同様の処理シーケンスがチェーンされたマッピングにも適用されますチェーンされたマッピンググループは 1 つのユニットとしてとられます。単一のチェーンされたマッピングの中間または最終ターゲットコンポーネントの位置変更は処理シーケンスに影響を与えません。マッピング内に存在する複数の「チェーン」または複数のターゲットコンポーネントのみが各グループの最初に処理される最終ターゲットコンポーネントの位置を決定します。

- 最終ターゲットコンポーネントが同じ垂直の位置にある場合、左側が優先されます。
- 最終ターゲットコンポーネントが同じ水平の位置にある場合、上側が優先されます。
- コンポーネントが同じ位置に存在することは少なく、定義された順序を保証するが、変更不可能な一意の内部コンポーネント ID が自動的に使用されます。

5 データソースとターゲット

このチャプターでは**MapForce Basic Edition** がマッピングに必要なソースとターゲットコンポーネントの型について説明されています。

- XML とXML スキーマ

5.1 XML と XML スキーマ

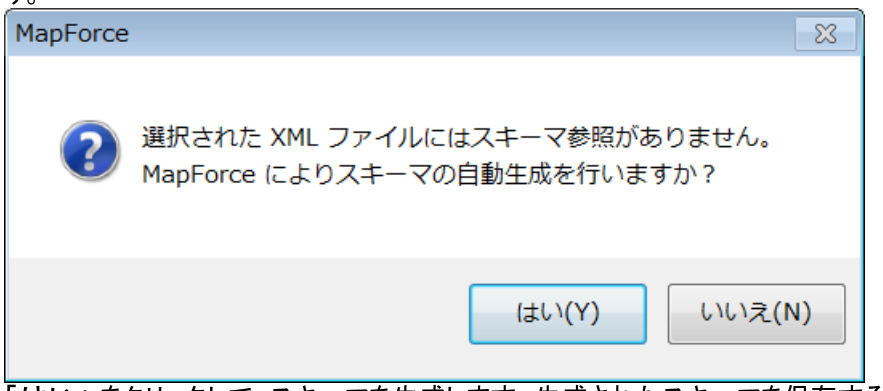
Altova W eb サイト: XML マッピング

このドキュメントのはじめに、XML とXML スキーマファイルをソースまたはターゲットコンポーネントとして、使用する簡単なマッピングのサンプルについて説明しました。このセクションはマッピング内でXML コンポーネントの使用についての詳細について説明します。以下のトピックが含まれます:

- XML スキーマの生成
- XML コンポーネント設定
- DTD をスキーマコンポーネントとして使用
- 派生したXML スキーマ型
- QName
- Nil 値 / Nillable
- コメントと処理命令
- CData セクション
- ワイルドカード – xs:any / xs:anyAttribute
- 複数のスキーマからのデータのマージ
- カスタム名前空間の宣言

5.1.1 XML スキーマの生成

スキーマがない場合、MapForce は、自動的にXML スキーマを既存のファイルをベースに生成することができます。(メニューコマンド「挿入 | XML スキーマ/ファイル」を使用して) マッピングエリアにスキーマを持たないXML ファイルを追加すると、次のダイアログボックスが表示されます。



「はい」をクリックして、スキーマを生成します。生成されたスキーマを保存するディレクトリを選択するようにプロンプトされます。

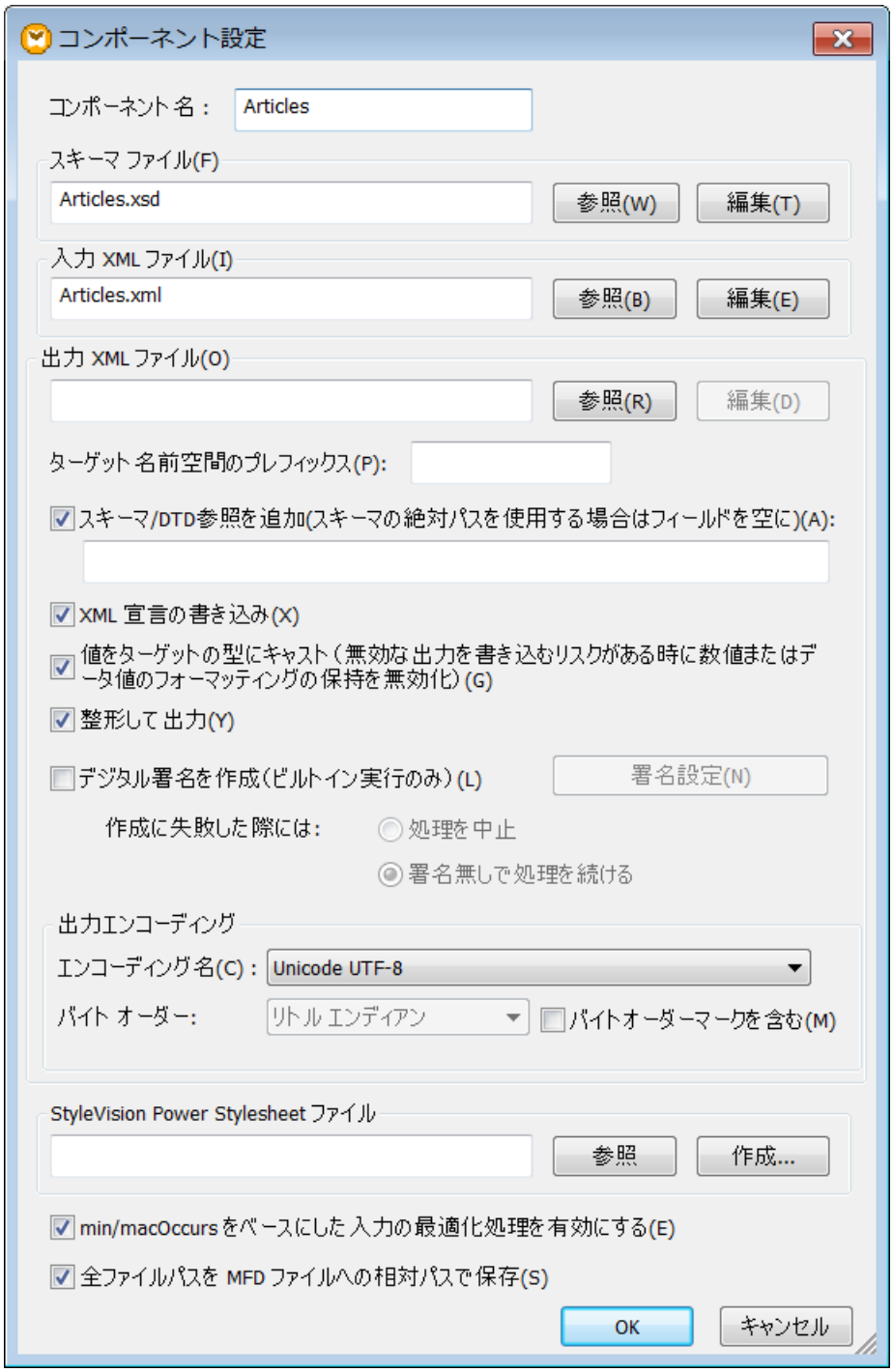
MapForce がXML ファイルからスキーマを生成する場合、要素/属性は、XML インスタンスドキュメントから推定されなければならず、予期しないものの可能性があります。生成されたスキーマがインスタンスデータを正確に表していることを確認することが奨励されます。

要素と属性が1つ以上の名前空間に存在する場合、MapForce は個別のXML スキーマをそれぞれの名前空間のために生成します。このため、複数のファイルがディスク上で作成される可能性があります。

5.1.2 XML コンポーネント設定

マッピングエリアにXML コンポーネントを追加した後、コンポーネント設定ダイアログボックスから適用することのできる設定を構成することができます。コンポーネント設定 ダイアログボックスを以下の方法で開くことができます:

- マッピング上のコンポーネントを選択し、「**コンポーネント**」メニューから「**プロパティ**」をクリックします。
- コンポーネントヘッダーをダブルクリックします。
- コンポーネントヘッダーを右クリックして、「**プロパティ**」をクリックします。



XML コンポーネント設定 ダイアログボックス

使用することのできる設定は、以下のとおりです.

コンポーネント名	コンポーネント名はコンポーネントを作成する際に自動的に生成されます。ですが、後に名前を変更することが可能です。

	コンポーネント名が自動的に生成され、*インスタンスファイル*をその後に選択した場合、コンポーネント名もオプションで更新するようにMapForce は、プロンプトします。 コンポーネント名 はスペース(例: "ソースXML ファイル") またフルストップ文字も含むことができます(例: "Orders.EDI")。コンポーネント名は、スラッシュ、バックスラッシュ、コロン、二重引用符、行頭および末尾スペースを含んではなりません。全般的に、コンポーネントの名前を変更するさいの以下の影響を考慮してください: • マッピングを FlowForce Server にデプロイする場合、コンポーネント名は一意である必要があります。 • コマンドラインに入力できる文字のみを使用することが奨励されます。コマンドラインとW indows 内では、それぞれの文字で異なったエンコードを使用する場合があります。
スキーマファイル	MapForce によりデータを検証とマップするために使用されたXML スキーマファイルの名前とパスを指定します。 スキーマファイルを変更するには、「参照」をクリックして、新しいファイルを選択します。XMLSpy 内のファイルを編集するには、「編集」をクリックします。
入力 XML ファイル	MapForce がデータを読み込むXML インスタンスファイルを指定します。このフィールドはソースコンポーネントにより重要で、最初にコンポーネントを作成した際に入力され、XML インスタンスファイルに割り当てられます。 ソースコンポーネント内で、インスタンスファイル名はXML ルート要素とレファレンススキーマを検出するため、また選択されたスキーマに対して検証するために使用されます。 ファイルのロケーションを変更するには、「参照」新しいファイルを選択します。 XMLSpy 内のファイルを編集するには、「編集」をクリックします。
出力 XML ファイル	MapForce がデータを書き込むXML インスタンスファイルを指定します。このフィールドはターゲットコンポーネントにより重要です。 ファイルのロケーションを変更するには、「参照」をクリックして、新しいファイルを選択します。XMLSpy 内のファイルを編集するには、「編集」をクリックします。
ターゲット 名前空間のためのプレフィックス	ターゲット名前空間のためにプレフィックスを入力することを許可します。プレフィックスを割り当てる前に、ターゲットスキーマ内でターゲット名前空間が定義されている必要があります。
スキーマ/DTD レファレンスの追加	参照されたXML スキーマファイルをXML 出力のルート要素に対してパスを追加します。このフィールドに入力されたスキーマのパスは、`xsi:schemaLocation` 属性またはDTD が使用される場合、`DOCTYPE` 宣言 内の生成されたターゲットインスタンスファイルに書き込まれます。 このフィールドにパスを入力することにより、XML インスタンスファイルにより参照されたスキーマファイルがどの箇所に位置するかを決定することができます。これにより、マッピングが実行される際、出力インスタンスがマッピングのデスティネーションで検証されることを保証できます。また、**http://**アドレスや絶対及び相対パスをこのフィールドに入力することもできます。 このオプションを無効化することにより、XML インスタンスを参照されたXML スキーマまたはDTD から、切り離すことができます(例: 結果するXML 出力を基となるXML スキーマを持たない相手に送信する場合など)。

<table>
<tr><td>XML 宣言の書き込み</td><td>このオプションにより生成された出力からXML 宣言 を表示しないようにすることができます。デフォルトでは、オプションが有効化されていると、XML 宣言 が出力に表示されます。

この機能はMapForce ターゲット 言語と実行エンジンでサポートされています。

<table>
<tr><th>ターゲット言語 / 実行エンジン</th><th>出力がファイルの場合</th><th>出力が文字列の場合</th></tr>
<tr><td>XSLT, XQuery</td><td>はい</td><td>いいえ</td></tr>
</table>
</td></tr>
<tr><td>ターゲット型に値をキャストする</td><td>Aターゲット XML スキーマ型がマッピングで使用されるかを定義します。またはターゲットコンポーネントにマップされる全てのデータが文字列値として扱われるかを定義します。デフォルトでは、この設定 が有効化されています。

このオプションを無効化することにより、正確なフォーマットの値を保持することができます。例: これは、数値内で特定の小数点を必要とするスキーマ内のパターンファセットを満足させる場合に役にたちます。

マッピング関数を使用して、数値を文字列として必要とされるフォーマット内で使用し、この文字列をターゲットにマップすることができます。

このオプションを無効化にすると、無効な値の検出も無効化されます。例: 数値フィールドに文字が入力された場合。</td></tr>
<tr><td>整形出力</td><td>出力 XML ドキュメントに構造化された外見を与えるために整形出力します。それぞれの子ノードは、単一タブ文字のそれぞれの親のオフセット です。</td></tr>
<tr><td>出力エンコード</td><td>出力インスタンスファイルの次の設定を指定することを許可します:

• エンコード名
• バイトオーダー
• バイトオーダーマーク（BOM） 文字が含まれるか否か。

デフォルトでは、「新しいコンポーネントのためのデフォルトのエンコード」オプション内で定義されたエンコードを持つ新しいコンポーネント。「ツール | オプション」からこのオプションにアクセスし、タブを生成します。

マッピングがXSLT 1.0/2.0 を生成すると、バイトオーダーマーク チェックボックスを有効化しても、これらの言語はバイトオーダーマークをサポートしないため効果はありません。</td></tr>
<tr><td>StyleVision Power スタイルシートファイル</td><td>このオプションにより、Altova StyleVision スタイルシートファイルを選択または作成することができます。このようなファイルは、XML インスタンスファイルからのデータをHTML、RTF などの多種のレポートに適したフォーマットで出力することができます。

以下も参照: コンポーネント上で相対パスを使用する</td></tr>
<tr><td>min/maxOccurs をベースに入力処理の最適化の有効化</td><td>このオプションにより、<code>minOccurs</code> と<code>maxOccurs="1"</code> を持つ必要な属性または子要素 などの1つのアイテムのみを含むシーケンスを特別に処理することが許可されます。この場合、シーケンスの最初のアイテムが抽出され、アイテムは シーケンスとしてではなく直接動的な値として処理されます</td></tr>
</table>

	入力データがスキーマに対して場合、有効でない場合、エラーメッセージと共にマッピングを停止する空のシーケンスがマッピング内で生じる可能性があります。このような無効な入力を処理するには、このチェックボックスを無効化にします。
MFD ファイルに相対的なすべてのファイルパスを保存する	このオプションが有効化されると、MapForce は、コンポーネント設定ダイアログボックスに表示された、MapForce Design (.mfd) ファイルの場所に相対したファイルパスを保存します。以下も参照: コンポーネント上で相対パスを使用する。

5.1.3 "スキーマ" コンポーネントとしてを DTD 使用する

MapForce 2006 SP2 以降では、ソースならびにターゲットコンポーネントにて名前空間を意識したDTD がサポートされます。名前空間 URI がDTD の"xslns" 属性宣言から抽出され、マッピングを行うことが可能になります。

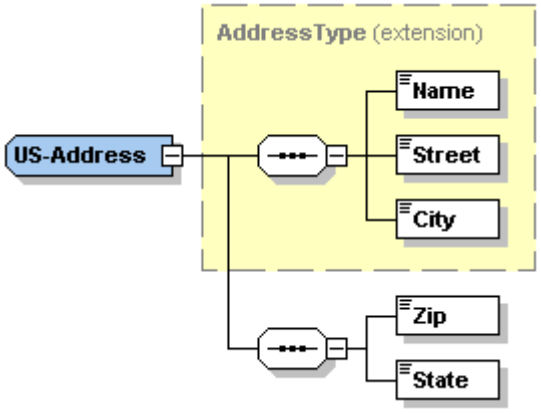
しかし StyleVision により使用されるDTD の様に、名前空間 URI を持たずにxmlns* 属性を含むDTD も存在します。このようなDTD は、MapForce で使用することができるように拡張する必要があります。以下に示されるように、DTD を修正して名前空間 URI を含むxmlns 属性を定義する必要があります:

```
<!ATTLIST fo:root
   xmlns:fo CDATA #FIXED 'http://www.w3.org/1999/XSL/Format'
   ...
>
```

5.1.4 派生した XML スキーマの型

MapForce では複合型の派生型に対するマッピングがサポートされます。派生型は、**xsi:type** 属性により特定の派生型を指定する XML スキーマの複合型です。

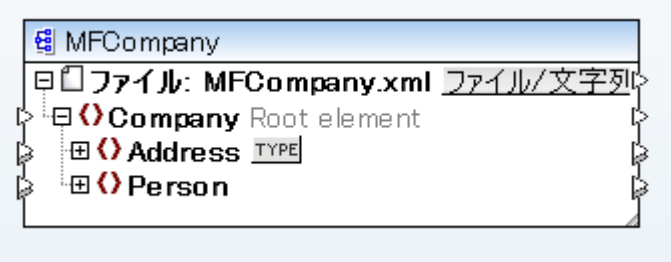
以下のスクリーンショットは、XMLSpy にて派生型の"US-Address" の定義を示したものです。基底型(または オリジナルとなっている複合型) はこの場合 AddressType となります。Zip とState の2 つの要素を新たに追加することで、派生型のUS-Address が作成されます。



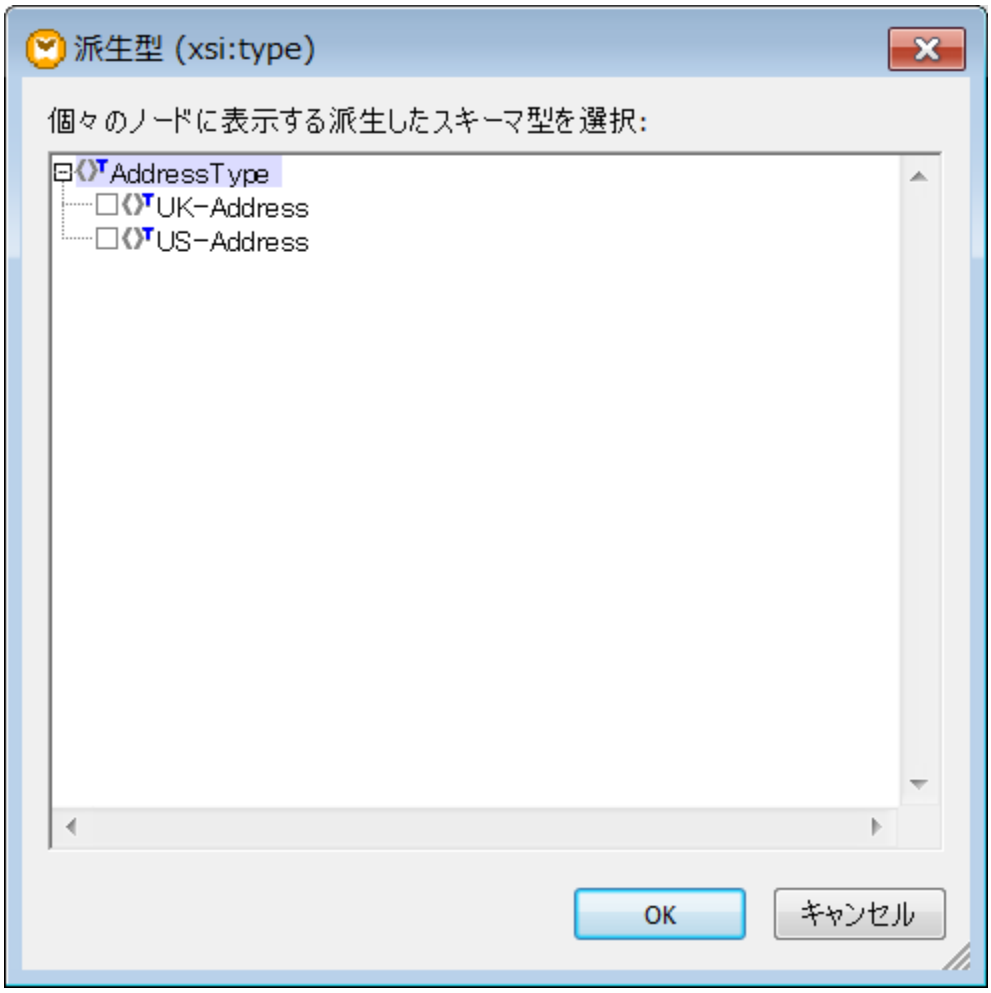
生成された型のサンプル(XMLSpy スキーマビュー)

次のサンプルは、派生したXML スキーマ型からデータをマップ、またはスキーマ型へのデータのマップの方法を説明しています。

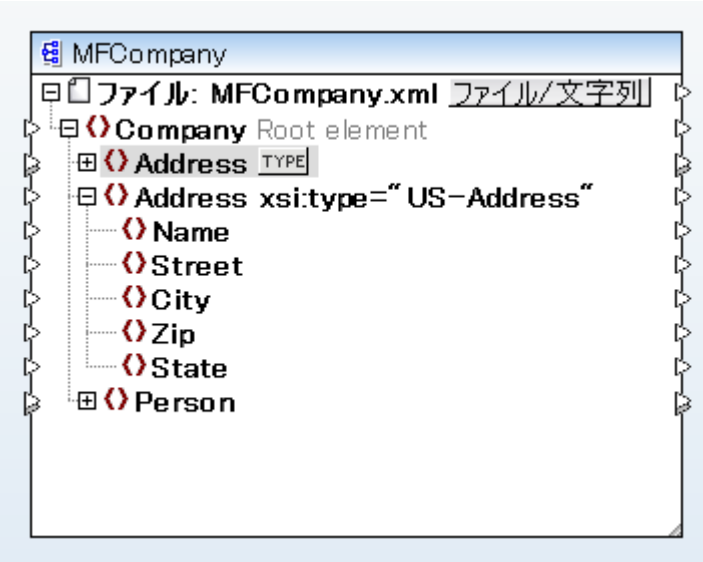
1. 「挿入」メニューから「XML スキーマ/ファイル」をクリックします。以下のXML スキーマ：<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\MFCompany.xsd を開きます
2. インスタンスファイルを提供するようにプロンプトされると、「スキップ」をクリックして、ルート要素をとして Company 選択します。



3. Address 要素の横の TYPE ボタンをクリックします。このボタンは、スキーマ内のこの要素のために存在する派生した型であることを示しています。



4. 使用する派生した型の横のチェックボックスを選択し、（この場合、US-Address）、「OK」を押して確認します。新規の要素 Address xsi:type="US-Address" がコンポーネントに追加されます。



これらのアイテムに対して直接、または US-Address から、マッピングを行うことが可能になります。

派生型ダイアログボックスにて複数のアイテムを選択することで、複数の派生型を追加/挿入することができます。各ノードにはそれぞれ独自の xsi:type 要素が与えられます。

5.1.5 QNames

MapForce は、マッピングの実行ランタイムで XML ファイルからデータを読み込む際に QName (修飾名) プレフィックス (http://www.w3.org/TR/xml-names/#ns-qualnames) を解決します。

QNames は、XML インスタンスドキュメント内の名前空間 URI を参照および省略するために使用されます。QNames には、2 つの種類があります: プレフィックスを持つ、またはプレフィックスを持たない QNames.

PrefixedName	Prefix ':'	LocalPart
UnPrefixedName		LocalPart
		LocalPart が要素 または 属性の名前の箇所

例: 下のリストでは、<x:p/> が QName です:

- プレフィックス "x" は名前空間 "**http://myCompany.com**" の省略形です。
- p は、ローカルパートです。

```
<?xml version='1.0'?>
<doc xmlns:x="http://myCompany.com">
   <x:p/>
</doc>
```

MapForce は、xpath2 | qname-related 関数 ライブラリに QName に関連した関数の一部を含みます。

5.1.6 Nil の値 / Nillable

XML スキーマ仕様に従えば、nillable="true" 属性をスキーマ内の要素に対して指定することで、(コンテンツのデータ型により空のコンテンツが許されていない場合でも)コンテンツを含まない要素でも妥当とみなすことができるようになります。XML インスタンスドキュメント内に記述される属性のxsi:nil="true" により、要素は存在するが、コンテンツは含まれない(つまり空要素である)ことが示されます。このセクションはどのように、MapForce がソースとターゲットコンポーネント内のnil 要素を扱うかを説明します。

'xsi:nil' と 'nillable' の比較

xsi:nil="true" 属性はXML インスタンスファイルにおいてのみ定義されます。

```
14   <Person>
15     <PrimaryKey>2</PrimaryKey>
16     <ForeignKey>1</ForeignKey>
17     <EMail>biff@amail.com</EMail>
18     <First>biff</First>
19     <Last>bander</Last>
20     <PhoneExt>22</PhoneExt>
21     <OrderID xsi:nil="true"/>
22     <Title>IT services</Title>
23   </Person>
```

xsi:nil 属性はMapForce のマッピングにて視覚的(かつ明示的)に示されることは無く、自動的に処理されます。通常、null 値が許容されたノード(xsi:nil="true" 属性が含まれるノード)は存在するが、コンテンツには何も含まれない状態になります。

`xsi:nil` 属性は、通常自動的に処理されるため、MapForce グラフィカルなマッピングで明示的に表示されません。特に"nilled" ノード(`xsi:nil="true"` 属性を持つ) が存在する場合、そのコンテンツは存在しません。

マッピングソースにある null 値が許容された要素

マッピングにて null 値を許容する(nillable な) データを XML 要素から読み取った場合、xsi:nil 属性が自動的にチェックされます。xsi:nil の値がtrue となっている場合、コンテンツは存在しないものとして扱われます。

nillable なソース要素から*単純コンテンツの*(属性は含まれるが子要素を含まない) nillable なターゲット要素に対して、target-driven 接続を作成し、xsi:nil がソース要素にセットされている場合、xsi:nil 属性がターゲット要素にも挿入されます(例: `<OrderID xsi:nil="true"/>`)。

全てコピー 接続を nillable なソース要素から nillable なターゲット要素へマッピングして、xsi:nil がソース要素にてセットされている場合、xsi:nil 属性がターゲット要素にも挿入されます(例: `<OrderID xsi:nil="true"/>`)。

ソース要素のxsi:nil にtrue がセットされているかを明示的にチェックするには、**`is-xsi-nil`** 関数を使用します。「nil 化」された要素に対して true が返され、それ以外のノードに対して false が返されます。

nil 化された(存在しない)ソース要素の値を別の値で代用するには、**`substitute-missing`** 関数を使用してください。

メモ

- コンテンツが無くても要素ノードが実際存在するため、**`exists`** 関数をnill 化されたソース要素 に接続すると、TRUE を返します。

- xsi:nil が設定されている箇所で（multiply と concat などの）単純型の値を期待する関数を要素に対して使用すると、要素コンテンツが存在しなく、結果を取得することはできません。これらの関数はソースノードが存在しないかのように振舞います。

マッピングターゲットとしての Nillable 要素

ターゲット優先 接続から nillable ソース要素に*単純型コンテンツ*（任意の追加属性は含むが子要素を含まない単一の値）を持つソース要素が xsi:nil に設定されているターゲット要素に作成する場合、MapForce は xsi:nil 属性をターゲット要素に挿入します（例: <OrderID xsi:nil="true"/>）。 xsi:nil="true" 属性が XML ソース要素にセットされていない場合、通常の方法で要素コンテンツがターゲット要素へマッピングされます。

マッピングが（子要素を持つ）*複合型*の nillable 要素に対して行われている場合、xsi:nil 属性が自動的に書き込まれることはありません。これはマッピングが行われた後で子要素が書き込まれるか分からない為です。全てコピー 接続を定義することで、ソース要素から xsi:nil 属性をコピーすることができます。

空のシーケンスまたはデータベースの NULL 値をターゲット要素へマッピングする場合、それが nillable であるかどうかにかかわらず、要素が作成されることはありません。

空のターゲット要素を xsi:nil="true" 属性がセットされた状態で（強制的に）作成するには、**set-xsi-nil** 関数をターゲット要素へ直接接続してください。この方法は単純型ならびに複合型のターゲット要素に対して使用することができます。

substitute-missing-with-xsi-nil 関数を使用することで、マッピングソースから値を得ることができない場合に xsi:nil をターゲットへ挿入することができます。これはソースノードが全く存在しない場合、または（multiply のような）計算に対して nil 化されたソースノードが与えられ、結果が得られなかった場合などに使用することができます。

この関数はデータベースの NULL 値から xsi:nil="true" を持つ要素を作成するために使用することができます。この関数は単純型のコンテンツを持つノードに対してのみ使用することができます。

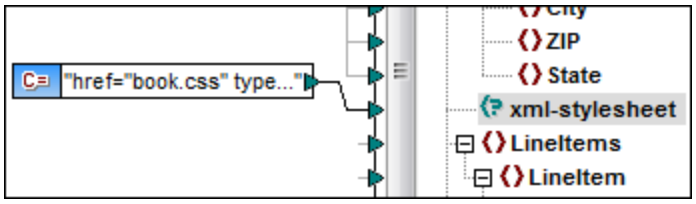
xsi:nil を生成する関数は、（**if-else** 関数などの）値に対して動作する関数またはコンポーネントにパススルーされることはできません。

5.1.7 コメントと処理命令

コメントと処理命令はターゲット XML コンポーネントに挿入することができます。処理命令は、XML ドキュメントを更に処理するアプリケーションに情報をパスするために使用されます。コメントと処理命令は、全てコピーマップグループのためのノードのためには定義することができないことにご注意してください。

処理命令の挿入方法:

1. ターゲットコンポーネント内の要素を右クリックして、コメント/処理命令を選択し、メニュー（前、後）から処理命令 オプションを選択します。
2. 処理命令（ターゲット）名をダイアログ内に入力し、「OK」を押して確認します。例: xml-stylesheet。
これによりこの名前をコンポーネントツリーに追加します。

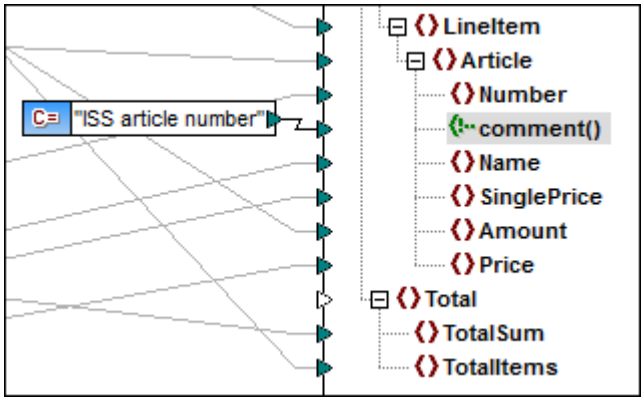


3. 以下を使用することができます。例: 処理命令属性の値を与えるために定数コンポーネントを使用する。例:
`href="book.css" type="text/css".`

メモ
複数の処理命令は、ターゲットコンポーネント内の要素の前または後ろに追加することができます。

コメントの挿入:

1. ターゲットコンポーネント内の要素を右クリックして、コメント/処理命令を選択し、メニューからコメントオプション(前、後ろ)を選択します。



これは、コンポーネントツリーにコメントノード(`<!--comment()`)を追加します。
2. 定数コンポーネントを使用して、コメントコンテキストを与えます、またはソースノードをコメントノードに接続します。

メモ
単一のターゲットノードには1つのコメントのみを追加することができます。複数のコメントを作成するには、入力関数の複製を使用します。

コメント/処理命令の削除:

- 対応するノードを右クリックして、コメント/処理命令を選択し、ブライアウトメニューから「削除」コメント/処理命令を選択します。

5.1.8 CDATA セクション

CDATA セクションは、通常マークアップとして解釈される文字を含むテキストのブロックをエスケープするために使用されます。CDATA セクションは"<![CDATA[" で開始し、"]]>"で終わります。

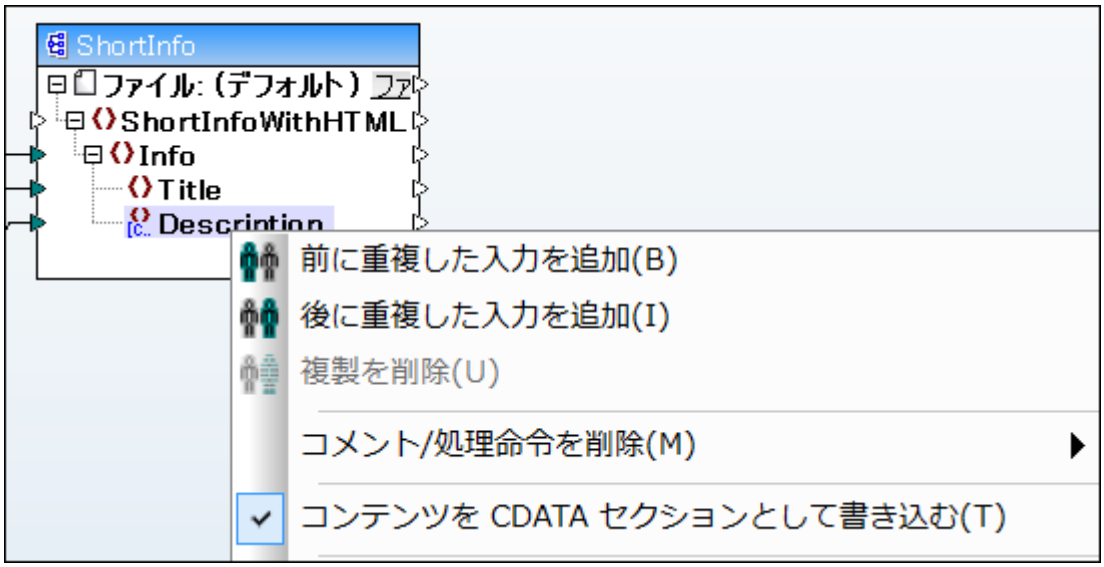
ターゲットノードは、CDATA セクションとして受信される入力データを書き込むことができます。ターゲットノードコンポーネントは以下であることができます:

- XML データ

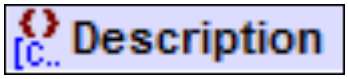
- データベースフィールド内に埋め込まれたXML データ
- XBRL ターゲット内の型指定されたディメンションのXML 子要素

CDATA セクションの作成:

1. CDATA セクションとして定義するターゲットノードを右クリックして、 "CDATA セクションとしてコンテンツを書き込む"を選択します。



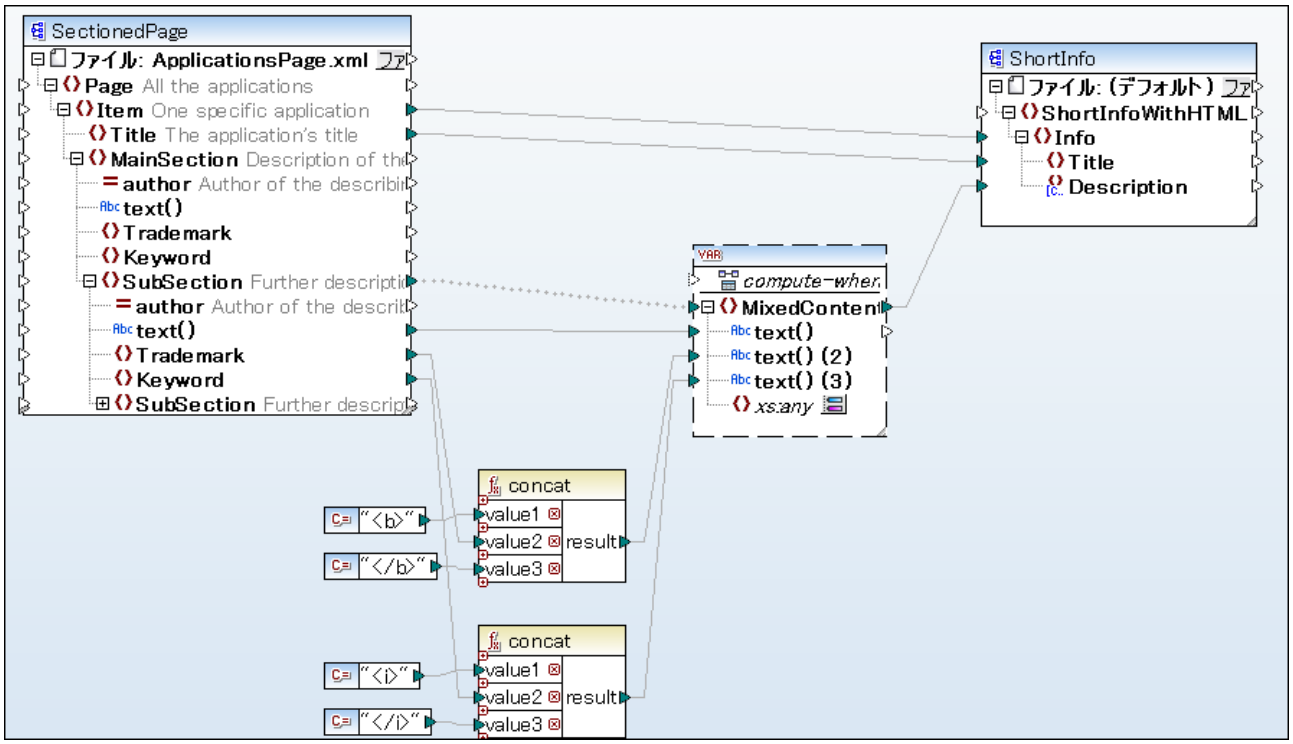
入力データは、区切り記号 ']]>' で終了されたCDATA セクションを含まない 用に警告プロンプトを表示します。「OK」をクリックしてプロンプトを閉じます。
下に表示される [C.. アイコン では、要素タグがこのノードがCDATA セクションとして定義されていることを示しています。



メモ
CDATA セクションは、複製されたノード、および、 xsi:type ノード上でも定義されることができます。

例:
…¥MapForceExamples フォルダー内で使用することのできる **HTMLinCDATA.mfd** マッピングファイルは、CDATA セクション が有効利用できる例を表示しています。

この例:
- 太字の開始 () と終了 () タグは **Trademark** ソース要素のコンテンツに追加することができます。
- イタリックの開始 () と終了 () タグは **Keyword** ソース要素のコンテンツに追加することができます。
- 結果のデータは、サブセクション要素コネクタが ソース優先 (複合型コンテンツ) ノードとして定義されているため、ソースドキュメント内に表示されるために複製 **text()** ノードにパスされます。
- 複合型ノードの出力は、CDATA セクションとして定義されている ShortInfo ターゲット コンポーネント内の **詳細** ノードにパスされます。

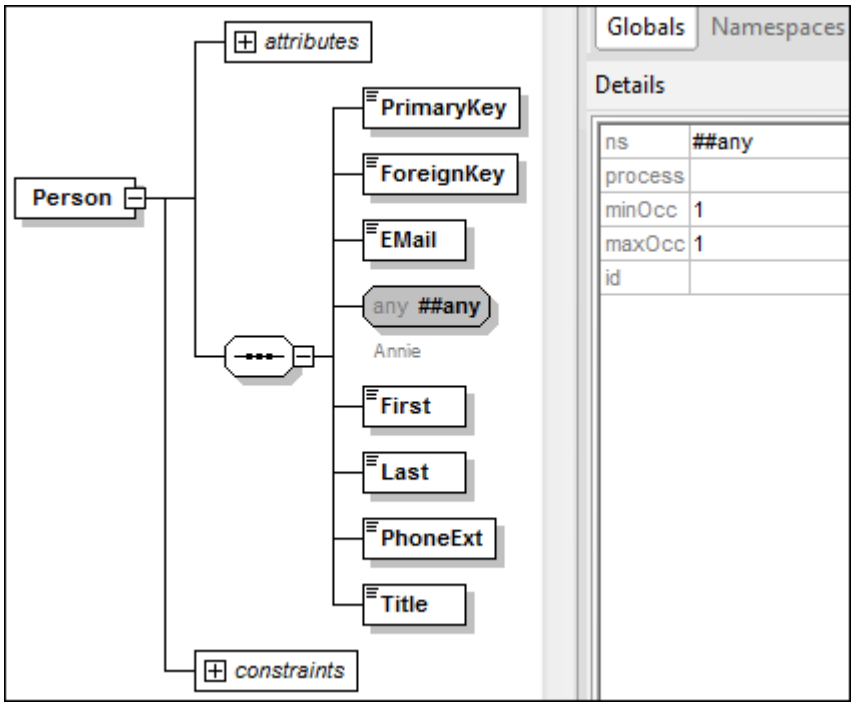


「出力」ボタンをクリックすることにより、マークアップされたテキストを含むCDATA セクションが表示されます。

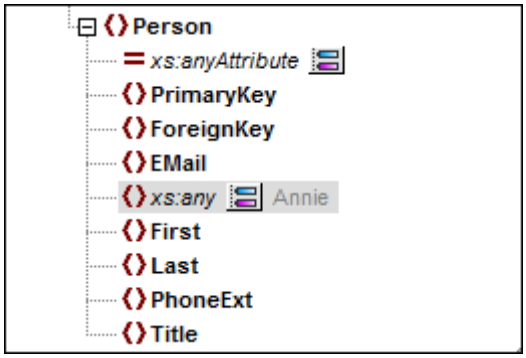
```
7     <Info>
8       <Title>MapForce</Title>
9       <Description><![CDATA[Altova  <b>MapForce</b>  2014 Enterprise Edition is the premier  <i>XML</i>
/ <i>database</i> / <i>flat file</i> / <i>EDI</i>  data mapping tool that auto-generates mapping code in
<i>XSLT</i>  1.0/2.0,  <i>XQuery</i> ,  <i>Java</i> ,  <i>C++</i>  and  <i>C#</i> . It is the definitive tool for
data integration and information leverage.]]></Description>
10    </Info>
```

5.1.9 ワイルドカード – xs:any / xs:anyAttribute

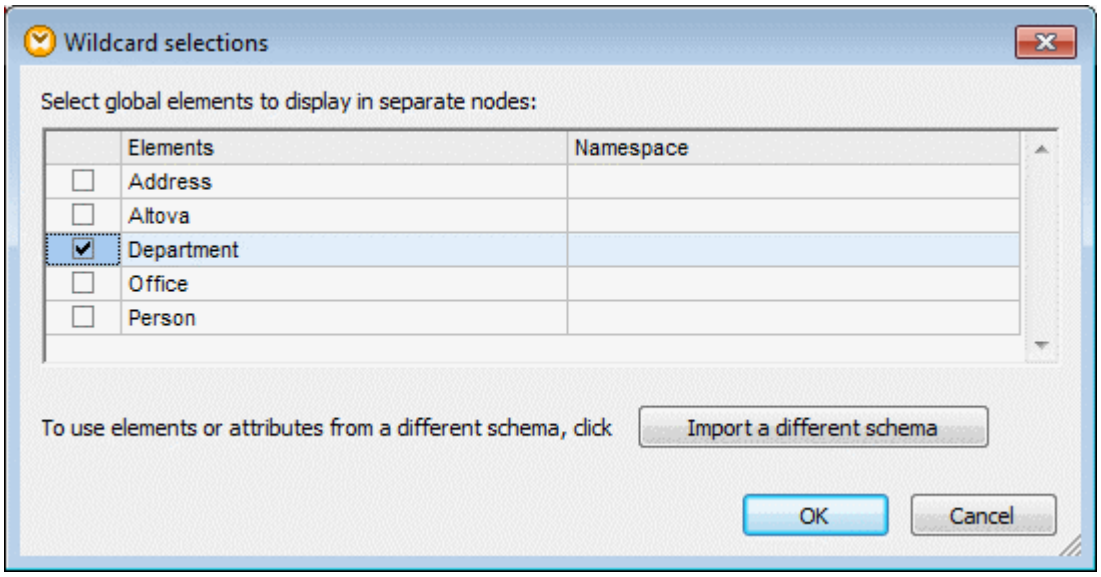
ワイルドカード xs:any (および xs:anyAttribute) は、すべての要素/属性をスキーマから使用することを許可します。スクリーンショットは XMLSpy のスキーマビュー内の"any" 要素を表示しています。



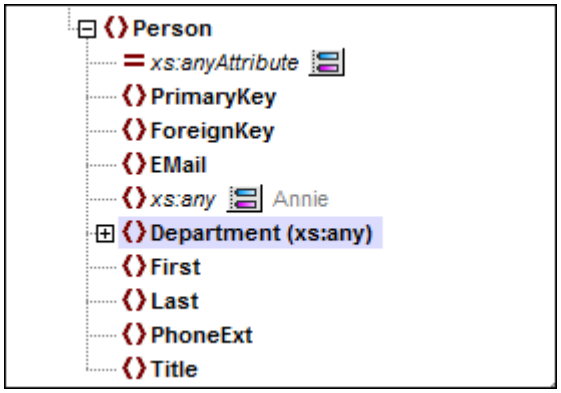
MapForce 内では、選択の変更（ excel1-compicon ）ボタンがxs:any（またはxs:anyAttribute）要素の横に表示されます。



クリックすると、選択の変更 ボタン excel1-compicon が開かれ、"ワイルドカードの選択"ダイアログボックスが開かれます。エントリは、現在のスキーマ内で宣言されているグローバルな要素と属性を表示します。



クリックすると、選択の変更 ボタン excel1-compicon が開かれ、"ワイルドカードの選択"ダイアログボックスが開かれます。エントリは、現在のスキーマ内で宣言されているグローバルな要素と属性を表示します。



.これらのノードから、またはノードへ、他の要素と同様にマップすることができます。

コンポーネント上では、ワイルドカード要素、または属性は、名前に追加されている **(xs:any)** テキストとして識別されることができます。

ワイルドカード要素を削除するには、選択の変更 (excel1-compicon) ボタンをクリックして、"ワイルドカードの選択"ダイアログボックスから選択の解除を行います。

ワイルドカードと動的なノード名

ワイルドカードから、またはワイルドカードへのマッピングは、通常、使用することのできる、要素または属性がインスタンス内に表示されており、XML インスタンスがコンポーネントの XML スキーマにより宣言されている場合(または、外部のスキーマからインポートされている場合)に適しています。しかしながら、インスタンス内に表示される要素または属性が多すぎ、スキーマ内で宣言されるには多すぎる場合があります。`<message>` の子要素の数が任意である次のインスタンスの場合を考慮してみてください:

```
<?xml version="1.0" encoding="UTF-8"?>
<message>
   <line1>1</line1>
   <line2>2</line2>
   <line3>3</line3>
   ...............
   <line999></line999>
</message>
```

上記のようなシチュエーションでは、ノード名への動的なアクセスがワイルドカードの使用より適しています（ノード名のマッピング を参照）。

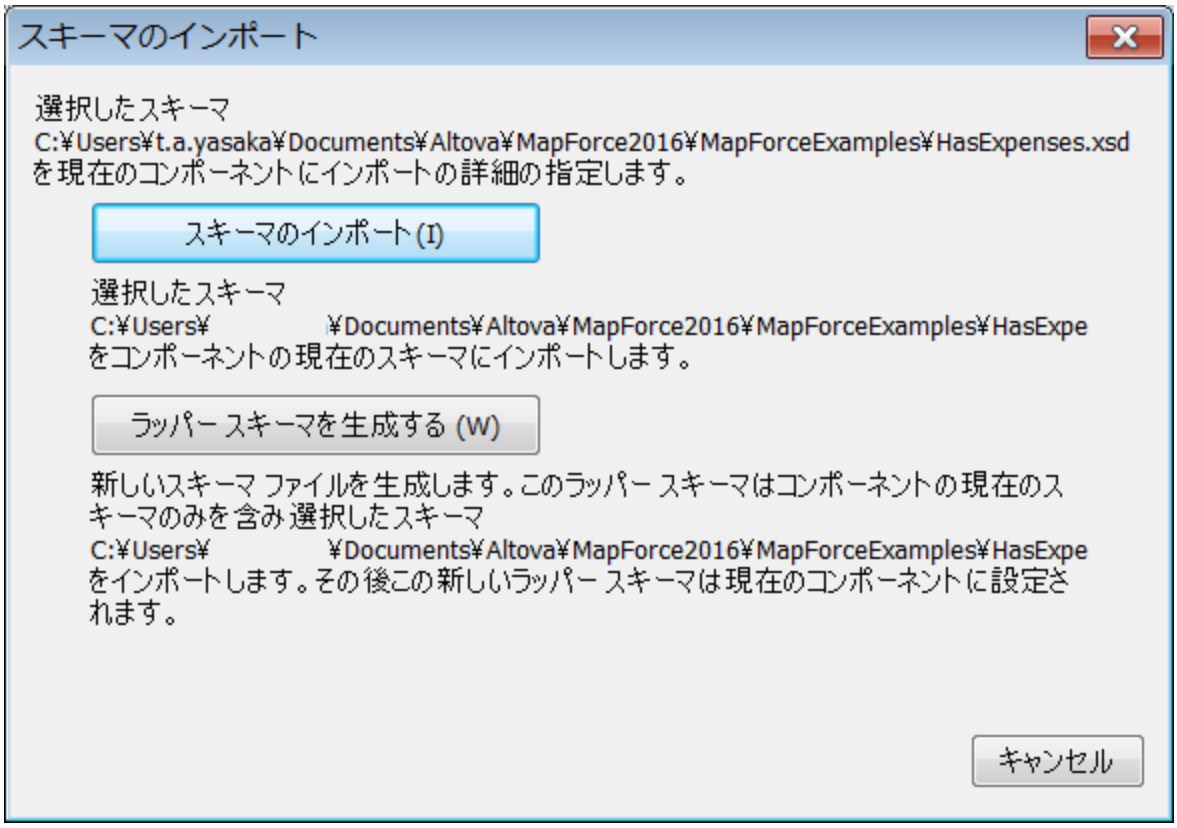
異なるスキーマからワイルドカードとして要素を追加する

コンポーネントに割り当てられた要素以外のスキーマからの要素は、ワイルドカードとして使用することができます。このような要素をコンポーネント上で表示するには、"ワイルドカードの選択" ダイアログボックス上の**異なるスキーマのインポート** ボタンをクリックします。これにより以下の2つのオプションを持つ新規ダイアログボックスが開かれます:

1. スキーマのインポート

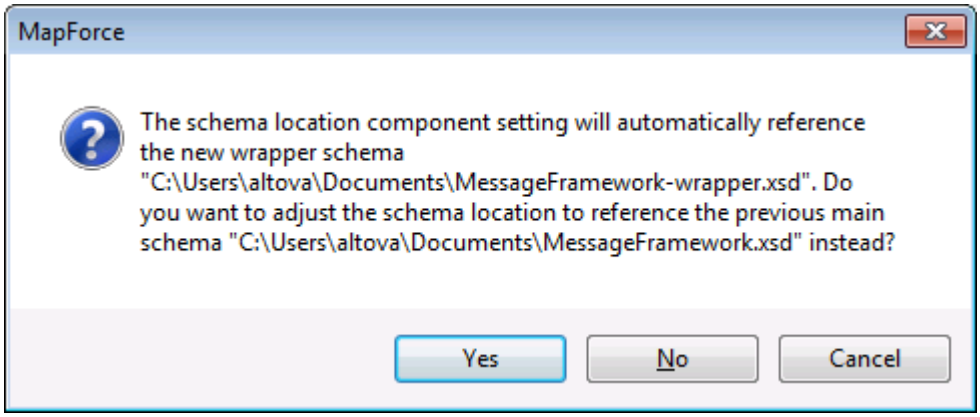
2. ラッパースキーマの生成

例えば、下のイメージは、**HasExpenses.xsd** という外部スキーマを現在のスキーマにインポートしようと試みると何が起こるかを表しています。



スキーマのインポート オプションは外部スキーマをコンポーネントに割り当てられている現在のスキーマにインポートします。このオプションはディスク上のコンポーネントの既存のスキーマをオーバーライドすることに注意してください。現在のスキーマがディスクではなく URL により開かれたリモートのスキーマである場合、変更することはできません（URL からコンポーネントを追加する を参照してください）。この場合、ラッパースキーマの生成オプションを使用してください。

ラッパースキーマの生成オプションは "ラッパー" スキーマという名前の新しいスキーマファイルを作成します。このオプションの利点は、コンテンツの既存のスキーマが変更されない点です。代わりに、既存のスキーマとインポートされるスキーマの両方を含む（ラッパースキーマという）新しいスキーマが作成されます。このオプションを選択すると、ラッパースキーマを保存する場所を問われます。デフォルトでは、ラッパースキーマは somefile-wrapper.xsd. という書式の名前を持ちます。ラッパースキーマを保存した後、デフォルトでコンポーネントに割り当てられ、ダイアログボックスがプロンプトします:

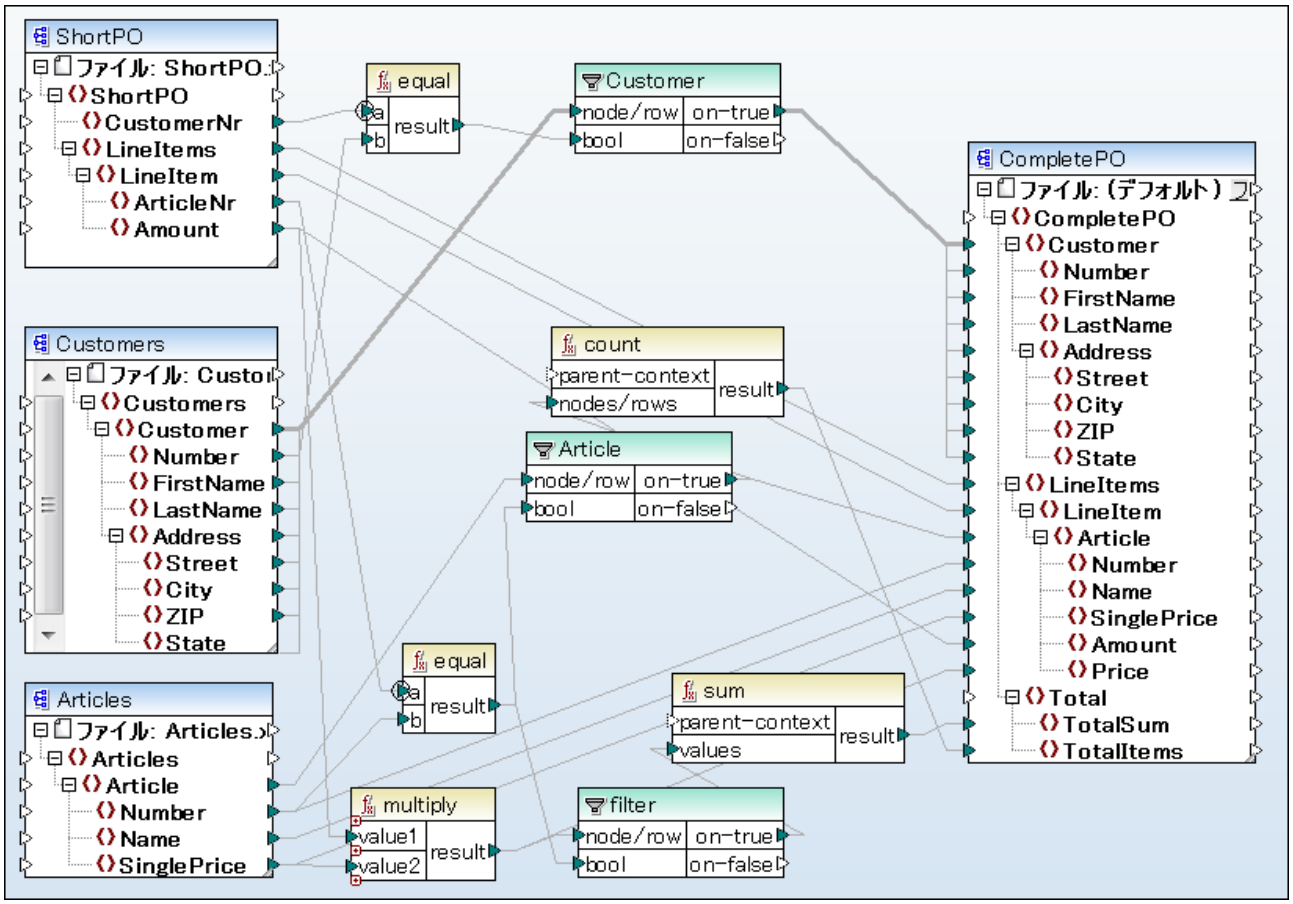


「はい」をクリックして、前のスキーマに戻します。それ以外の場合は、「いいえ」をクリックして新規に作成された、コンポーネントに割り当てられたラッパースキーマを保持するためにいいえをクリックします。

5.1.10 複数のスキーマからのデータのマージ

このサンプルは複数のファイルを単一のファイルにマージする方法について説明しています。具体的には、異なるスキーマを持つ複数のソースコンポーネントをターゲットスキーマにマージします。同一のスキーマをベースにした複数のファイルをマージする方法については、複数の入力または出力ファイルを動的に処理を参照ください。

CompletePO.mfd ファイルは **...\MapForceExamples** フォルダー内にあり、3 つのファイルを 1 つの XML ファイルへマージする方法を表示しています。



単一のターゲット XML ファイル **CompletePO.xml** に複数のソースコンポーネントデータが組み合わされていることにご注意してください。

- **ShortPO** は XML インスタンスファイルに関連付けられたスキーマで、番号と数量などの顧客番号と品物に関するデータのみが含まれています。(このファイルには顧客一人だけが収められています。)
- **Customers** は XML インスタンスファイルに関連付けられたスキーマで、顧客番号と、Name や Address といった顧客に関する詳細情報が収められています。
- **Articles** は XML インスタンスファイルが関連付けられたスキーマで、品物名やその数、単価といった品物に関する情報が収められています。
- **CompletePO** はインスタンスファイルを持たないスキーマファイルです。全てのデータは 3 つの XML インスタンスファイルより与えられています。このファイルの階層構造により、XML データのマージと出力を行うことが可能になります。

このスキーマファイルは XMLSpy のような XML エディターにより作成されており、MapForce から生成されたものではありません（**CompletePO.xml** インスタンスファイルが存在する場合、スキーマファイルの生成を行うことも可能です）。

CompletePO の構造はソースとなる XML ファイルの構造を組み合わせたものとなります。

filter コンポーネント（Customer）を使用することで、ShortPO と Customer XML ファイルで顧客番号が一致するデータを検索することができ、関連するデータをターゲットの CompletePO コンポーネントへ渡すことができます。

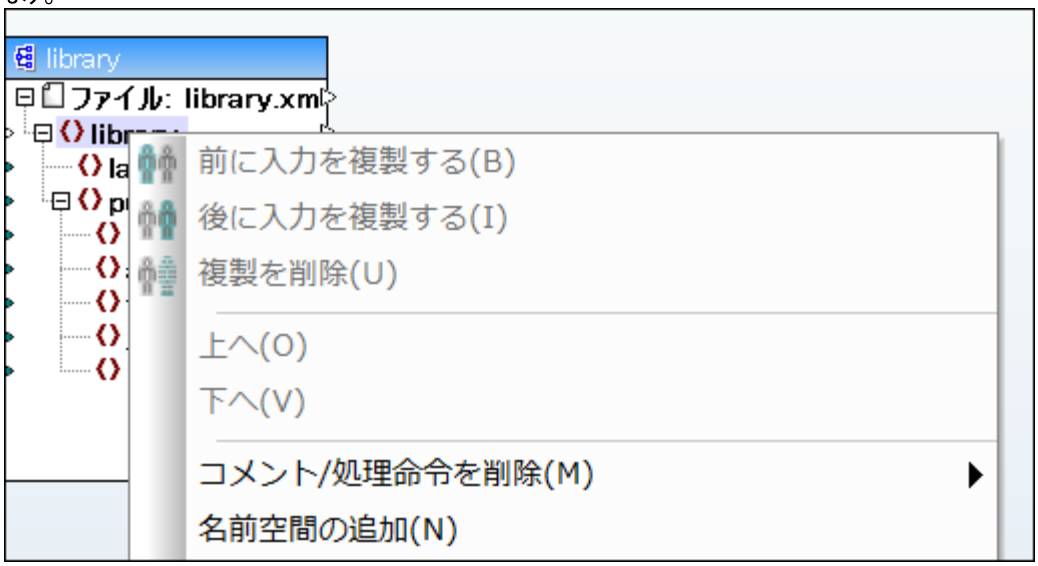
- `equal` 関数により、ShortPO の **CustomerNr** が Customer 以下にある **Number** と比較されます。
- ShortPO には 1 つの顧客情報（番号が 3）しか含まれていないため、顧客番号が 3 となっている顧客の情報だけがフィルターコンポーネントにお渡されます。
- フィルターコンポーネントの **node/row** パラメーターは **bool** パラメーターが true の時に Customer データを "on-true" ノードへ渡します。つまり、顧客番号が 3 の時だけ出力が行われます。
- 顧客と品物データの残りは、他にある 2 つのフィルターコンポーネントによりターゲットスキーマへパスされます。

5.1.11 カスタム名前空間の宣言

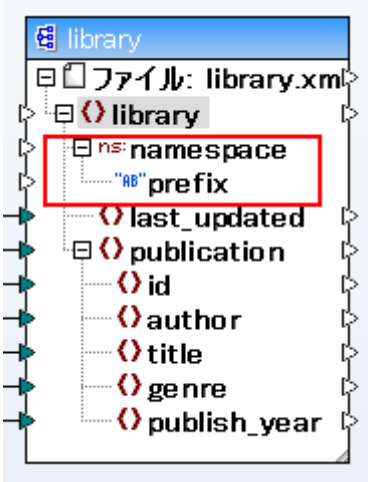
デフォルトでは、マッピングが XML 出力を生成する際に、各要素と属性の名前空間（または、名前空間のセット）が自動的に MapForce によりターゲットコンポーネントと関連するスキーマから生成されます。これは、MapForce 内のデフォルトの振る舞いで、XML 出力の生成を含むマッピングのシナリオの大半に適切です。

しかしながら、結果の XML 出力内で要素の名前空間の管理を更に行う場合、マッピングから要素の名前空間を手動で宣言することが必要な可能性があります。

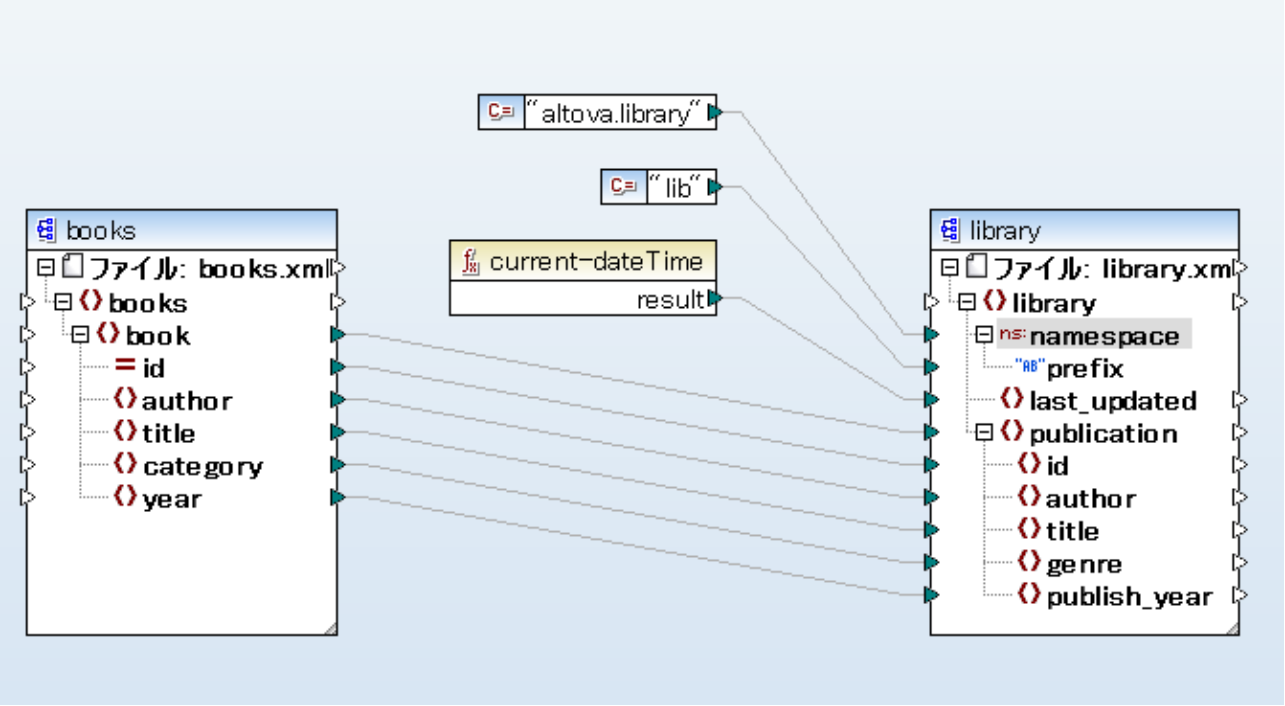
この仕組みを理解するには、＜マイドキュメント＞¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥ 内にある **BooksToLibrary.mfd** マッピングを開きます。`library` ノードを右クリックし、コンテキストメニューから「名前空間の追加」を選択します。



`library` ノードの下で次の 2 つの新しいノードが使用することができるようになっていることにご注意してください： `namespace` と `prefix`.



マッピングからの文字列の値をマップすることができます。下のイメージでは、名前空間「altova.library」とprefix「lib」を与える、2つの定数が「**挿入 | 定数**」メニューコマンドを使用して定義されています:



生成された出力内の結果としては、`xmlns:<prefix>="<namespace>"` 属性が、`<prefix>` と`<namespace>` がマッピング(この場合は定数)から来る値である箇所の要素に追加されます。生成された出力は、以下のようになります(ハイライトされている部分に注意してください):

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns:lib="altova.library" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="library.xsd">
...
```

メモ カスタム名前空間の宣言(と「**名前空間の追加**」コマンド)は、ターゲット XML コンポーネントに取り有益であり、要素のみに適用することができます。属性に対して、「**名前空間の追加**」コマンドは、属性とワイルドカードノードに対しては使用することが

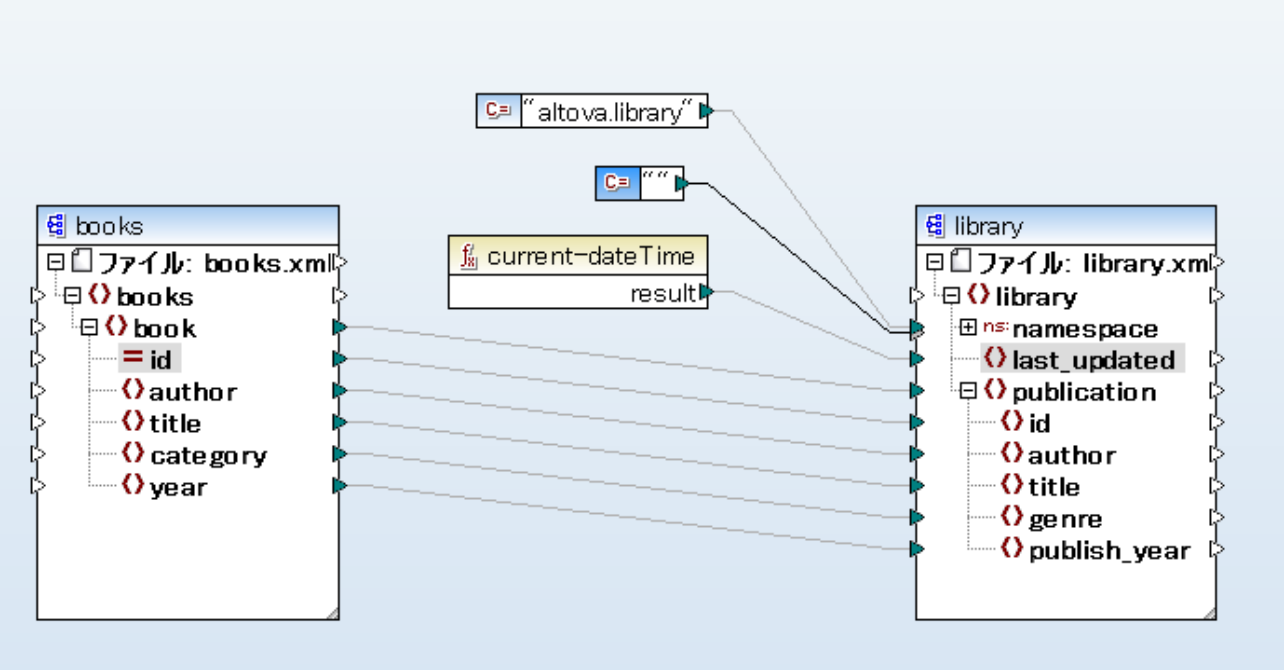
できません。ノードの子への動的なアクセスが有効化されている場合でも、または「全てをコピー」接続を使用してノードがデータを受け取るノードの場合でも使用することはできません（ノード名のマッピングを参照してください）。

必要に応じて、複数の名前空間を同じ要素のために宣言することができます。これを行うには、ノードをもう一度右クリックし、コンテキストメニューから「**名前空間の追加**」を選択します。新規のプレフィックスと名前空間の値に接続することのできる名前空間とプレフィックスノードの新規のペアを使用できるようになります。

以前に追加された名前空間の宣言を削除するには、 `ns:namespace` ノードを右クリックし、コンテキストメニューから「**名前空間の削除**」を選択します。

> 空の値が与えられる場合でも、`namespace` と `prefix` 入力コネクタの両方がマップされている必要があります。

（`xmlns="mydefaultnamespace"` のフォーマットの）デフォルトの名前空間を宣言する場合、`prefix` に対して空の文字列をマップします。このケースをマップ上で確認する場合は、2番目の定数が空の文字列になるように上のサンプルマッピングを編集します。



結果の出力は以下のようになります:

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="altova.library" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="library.xsd">
...
```

属性の名前のためにプレフィックスを作成する場合、例えば、`<number prod:id="prod557">557</number>` 、ノードの属性への動的なアクセスを有効化する、または`<number>` のために`prod:id` 属性を持つようにスキーマを編集することにより属性の名前のためにプレフィックスを作成することができます（次を参照してください: ノード名のマッピング）。

6 関数

MapForce 内では必要に応じてデータを変換するために以下の関数のカテゴリを使用することができます:

- **MapForce ビルドイン関数** – MapForce 内でこれらの関数は定義済みで、これらを文字列、数値、日付、および他のデータ型を扱う広範囲の処理を行い、マッピング内で使用することができます。グループ分け、集計、自動付番、および他の多種のタスクを行うために使用することができます。使用することのできるすべてのビルドイン関数への参照に関しては、関数ライブラリの参照を参照してください。
- **ユーザー定義関数** (UDF) – これらはMapForce 内で使用が可能なネイティブなコンポーネント型とビルドイン関数のベースとする、自身で作成が可能な MapForce 関数です。ユーザー定義関数を参照してください。
- **カスタム関数** – これらはXSLT ライブラリなどの外部ソースからインポートし MapForce に適用することが可能な関数です。MapForce 内で再利用可能にするには、カスタム関数は(文字列または整数) 単純型のデータを返し、単純型のパラメーターを取る必要があります。詳細に関しては カスタムXSLT 関数のインポート

以下のロードマップを使用して 特定のタスクに関連した関数にアクセスしてください:

...を実行するには	以下を参照してください...
MapForce 関数、または定数をマッピングに追加する方法	• マッピングに関数を追加する • マッピングに定数を追加する • 関数の検索 • 関数の型と詳細を確認する • 関数引数の追加、または削除
MapForce 内で自身の関数を作成し、同じマッピングまたは他のマッピングで後に使用する方法	• ユーザー定義関数
カスタムXSLT 関数を MapForce にインポートする方法	• カスタムXSLT 関数のインポート

6.1 使用方法

6.1.1 マッピングにビルトイン関数を追加する

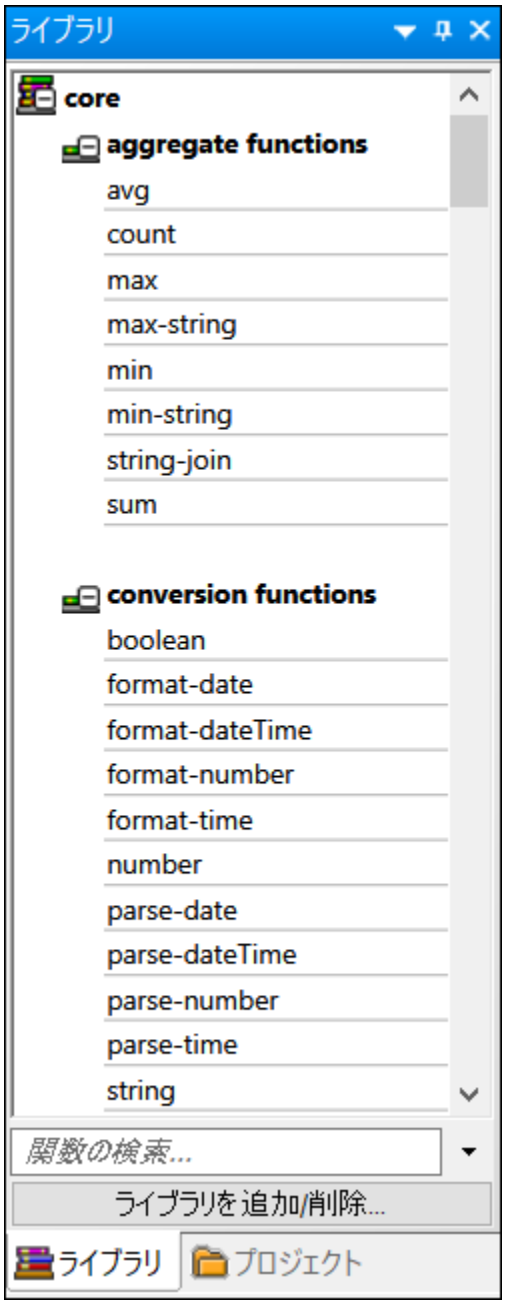
MapForce には下記の通り、マッピングに追加することのできるすぐに使用可能な多数のビルドイン関数が含まれています。使用することのできるすべてのビルドイン関数への参照に関しては、関数ライブラリの参照 を参照してください。

以下を前提条件として下で説明されるアプローチと同様ユーザー定義関数（UDF）をマッピングに追加することができます:

- 同じマッピング内でUDF が既に作成されている、または
- UDF を含むマッピングをローカルまたはグローバルライブラリとして既にインポートしている。

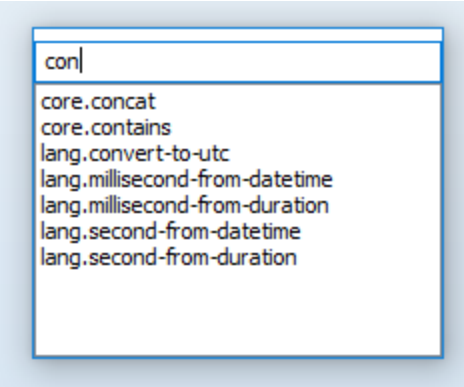
マッピング内で関数を使用する:

1. 変換言語を選択します。ライブラリウィンドウ内の必要な関数をクリックし、マッピングエリアにドラッグします。
2. 名前別に関数をフィルターするには、ウィンドウの下の部分内にあるテキストボックス内に関数名を入力します: ライブラリウィンドウの下の部分内にあるテキストボックス内で関数名の入力を開始します。



代わりに、定数を次のように追加することができます:

1. マッピングの空白のエリアをダブルクリックし、関数名を開始します。入力されるテキストによりフィルターされるライブラリウィンドウと同じ関数を持つコンボボックスが表示されます。各関数に関する詳細を表示するヒントを確認するには、リスト内から関数を選択してください。



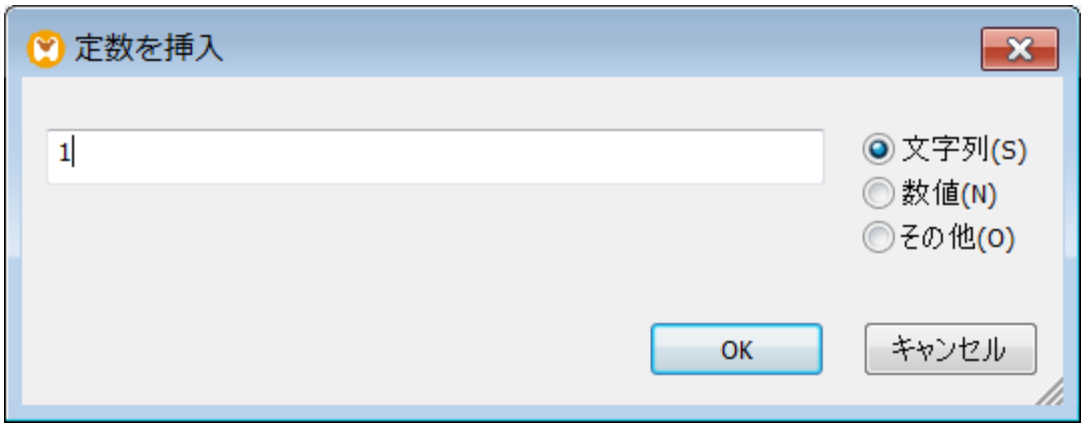
2. 必要とされる関数を選択し、マッピングに追加するために **Enter** を押します。関数を選択することなく、コンボボックスを閉じるには、**Escape** を押し、ボックス外のエリアをクリックします。

6.1.2 マッピングに定数を追加する

定数を使用することによりマッピングにカスタムのテキスト、または数値を与えることができます。定数の値は、その名前が示すように、マッピングのライフタイムの期間中、変化せず同じままの状態でいます。

マッピングに定数を追加する方法:

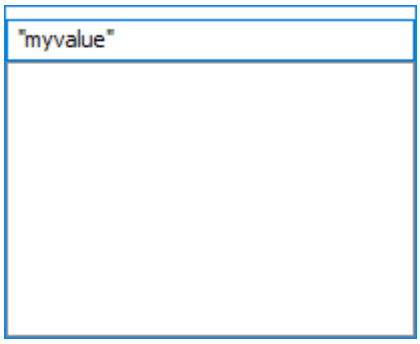
1. 以下のいずれかを行います:
   a. 「**挿入**」メニューから、「**定数**」をクリックします。
   b. マッピングを右クリックし、コンテキストメニューから「**定数を挿入する**」を選択します。



2. 定数の値を入力し、データ型(「文字列」、「数値」、「その他」)を選択し、「**OK**」をクリックします。

代わりに、定数を次のように追加することができます:

1. 空のマッピングエリア内をダブルクリックします。
2. 以下のいずれかを行います:
   a. 文字列の定数を追加するには、二重引用符の後に定数を入力します。終わりの二重引用符は任意です。



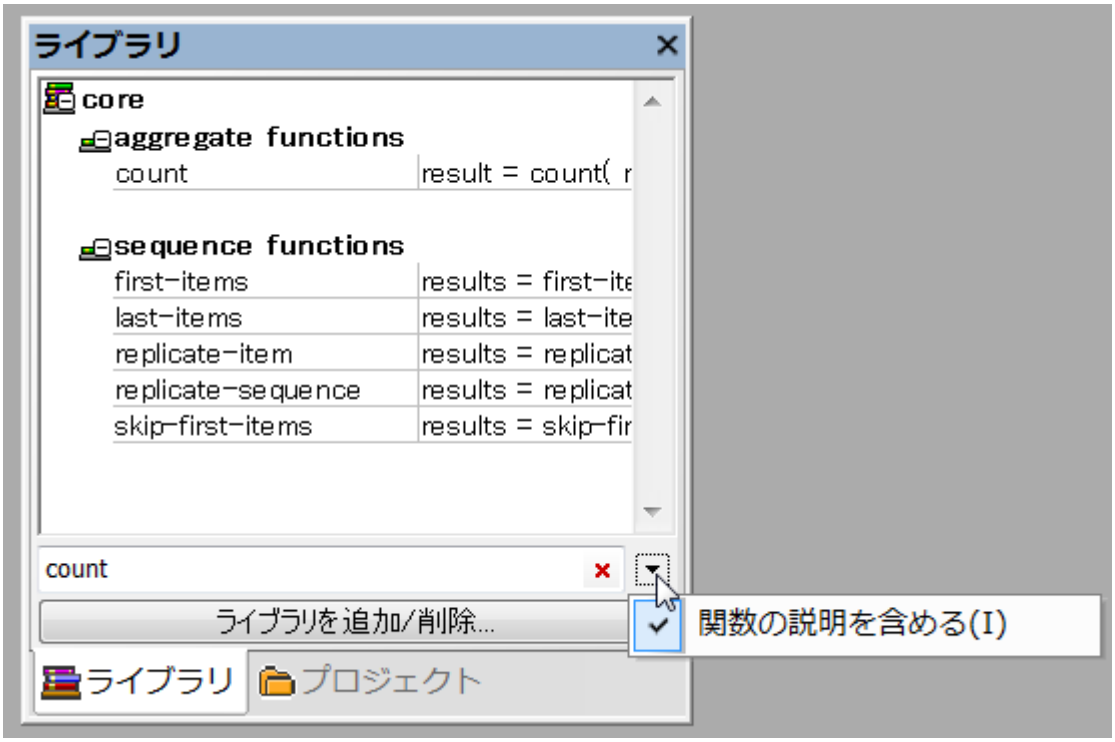
b. 数値の定数を追加するには、数値を入力します。

3. 「Enter」を押します。

6.1.3 関数の検索

ライブラリウィンドウ内で関数を検索するには、ライブラリ ウィンドウの下の部分内にあるテキスト ボックス内で関数名の入力を開始します。

デフォルトでは、MapForce は関数名、または説明テキストにより検索することができます。検索から関数の説明を除外するには、下向きの矢印をクリックし、「関数の説明を含む」オプションを無効化します。



検索をキャンセルするには、「Esc」キーを押す、または「 × 」をクリックします。

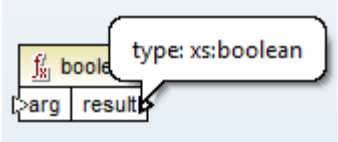
ライブラリ ウィンドウ内で使用することのできる関数は、現在選択されている変換言語によりことなります。次を参照してください: 変換言語の選択。

現在アクティブなマッピング内での関数のすべての発生を検索するには、ライブラリ ウィンドウ内の関数名を右クリックし、「全ての呼び出しを検索する」をコンテキスト メニューから選択します。検索結果 がメッセージ ウィンドウ内に表示されます。

6.1.4 関数の型と詳細を確認する

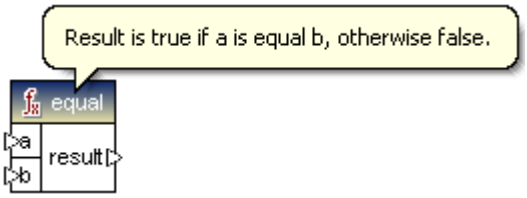
入力、または出力の引数のデータ型を確認する

1. 「ヒントの表示」 ツール バーボタンが有効化されていることを確認してください。
2. 関数の引数の部分にマウスを移動させ、ポイントします。



関数の説明を確認する:

1. 「**ヒントの表示**」 ツールバーボタンが有効化されていることを確認してください。
2. 関数のマウスを移動します（マッピングエリア上のライブラリペイン内でこれらの機能を使用することができます）。

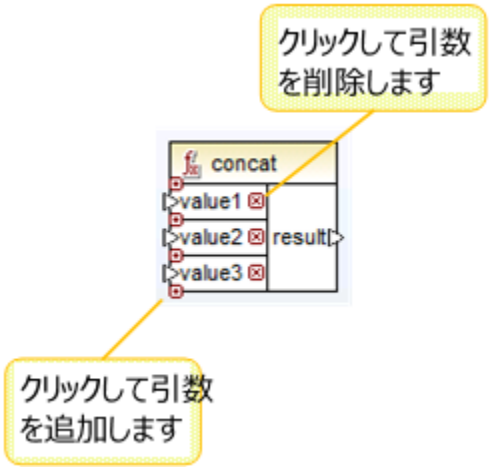


6.1.5 関数引数の追加、または、削除

コンテキストにより必要とされる通り必要な数量の**パラメーター**を追加できるという意味で MapForce ビルトイン関数の一部は拡張可能です。このような関数のよい例は全ての文字列を連結を必要とする `concat` です。

（このような振る舞いをサポートする関数のための）関数の引数の追加、または削除方法：

- **パラメーターの横の**パラメーターの追加（ ）または**パラメーターの削除**（ ）を使用して、パラメーターを追加、または削除します。



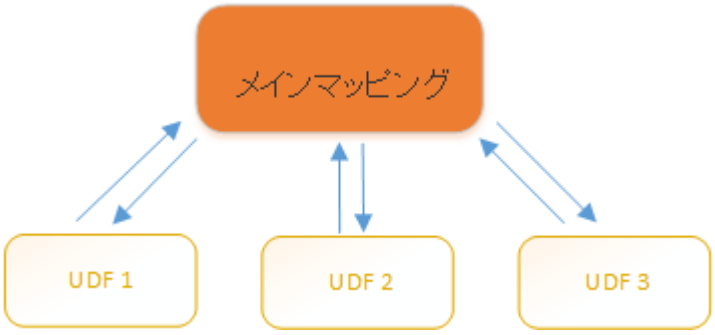
シンボル上に接続をドロップすると、パラメーターが自動的に追加され、接続されます。

6.2 ユーザー定義関数

ユーザー定義関数（UDF）は一度定義されるカスタム関数で、同じマッピング、または複数のマッピング内で複数回使用することができます。ユーザー定義関数自身は小さなマッピングと類似しています。通常、一つまたは複数の入力パラメーター、データを処理するための中間パラメーター、および、と呼び出し元にデータを返すための出力により構成されています。呼び出し元はメインマッピング、または他のユーザー定義関数であることができます。

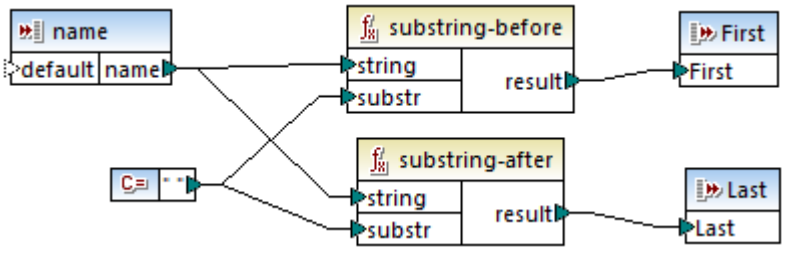
ヒント: 複数の出力を返すユーザー定義関数を作成することは可能です。関数が「インライン」と定義されているとこの機能はサポートされます。インラインと正規ユーザー定義関数を参照してください。

再使用の可能性とは別に、マッピングの部分を小さいコンポーネントにパッケージ、および、実装の詳細を無視し、結果、マッピングを簡単に理解するためにユーザー定義関数は役に立ちます。



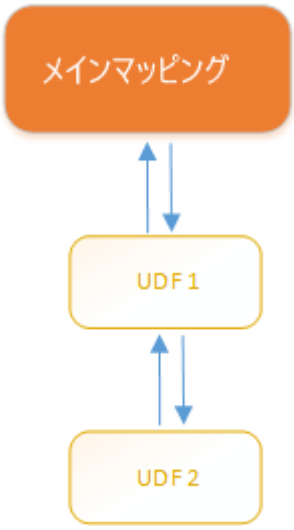
通常、文字列、数値、日付、および、他のデータを内蔵のMapForce 関数を拡張しカスタム化して処理するためにユーザー定義関数を作成します。例えば、特定の方法で結合、または分割する、または高度な計算を実行、時刻を操作、再利用できるようにマッピングの一部をパッケージするなどの場合が挙げられます。ユーザー定義関数の他の一般的な使用方法は、MapForce によりサポートされる他の XML ファイル内のフィールドを検索することです。

以下のマッピングは、文字列を2つの異なる文字列に分割する、ユーザー定義関数のサンプルです。このユーザー定義関数は次のデモマッピングの一部です: <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥**ContactsFromPO.mfd**。パラメーター（例えば、"Helen Smith"）としての名前を取り、
ビルトイン関数 `substring-before` と `substring-after` を適用し、2つの結果する値（"Helen" と "Smith"）を返します。

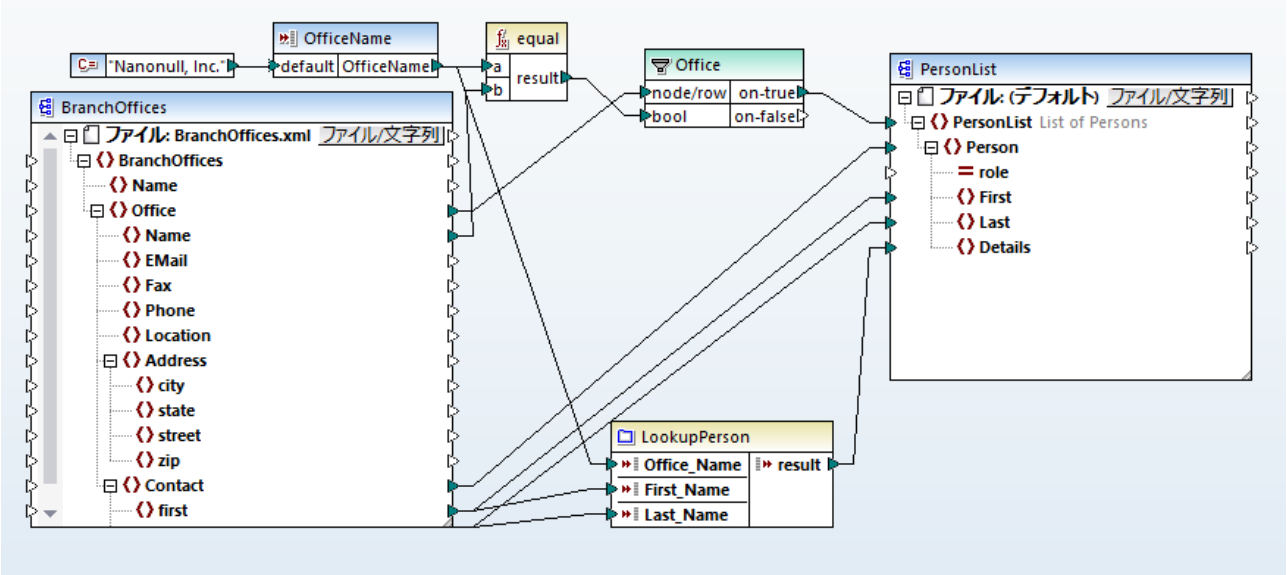


ContactsFromPO.mfd

上記のとおり、ユーザー定義関数をメインのマッピング、またはユーザー定義関数から呼び出すことができます。すなわち、ユーザー定義関数を必要であれば、以下に示されるように、ネストすることができます。

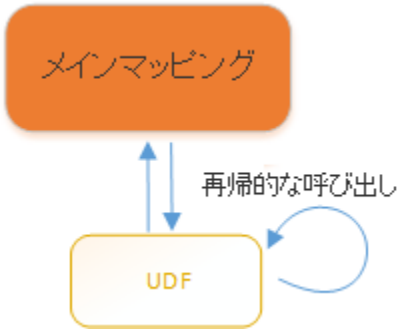


例えば、下のマッピングはユーザー定義関数（「LookupPerson」）を呼び出し、XML ファイル内の個人の名前を検索します。「LookupPerson」コンポーネントのヘッダーをダブルクリックすると、マッピングウィンドウ内の定義が開かれ、この関数は次のユーザー定義関数を呼び出していることに気が付くはずです: "EqualAnd" と "Person2Details"。このマッピングはデモとして使用することができ、次のパスで見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\PersonListByBranchOffice.mfd**.



PersonListByBranchOffice.mfd

ユーザー定義関数を再帰的に呼び出すことができます（すなわち、ユーザー定義関数が自身を呼び出します）。これは、ユーザー定義関数を（インラインではなく）関数正規関数として定義する必要があることを意味します。インラインと正規ユーザー定義関数を参照してください。



再帰的なユーザー定義関数により、*N* が事前に既知ではなく、*N* 個の子の深さを持つデータの構想を反復するなど、多種の高度なマッピングの必要条件を解決することができます。例: 再帰的な検索を参照してください。

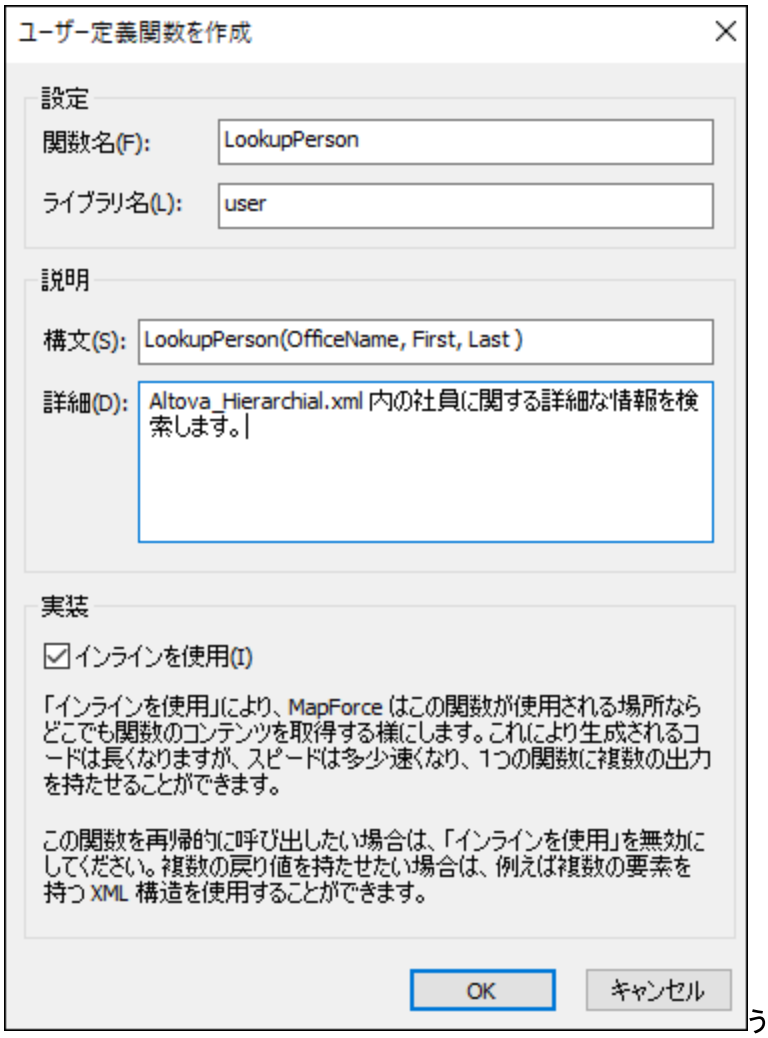
ユーザー定義関数の作成後、作成した同じマッピング内に保存されます。しかしながら、他のマッピングにインポートし、そこから呼び出すこともできます。詳細に関しては、ユーザー定義関数の呼び出しとインポートを参照してください。

6.2.1 ユーザー定義関数の作成

ユーザー定義関数の作成を始めから作成、またはマッピング上に既存のコンポーネントの選択から作成することができます。

ユーザー定義関数を最初から作成する方法:

1. 「関数」メニューから、「**ユーザー定義関数の作成**」をクリックします。または、代わりに、「**ユーザー定義関数の作成**」( ) ツールバーボタンをクリックします。

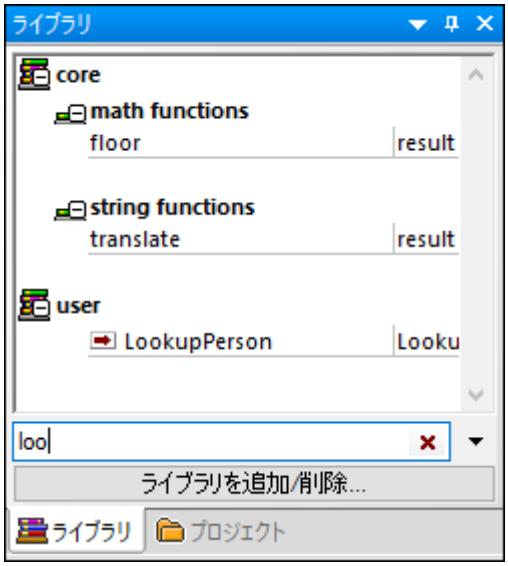


2. 必要なフィールドに情報を挿入します（下のテーブルを参照してください）。

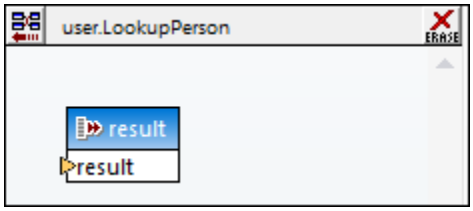
関数名	必須のフィールド。作成するユーザー定義関数のために名前を入力します。有効な文字: • 英数字文字（a-z、A-Z、0-9） • 下線（ _ ） • ハイフン/点線（ - ） • コロン（ : ）
ライブラリ名	必須のフィールド。関数が属するべきライブラリ名を入力します。関数はライブラリウィンドウ内でこのライブラリ名で表示されます。ライブラリを指定しない場合、関数は "user" という名前のデフォルトのライブラリ内に配置されます。
構文	任意のフィールド。関数の構文を的確に詳細するテキストを入力します。（例えば、期待されるパラメーター）。このテキストは、ライブラリウィンドウ内の関数の横に表示され、関数の実行には影響を与えません。

詳細	任意のフィールド。関数の任意のテキストの詳細を入力してください。このテキストは、ライブラリウィンドウ内で、または他のコンテキスト内でカーソルを動かすと表示されます。
インラインされた使用	関数がインライン関数として作成される場合、このチェックボックスを選択してください。正規関数を作成する場合は、チェックを解除してください。詳細に関しては、インラインと正規ユーザー定義関数を参照してください。

3. 「OK」をクリックします。上記で指定されているライブラリ名の下のライブラリウィンドウ内で関数がすぐに表示されます。例:



また、マッピングウィンドウが再度描き直され、新規の関数を作成することができます(これは"関数のマッピング"として参照されるスタンドアロンマッピングです)。関数は出力を必要とするため、関数のマッピングにはデフォルトで出力コンポーネントが含まれています。



左上の角の「**メインマッピングに戻る**」( ) ボタンにより、関数のマッピングからメインマッピングに戻ることができます。関数のマッピングを開くためには、ライブラリウィンドウ内の関数をダブルクリックします。詳細に関しては、ユーザー定義関数の呼び出しとインポートとユーザー定義関数のナビゲートを参照してください。

4. 関数の定義により必要とされるすべてのコンポーネントを関数のマッピングに追加します。標準のマッピングと同じ方法でこれを行うことができます。例えば、入力または出力パラメーターを追加するには、次のいずれかを実行してください:

- メニューコマンド「**関数 | 入力の挿入**」、または「**関数 | 出力の挿入**」をそれぞれ実行します。
- マッピングエリアを右クリックし、「**入力の挿入**」または「**出力の挿入**」をコンテキストメニューから選択します。
- 「**入力の挿入**」( ) または「**出力の挿入**」( ) ツールバーボタンをクリックします。

少なくとも、関数はデータが接続される1つの出力コンポーネントを必要とします。入力パラメーターに関しては、関数は、ゼロ、または1つ以上の入力を持つことができます。入力または出力パラメーターは、単純型（文字列、または整数など）、または複合型（構成）であることができます。単純型、および、複合型のパラメーターに関する詳細は、ユーザー定義関数内のパラメーターを参照してください。

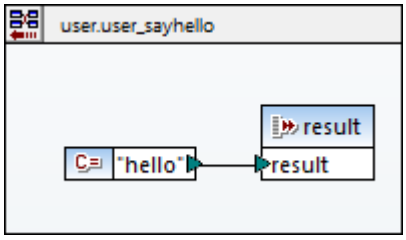
ユーザー定義関数が作成されましたが、まだ使用されていません。関数をマッピング内で使用するには、ライブラリウィンドウから関数をメインマッピング エリアにドラッグします。 ユーザー定義関数の呼び出しとインポートを参照してください。

ユーザー定義関数を既存のコンポーネントから作成する方法:

1. マウスを使用して正方形を作成し、マッピング上で複数のコンポーネントを選択します。「Ctrl」 キーを押したまま、それぞれをクリックして複数のコンポーネントを選択することができます。
2. 「関数」メニューから、「選択からユーザー定義関数を作成する」をクリックします。代わりに、「選択からユーザー定義関数を作成する」（ ）ツールバーボタンをクリックします。
3. 上記のステップ 2-4 を繰り返します。

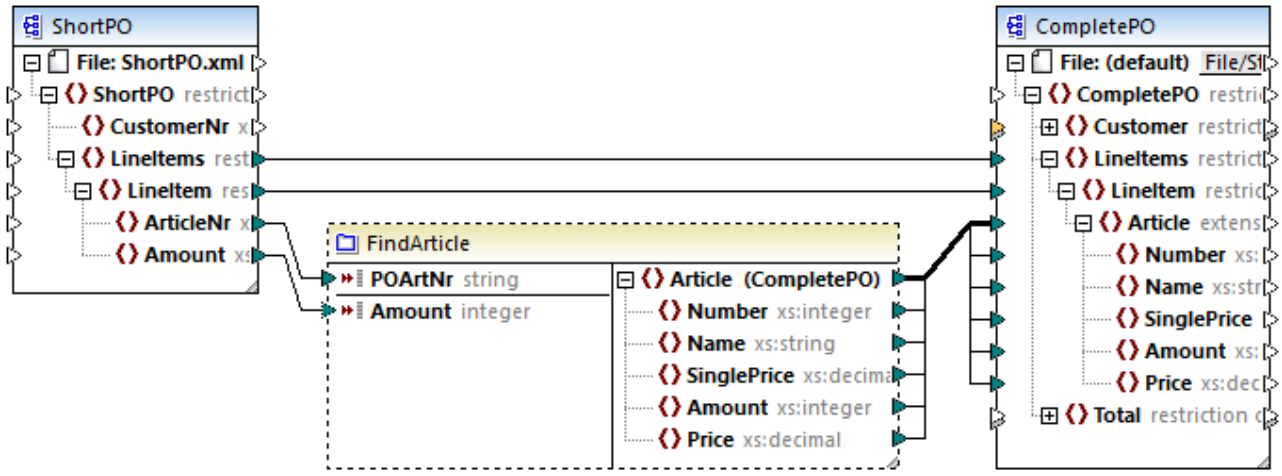
6.2.2 ユーザー定義関数内のパラメーター

ユーザー定義関数を作成する場合、（必要な場合）どの入力パラメーターが取られるか、および、出力が返されるかを指定する必要があります。入力パラメーターが必要とされない場合がありますが、出力パラメーターはすべての場合に必要とされます（すなわち、関数は常に何かを返す必要があります）。例えば、下の関数には入力は存在しませんが、呼出元にテキスト「hello」を返す1つの出力が存在します:



関数パラメーターは、単純型（文字列、または整数などの）または複合型の構成です。例えば、下に表示されるユーザー定義関数 "FindArticle" では、2つの入力と1つの出力パラメーターが存在します。

- **POArtNr** は単純型 "string" の入力パラメーターです。
- **Amount** は単純型 "integer" の入力パラメーターです。
- **CompletePO** は複合型 XML の出力パラメーターです。



LookupArticle.mfd

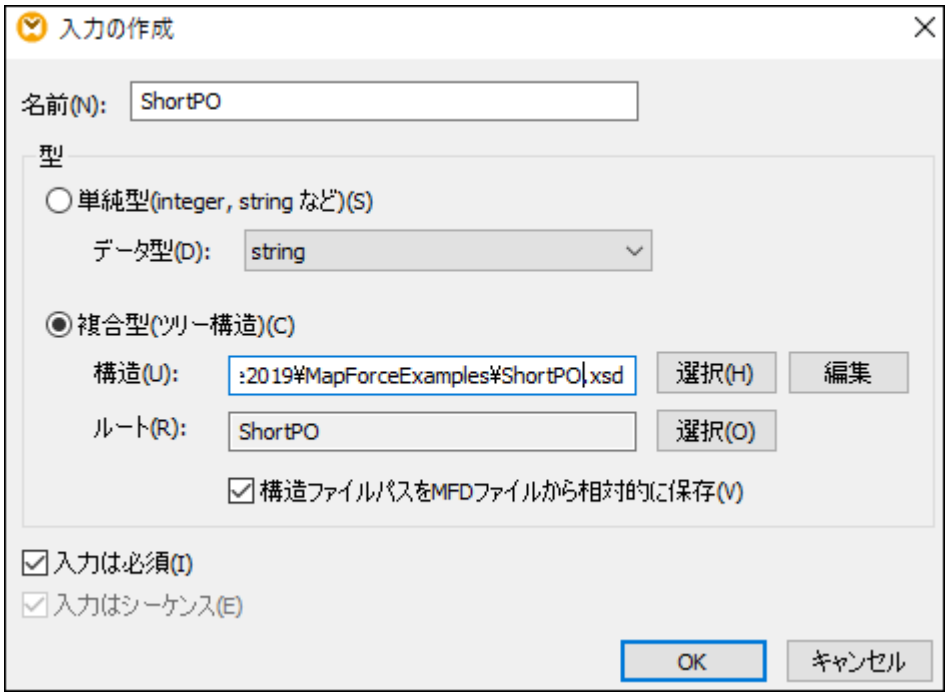
この上のマッピングはデモとして次のパスで見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\LookupArticle.mfd**。

パラメーターの追加

入力または出力パラメーターを追加する方法:

1. ユーザー定義関数 マッピングを作成、または既存のものを開きます（ ユーザー定義関数の作成を参照してください。）または、（ ユーザー定義関数の編集を参照してください）。
2. 次のいずれかを実行してください:

   - メニューコマンド「**関数 | 入力の挿入**」または「**関数 | 出力の挿入**」を実行します。
   - **入力の挿入** または **出力の挿入** ツールバーボタンをクリックします。



3. 上のダイアログボックスで、入力または出力パラメーターが文字列、または整数などの単純型（XML 構成などの）複合型の構成であるかを選択してください。複合型の XML 型であるパラメーターを作成するためには、「構造」の横の「選択」をクリックし、必要とされた構成を説明する XML スキーマを参照してください。

関数のマッピングは XML スキーマを既に含む場合、構成として選択のために使用することができます。それ以外の場合、パラメーターの構成を適用する新規のスキーマを選択することができます。

XML 構成を使用すると、XML スキーマが許可する場合、構成のためにルート要素を選択することができます。ルート要素を指定するためには、「Root」の横の「選択」し、開かれるダイアログボックスからルート要素を選択します。

選択された場合、チェックボックス「MFD ファイルから相対的に構成ファイルパスを保存」はファイルを保存する際、構成ファイルの絶対パスを現在のマッピングに対して相対的なパスに変更します。詳細に関しては、コンポーネント上で相対パスを使用するを参照してください。

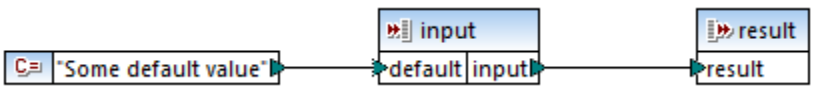
「入力は必須」と「入力はシーケンス」チェックボックスは次のセクションで説明されています。

必須のパラメーター

ユーザー定義関数内でパラメーターを必須にするには、「入力は必須」チェックボックスを選択します。パラメーターが必須の場合、入力が接続されていない場合検証エラーが発生します。

パラメーターを任意にするには、「入力は必須」チェックボックスのチェックを解除します。メインマッピングでは、任意のパラメーターには若干異なる外観（入力コネクタ（小さな三角形）の点線の境界線で表示）が存在します。

パラメーターの「default」入力に接続することにより、デフォルトのパラメーターの値を指定する必要があります。例:

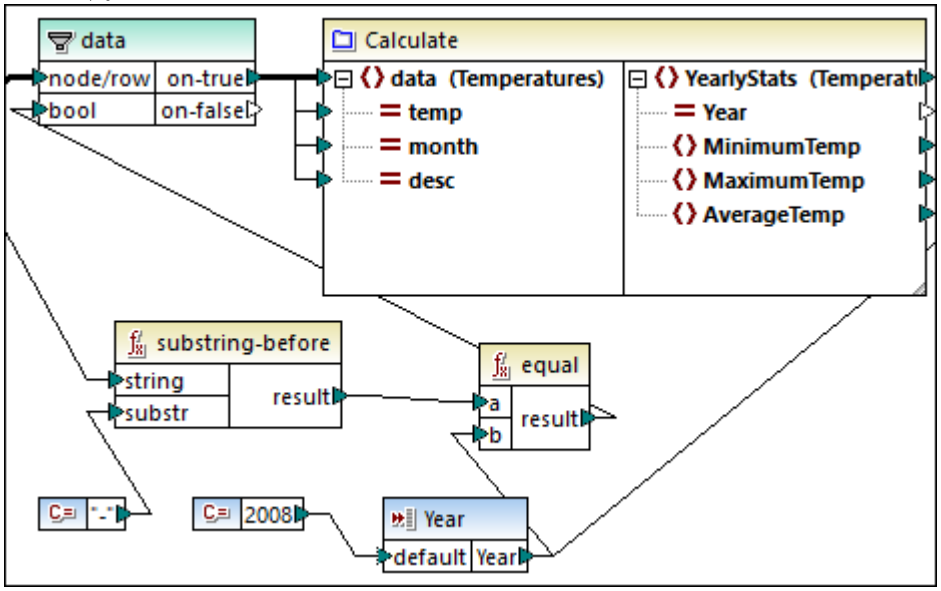


デフォルトの値はその他の値が存在しない場合のみ適用されます。関数が呼び出される際に、任意のパラメーターが値を受け取ると、この値はデフォルトの値に対し優先されます。

シーケンスパラメーター

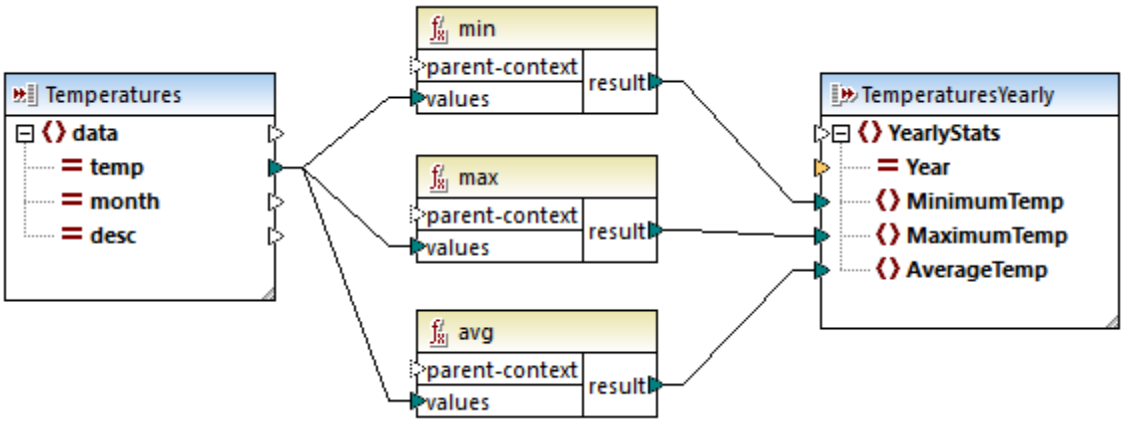
関数のパラメーターが単一の値（デフォルトの振る舞い）、またはシーケンスとして扱われるように任意で指定することができます。パラメータの入力値を単一の値と異なり、シーケンスとして扱うには、「**入力はシーケンス**」チェックボックスを選択します。このチェックボックスは役に立ち、ユーザー定義関数が型「正規」に設定されている場合のみ有効化されます。[インラインと正規ユーザー定義関数](#)を参照してください。それ以外の場合、チェックボックスは無効化されます。

シーケンスはゼロ、またはゼロ以上の値の範囲です。ユーザー定義関数が入力データをシーケンスとして期待する場合、そのシーケンス内で値の集計を行うには、パラメーターの入力値をシーケンスとして扱う必要がある場合があります（例えば、**avg**, **min**, **max** などの関数を呼び出す場合などが挙げられます）。例えば、デモマッピングを開いてください：**<マイドキュメント>\Altova\MapForce2021\MapForceExamples\InputIsSequence.mfd**。このマッピングでは、「data」フィルターはユーザー定義関数「Calculate」に接続されています。フィルターの出力は、アイテムのシーケンスで、関数の入力パラメーターはシーケンスに設定されています。



InputIsSequence.mfd

内部では、「Calculate」関数はすべてのシーケンス値が集計されます（下で示されるように**min**、**max** と**avg** を実行し入力シーケンス上の関数を集計します）。



一般的には、シーケンス、または非シーケンスに関わらず、入力データはどのように関数が呼び出されるかを決定します。

- **sequence** パラメーターに入力データが接続されている場合、ユーザー定義関数1度のみ呼び出され、完全なシーケンスがユーザー定義関数内にパスされます。
- **non-sequence** パラメーターに入力データが接続されている場合、ユーザー定義関数はシーケンス内の各単数のアイテムのためにそれぞれ呼び出されます。

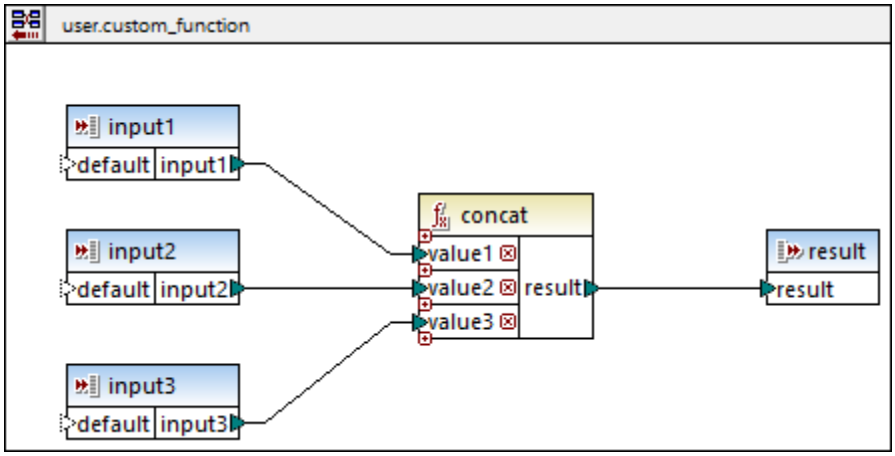
空のシーケンスを非シーケンスパラメーターに接続すると、パラメーターは関数が呼び出されないままの結果を取得します。

これは、ソース構成が任意のアイテムを持つ場合、またはフィルターの条件が一致するアイテムを返さない場合に発生します。これを回避するには、シーケンスが空にならないことを保証するために、関数を入力する前にsubstitute-missing 関数を使用し、パラメーターをシーケンスに設定し、関数内の空のシーケンスのために処理を追加します。

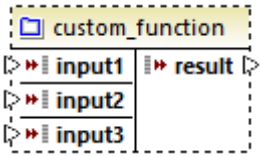
「出力はシーケンス」チェックボックスは、出力パラメータのために必要とされています。関数が複数の値のシーケンスを出力コンポーネントにパスし、出力コンポーネントがシーケンスに設定されていない場合、関数は、シーケンス内の最初のアイテムのみを返します。

パラメーターの順序

ユーザー定義関数が複数の入力または出力パラメーターを持つ場合、この関数の呼出元に表示されるパラメーターの順序を変更することができます。例えば、下の関数には3つのパラメーター（**input1**、**input2**、および**input3**）が存在します。



関数のマッピング内の(上から開始する)パラメーターの順序は、この関数の送信元が表示される順序を指定します:



以下の点に注意してください:

- 入力と出力パラメーターは上から下に位置が並べ替えられます。このため、パラメーター**input3** を関数のマッピングに移動すると、この関数の1パラメーターは最初になります。
- 2つのパラメーターが同じ垂直の位置にある場合、左側のパラメーターが優先されます。
- 2つのパラメーターが同じ位置にある場合、内部コンポーネント ID が自動的に使用されます。

6.2.3 インラインと正規ユーザー定義関数

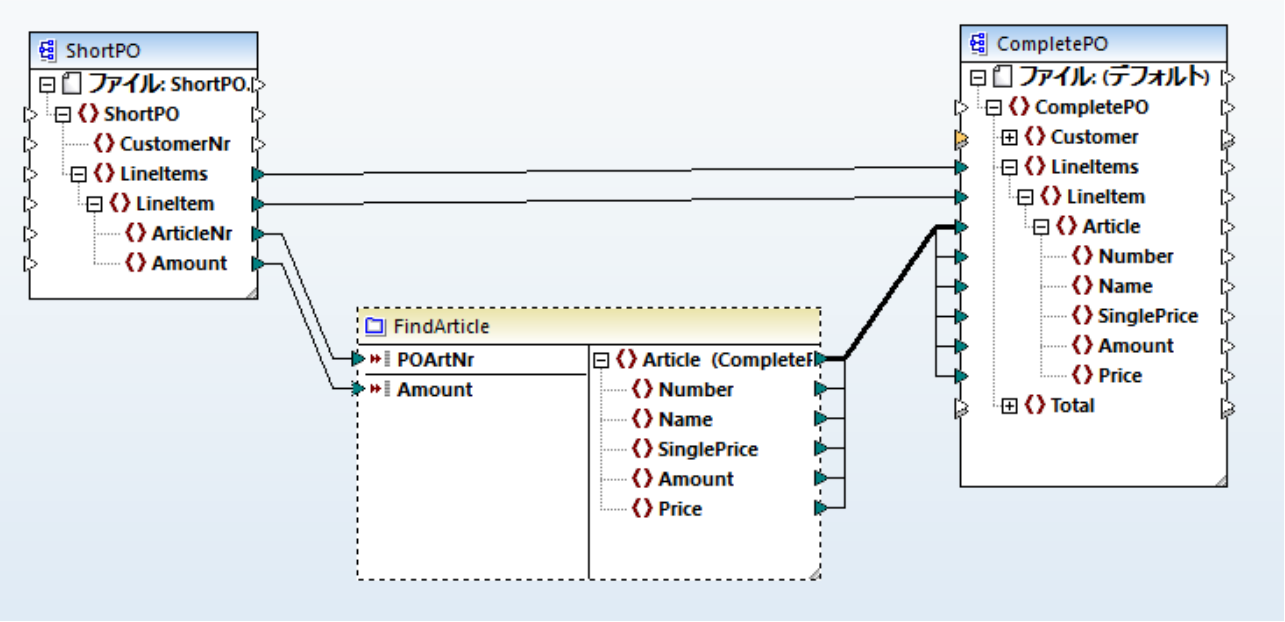
ユーザー定義関数には2つの種類が存在します: インラインと正規関数。関数を作成する際に、関数がインライン、または正規関するであるかを指定することができます。 ユーザー定義関数の作成を参照してください。インラインと正規関数は、コード生成、再帰性、複数のパラメータを持つことができる能力などで異なる振る舞いをします。

インライン関数	正規関数
インライン関数は、生成されたコード内で発生する箇所ですべてのインスタンス内から抽出することができ、コードは長くなりますが、若干速くなります。 インライン関数は、大幅に生成されたプログラムコードを増加することができます。ユーザー定義関数コードは、関数が呼び出されるすべての場所で実際に挿入されます。このため、正規表現とは異なり、コードのサイズが増加します。	各ユーザー定義関数コンポーネントは、入力がパラメーターにパスされる個所で、関数の呼び出しのためにコードを生成し、出力は値を返す関数(コンポーネント) ランタイムでは、すべての入力パラメーターが最初に評価され、関数が入力データの各発生のために呼び出されます。
インライン関数は、複数の出力を持つことができ、このため、複数の値を返します。	正規関数は1つの出力のみを持つことができます。複数の値を返すには、複数の値を発信者にパスすることを許可する出力を複合型(例えば、XML 構成)に宣言することができます。

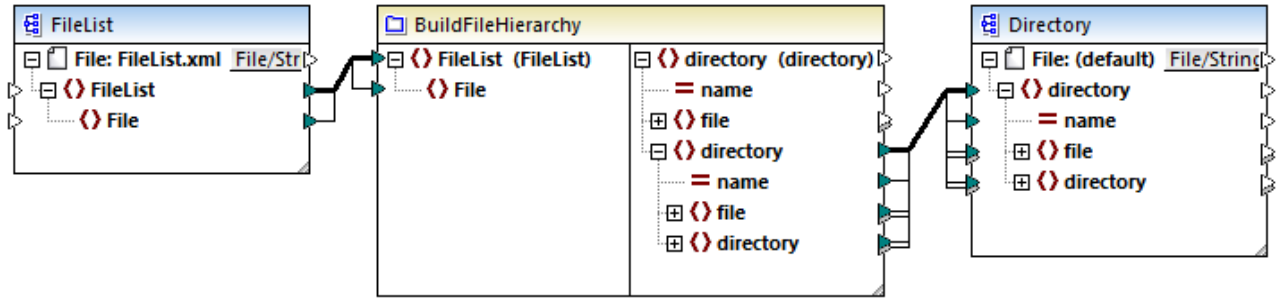
インライン関数	正規関数
インライン関数を再帰的に呼び出すことはできません。	正規関数を再帰的に呼び出すことは可能です。
インライン関数はパラメーター上で優先コンテキスト[優先コンテキスト]を設定することをサポートしません。	正規関数は、パラメーター上で優先コンテキストの設定をサポートします。

ユーザー定義関数を[インライン]から[正規]に切り替える、またはその逆の場合、[マッピングコンテキスト]に影響を与える場合があり、これはマッピングが異なる結果を生成する可能性を引き出します。

マッピング上で、インライン ユーザー定義関数は、点線の囲いにより表示されています。例えば、下のマッピング内の中間コンポーネントはインライン ユーザー定義関数です。



正規関数は、実線の囲いにより表示されています。例えば、下のマッピングの中間コンポーネントは正規のユーザー定義関数です。

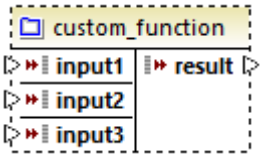


6.2.4 ユーザー定義関数のナビゲート

マッピングにユーザー定義関数が含まれている場合、各ユーザー定義関数とメインマッピングの間を下に示されているように簡単にナビゲートすることができます。

ユーザー定義関数を表示、または編集のために開く方法:

- マッピング上のユーザー定義関数のタイトルバーをダブルクリックします。



- ライブラリウィンドウの特定のユーザー定義関数をダブルクリックします。

ライブラリウィンドウの名前をダブルクリックして関数を編集することができます。現在アクティブなドキュメント内の関数をこの方法で開くことができます。他のマッピング内で作成されたユーザー定義関数をダブルクリックすると、新規のウィンドウ内にそのマッピングを開きます。

メモ 複数のマッピングにインポートされたユーザー定義関数を編集または削除する場合、この変更によりすべてのインポートされるマッピングが影響を受けます。

メインマッピングに戻る方法:

- メインマッピングに戻るには、マッピングウィンドウの左角の「メインマッピングに戻る」( ) ボタンをクリックします。

更に、ユーザー定義関数を含む多種のMapForce タブをナビゲートするとその履歴が保存されます。使用したタブ間を前後に移動するには、「戻る」( ) と「次へ」( ) ツールバーボタンをクリックします。これらのボタンに対応するキーボードのショートカットは、「Alt+Left」と「Alt+Right」です。

6.2.5 ユーザー定義関数の編集

ユーザー定義関数を編集する方法:

1. ユーザー定義関数を含むマッピングを開きます。
2. マッピング上のユーザー定義関数のタイトルバーをダブルクリックします。コンテンツを表示するために、必要に応じてコンポーネントを追加、編集、または削除することのできるマッピングウィンドウが変更されます。
3. (名前または詳細などの) 関数のプロパティを変更するために、以下の一つを行ってください:

   - マッピングの空白の部分を右クリックし、コンテキストメニューから「関数の設定」を選択します。
   - 「ユーザー定義関数設定」( ) ツールバーボタンをクリックします。

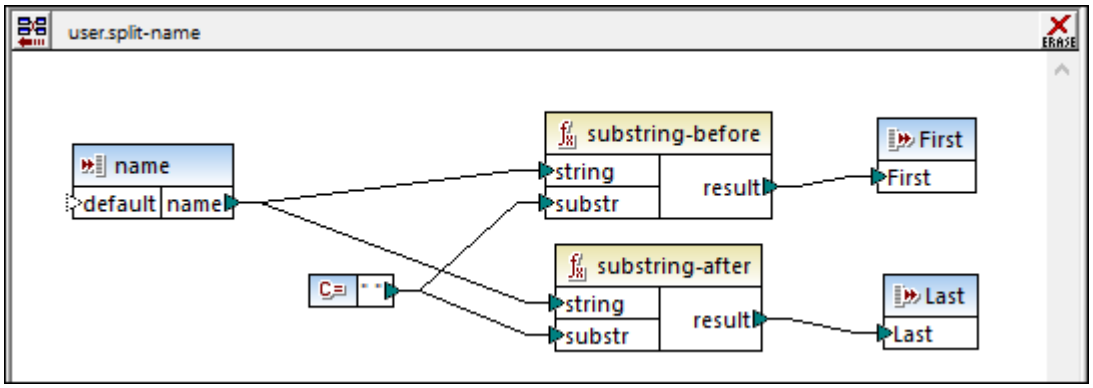
ライブラリウィンドウの名前をダブルクリックして関数を編集することができます。現在アクティブなドキュメント内の関数をこの方法で開くことができます。他のマッピング内で作成されたユーザー定義関数をダブルクリックすると、新規のウィンドウ内にそのマッピングを開きます。

メモ 複数のマッピングにインポートされたユーザー定義関数を編集または削除する場合、この変更によりすべてのインポートされるマッピングが影響を受けます。

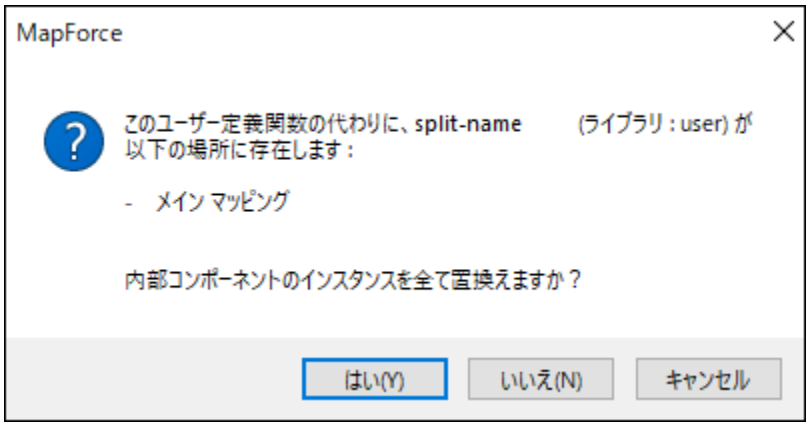
6.2.6 ユーザー定義関数の削除

マッピング上のユーザー定義関数のタイトルバーをダブルクリックしてユーザー定義関数を削除します。

1. マッピングウィンドウの右角の「削除」 ボタンをクリックします。



2. 現在開かれているマッピング内で関数が使用されている場合、ダイアログボックスが表示されます。



関するを削除し、関数のコンポーネントと共に呼び出されているすべてのインスタンスを置き換える場合、「はい」をクリックします。これにより、関数が削除される場合でも、メインマッピングを有効に保つことができます。削除された関数が他の外部のマッピングで使用されている場合、これらは無効になります。

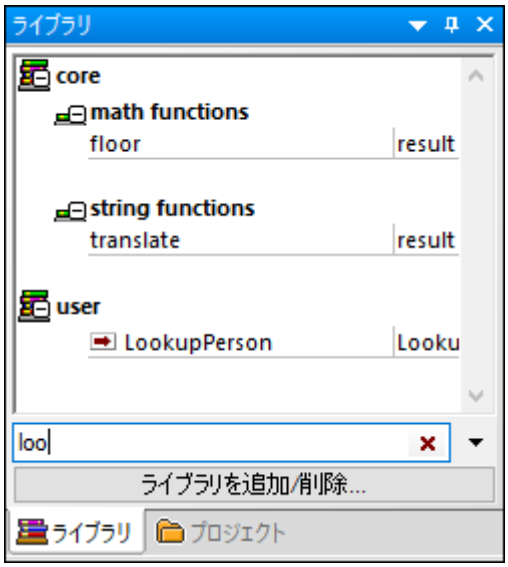
関数とその内部コンポーネントを永久に削除する場合は、「いいえ」をクリックします(この場合、関数が使用されるすべてのマッピングは無効になります)。

6.2.7 ユーザー定義関数の呼び出しとインポート

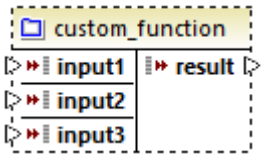
ユーザー定義関数の作成後、作成した同じマッピングから呼び出す、または他のMapForce マッピングから呼び出すことができます。

ユーザー定義関数を同じマッピングから呼び出す方法:

1. ライブラリウィンドウ内で関数を検索します。関数の作成時に指定したライブラリの下に関数が表示されます。デフォルトの「ユーザー」ライブラリを作成した場合、「ユーザー」ライブラリ内で関数を検索してください。名前別に関数を素早く検索するには、ライブラリウィンドウ内に名前を入力します。

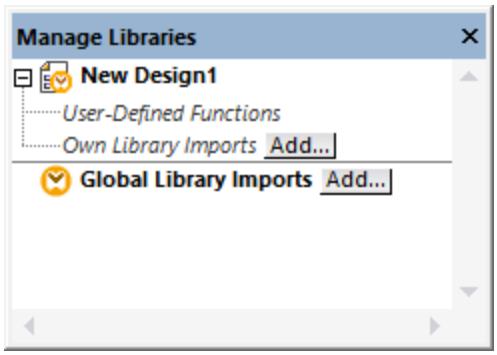


2. ライブラリウィンドウからメインマッピングに関数をドラッグします。必要とされるすべてのパラメーターを接続することができます。関数の結果は出力パラメーター(または、適用可能な場合複数のパラメーター)により適用されます。



ユーザー定義関数を他のマッピングからインポートする方法:

1. ライブラリ ウィンドウのベースの ライブラリの追加/削除 ボタンをクリックします。ライブラリの管理ウィンドウが開かれます。

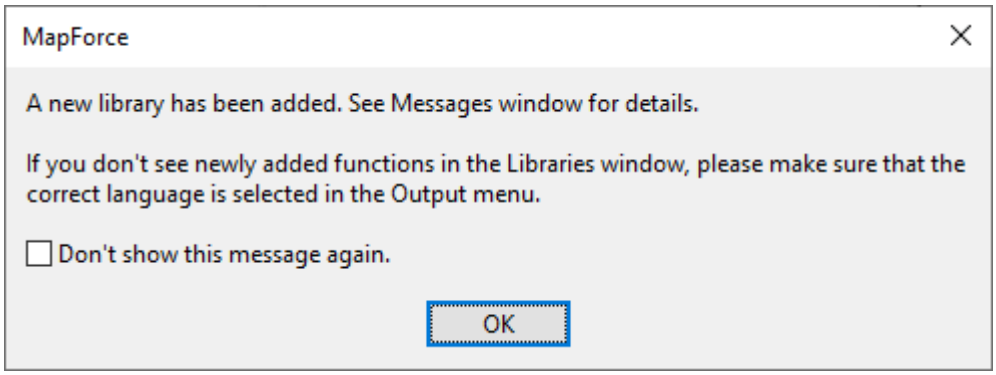


2. 以下の内の1つを行います:

   - (現在のマッピング内のスコープのみで) *ローカル*ライブラリとして関数をインポートする場合現在のマッピング名の下の「**追加**」ボタンをクリックします。
   - (プログラムレベルで) *グローバル*ライブラリとして関数をインポートする場合 **グローバルライブラリのインポート** の横の「**追加**」ボタンをクリックします。

メモ ライブラリを*ローカル*でインポートする場合、ライブラリファイルのパスをマッピングに対して相対的に設定することが可能です。グローバルにインポートされたライブラリでは、インポートされたライブラリへのパスは常に絶対的なパスです。

3. ユーザー定義関数が含まれるマッピングファイル(.mfd) を参照し、「**開く**」をクリックします。新規のライブラリが追加されたことを通知するメッセージボックスが表示され、新規のライブラリがライブラリウィンドウに表示されます。



ライブラリウィンドウからマッピングにドラッグして、現在のマッピング内のインポート済みの関数を使用することができます。関数をマッピングに追加するも参照してください。

ライブラリウィンドウはビルトイン関数と現在のマッピングファイル内のユーザー定義関数を表示します。上に示されるように他の.mfd ファイルをライブラリとして現在のマッピングにインポートすると他のインポート済みのファイルからのユーザー定義関数も表示されます。複数の*.mfd ファイル。

関数ライブラリの表示と整理に関する詳細は関数ライブラリの管理を参照してください。

6.2.8 マッピング間で UDF をコピーし張り付ける用法

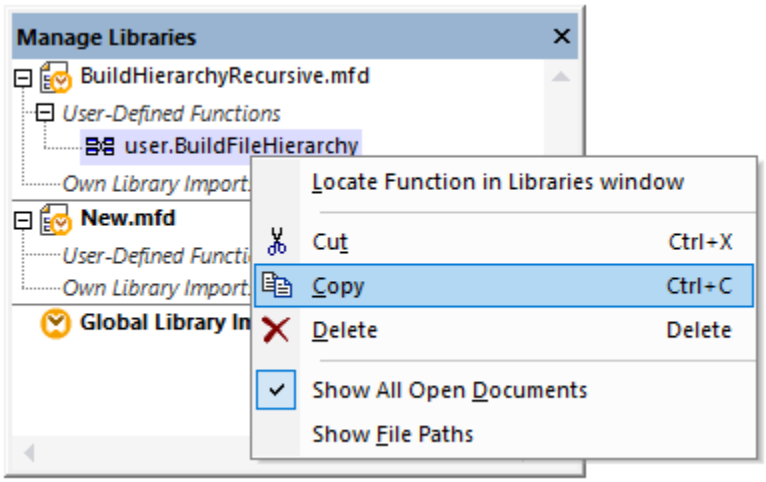
以下のようにマッピング間でユーザー定義関数を簡単にコピーして貼り付けることができます:

1. ライブラリウィンドウのベースのライブラリの追加/削除 をクリックします。ライブラリの管理ウィンドウが開かれます。
2. ライブラリウィンドウ内の空のエリアを右クリックしオプション「**全ての開かれているドキュメントを表示**」を選択します。

3. ソースと目的マッピングを両方開きます。例えば、下のイメージではソースは**BuildHierarchyRecursive.mfd** で、目的のマッピングは**New.mfd** です。

メモ ソースとターゲットマッピングが既にディスクに保存されていることを確認してください。これにより正確なパスが保証されます。相対的なパスのコピーと張り付けを参照してください。

4. ソースマッピングファイルからのユーザー定義関数を右クリックし、コンテキストメニューからコピーを選択します。（または「Ctrl+C」を押します）。

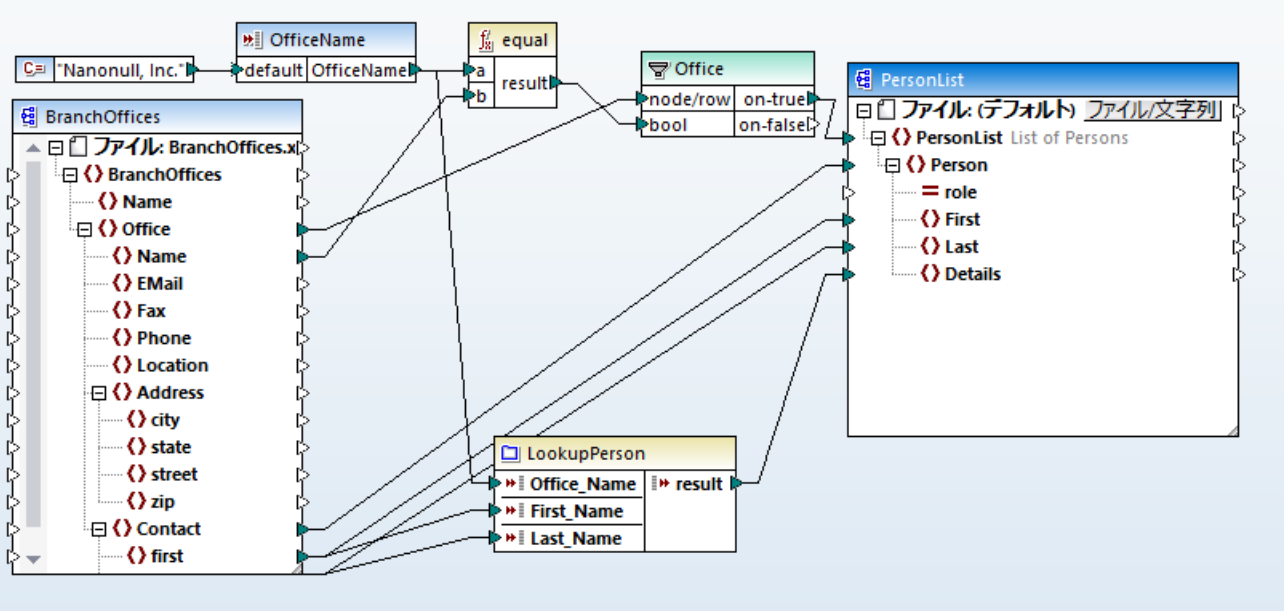


5. ターゲットマッピングファイルの「ユーザー定義関数」エントリを右クリックし、コンテキストメニューから「張り付け」を選択します。

関数ライブラリの管理も参照してください。

6.2.9 例: ルックアップと連結

ユーザー定義関数の一般的な使用方法を説明したいくつかのデモマッピングがMapForce で使用することができます。**PersonListByBranchOffice.mfd** ファイルの1つは<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥フォルダー内で見つけることができます。



PersonListByBranchOffice.mfd

マッピングには以下の必要条件があります:

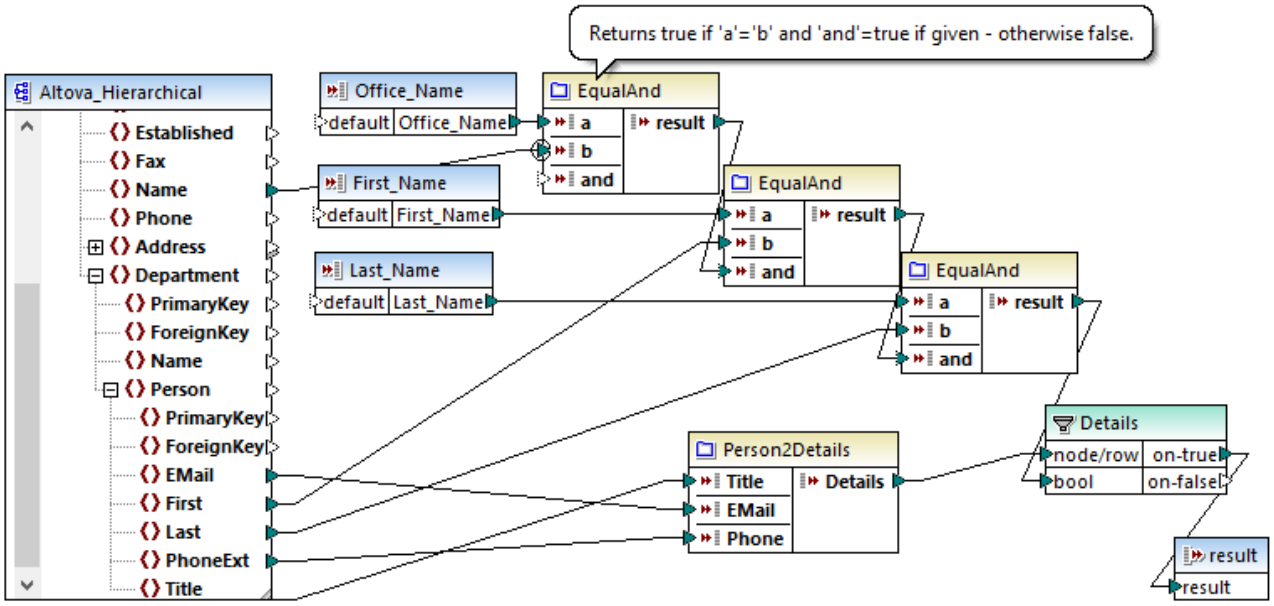
- XML ファイルからデータを抽出し、ターゲット XML ファイルに書き込みます。データは氏名など従業員の詳細から構成されています。
- 個別のXML ファイル(電話、電子メールアドレス、職位) 内で個別の従業員に関するデータを検索します。
- ターゲットに書き込む前に希望する方法でデータを処理します。具体的には、電話番号、電子メール、職位が単一の文字列で表示されており、ターゲット XML の **詳細** 要素に書き込まれています。
- XML 要素、この場合、特定の部署からの従業員に関する情報のみを抽出します。マッピングの発信者はオフィス名をコマンドラインでパラメーターとして指定できる必要があります。例えば、マッピングがMapForce Server により実行されている場合など。

上記の必要条件を実装するコンポーネントを検証してみましょう:

- マッピング("OfficeName") の入力パラメーターはシンプルな入力コンポーネントです。デフォルトの値("Nanonull, Inc.") は定数により提供されます。この値はマッピングの呼出元がパラメーターの値を提供しない場合使用されます。マッピングにパラメーターを提供するを参照してください。
- 特定のオフィスに属する従業員をフィルターするために、マッピングはフィルターコンポーネント ("Office") を使用します。基本的には、フィルターはパラメーターにより提供されているオフィス名がソースXML ファイル内のオフィス名と等価かどうかをチェックします。等価の場合、フィルターはソース **Office** アイテムからデータをターゲットコンポーネントにパスします。フィルターに関する詳細はフィルターと条件を参照してください。.
- 2番目のソースXML ファイルからの情報を検索するには、マッピングはユーザー定義関数 "LookupPerson" を呼び出します。この関数のロジックは以下で詳細に説明されています。
- データを処理するために"LookupPerson" 関数は内部で適切な方法で各従業員に関する情報を抽出し連結する他の関数を呼び出します。これらすべての操作は巻数の自身のマッピング内にあり、メインマッピング内では表示されません。これは一般的なカプセル化の例です。"LookupPerson" 関数は **Details** 要素をターゲット XML 内で作成します。

ルックアップの実行

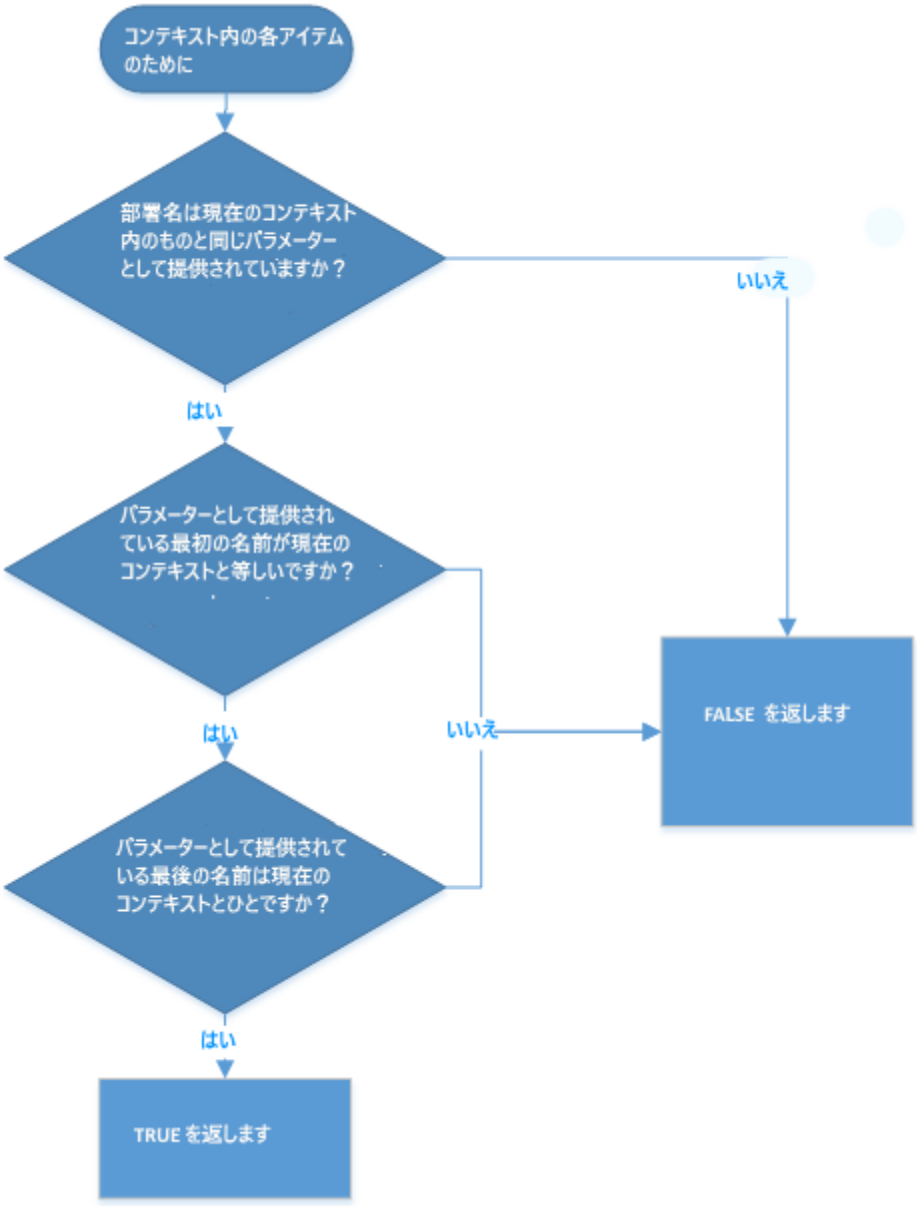
ルックアップ機能は機能について以下で説明されている"LookupPerson" 関数により提供されています。



"LookupPerson" function

上記の通り、関数にはデータを抽出することのできるソースXML ファイルが含まれています。次に、ルックアップ値を提供する3つの入力パラメーターが存在します: **Office_Name**、**First_Name**、および**Last_Name**。捨てべの入力パラメーターは必須として設定されます（すなわち、チェックボックス「入力が必要とされています」がプロパティダイアログボックス内で選択されています）。

"EqualAnd 関数" は現在のものに囲まれている個別のユーザー定義関数です。この機能はブール値を返します。上記のシーケンス内でのこの関数の呼び出しは以下のブールのロジックを説明しています:

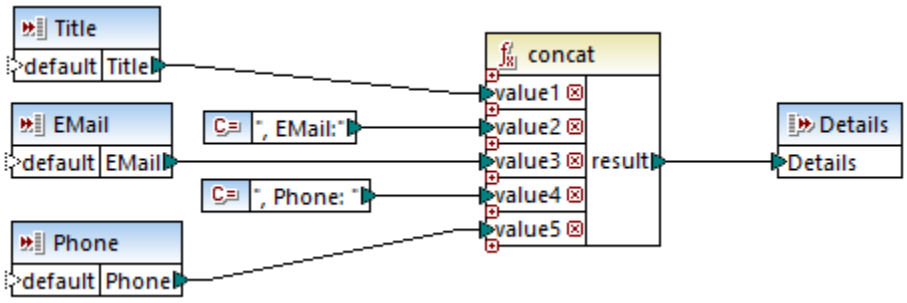


関数の値（TRUE または FALSE）は新規のアイテムが処理される都度フィルターにパスされます。フィルターが値 TRUE を得ると、ルックアップオペレーションは成功し、従業員の詳細が抽出され外部のマッピングに返されます。それ以外の場合、コンテキスト内の次のアイテムが検査され、ループが完了するまで継続されます。

さいしょの "EqualAnd" 関数の発生では、コネクタ **b** は丸枠で囲まれており、これはこのパラメーターが優先コンテキストとして設定されていることを示しています。優先コンテキストはマッピングの実行を最適化する任意の機能です。具体的には、特定のオフィスの個人情報が入力パラメータ **a** により提供され、最初に処理されます。優先コンテキストとしてパラメーターを設定するには、右クリックして、コンテキストメニューから **優先** を選択します。 詳細に関しては 優先コンテキスト を参照してください。

実行の連結

The "Person2Details" 関数は "LookupPerson" 関数にネストされている他の関数です。この関数は単一の値を返します。以下で説明されているとおり、パラメーターとして受け取った3つの値と2つのテキスト定数を連結します:



"Person2Details" function

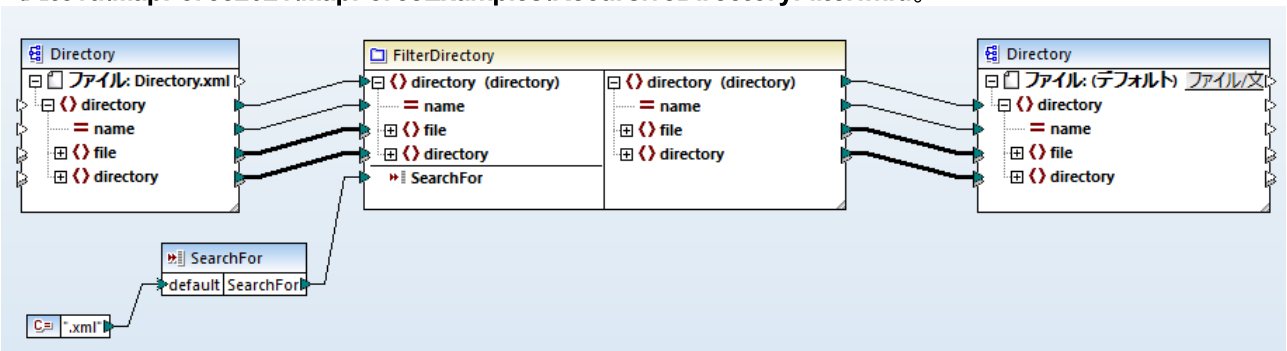
concat 関数はビルトイン関数です。必要とする数量のパラメーターを取ることのできるMapForce ビルトイン関数です。[関数の引数の追加または削除](#)を参照してください。

マッピングの実行

MapForce 内のマッピングの実行をプレビューするには「出力」タブをクリックします。マッピングはデフォルトの入力パラメーター（「Nanonull, Inc.」）を使用して実行され、この結果このオフィスからのみ従業員の情報を抽出します。他のオフィスからデータを抽出するには、入力パラメーターに背景されている定数を「Nanonull, Inc.」から「Nanonull Partners, Inc.」に変更しマッピングを再度実行します。

6.2.10 例:再帰的な検索

このサンプルは、再帰的なユーザー定義関数の助けを使用してソースXML ファイル内でデータを検索するマッピングを表しています。マッピングファイルは次のパスで見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\RecursiveDirectoryFilter.mfd**。



RecursiveDirectoryFilter.mfd

ソースXML ファイルには、下のコードリストに示されているとおり、ファイルとディレクトリに関する情報が含まれています(リスティングではデータの一部が省略されていることにご注意してください):

```
<?xml version="1.0" encoding="UTF-8"?>
<directory name="Examples">
   <directory name="ExampleSite">
      <file name="blocks.sps" size="7473"/>
      <file name="block_file.xml" size="992"/>
      <directory name="output">
         <file name="examplesite1.css" size="3174"/>
         <directory name="images">
```

```
            <file name="blank.gif" size="88"/>
            <file name="block_file.gif" size="13179"/>
         </directory>
      </directory>
   </directory>
</directory>
```

ソース XML ファイル

ソースとターゲット XML ファイルは同じスキーマ **Directory.xsd** を使用します。ファイルシステム上で、ディレクトリはファイルまたは他のディレクトリを含むことができ、これはスキーマ内で反映されます。重要な点は、スキーマが「ディレクトリ」要素が*再帰的*であることを指定していることです（ライン `<xs:element ref="directory"/>` を参照してください）。

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="directory">
      <xs:complexType>
         <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="file">
               <xs:complexType>
                  <xs:attribute name="name" type="xs:string"/>
                  <xs:attribute name="size" type="xs:unsignedLong"/>
               </xs:complexType>
            </xs:element>
            <xs:element ref="directory"/>
         </xs:choice>
         <xs:attribute name="name"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Directory.xsd

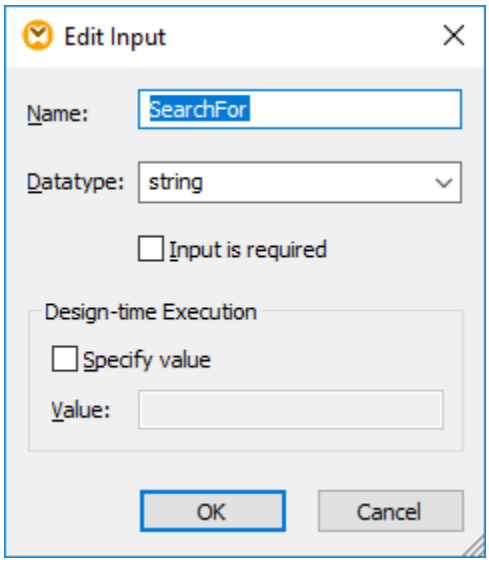
マッピングのビジネスの必要条件は、特定の拡張子を持つファイルの*み*をフィルターします。すべてのディレクトリのネストされた構造は、保存される必要があります。例えば、拡張子が ".xml" の場合、（ソース XML ファイルが以前にリストされているため）期待される出力は以下に類似します:

```
<?xml version="1.0" encoding="UTF-8"?>
<directory name="Examples">
   <directory name="ExampleSite">
      <file name="block_file.xml" size="992"/>
      <directory name="output">
         <directory name="images"/>
      </directory>
   </directory>
</directory>
```

期待される XML 出力

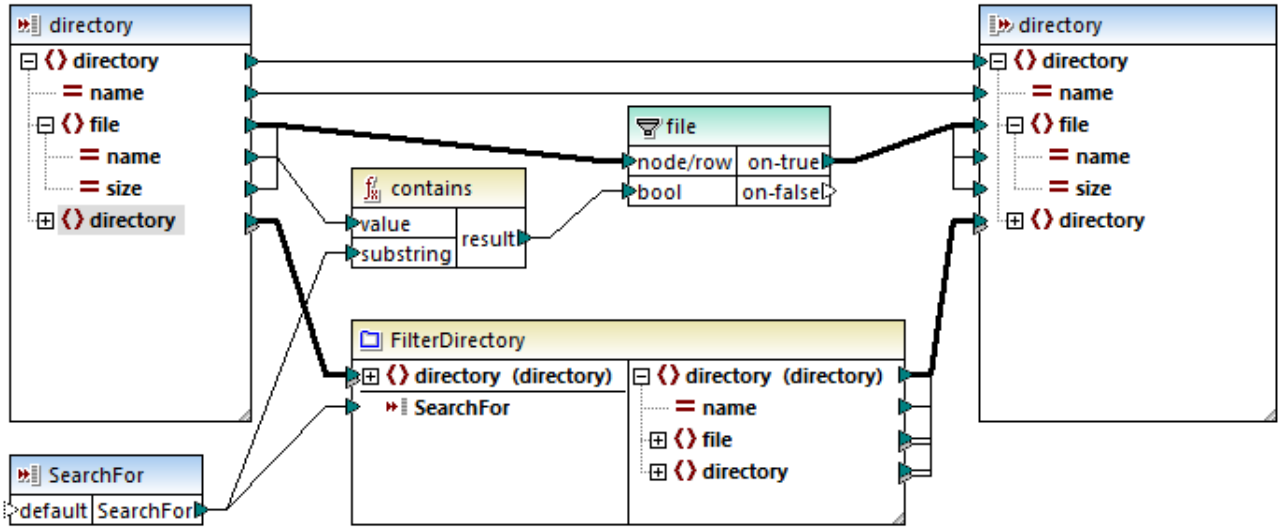
2番目に、マッピングの発信者は、パラメーターとしてファイル拡張子を提供する必要があります。デフォルトで、発信者がパラメーターの値を提供しない場合、マッピングは拡張子を持つ .xml ファイルをフィルターアウトします。

上記の必要条件を対処するために、マッピングにはテキスト定数を介してデフォルトのファイルの拡張子を提供する単純型入力パラメーター、"SearchFor"、が含まれています。このパラメーターは任意です（「入力は必須」チェックボックスはプロパティダイアログボックス内で選択されていません）:



入力パラメーターに関する詳細は、マッピングにパラメーターを与えるを参照してください。

次に、マッピングにはユーザー定義関数、"FilterDirectory" が含まれています。この関数は再帰的で、自身への呼び出しも含んでいます。再帰的な要素の「ディレクトリ」に接続されているため、この関数はソースXMLインスタンス内のネストされた「ディレクトリ」要素を必要なだけ呼び出すことができます。再帰的な呼び出しをサポートするために、この関数は、インライン関数ではなく、正規関数として作成されています（「インラインの使用」オプションは、関数のプロパティ内で選択されていません）。関数のプロパティを確認するには、マッピング内の空のエリアを右クリックし、コンテキストメニューから「関数の設定」を選択します。 ユーザー定義関数の編集を参照してください。



上記で示されるように、関数は入力として2つのパラメーターを取ります:

1. 検索されるXML構成を定義する複合型パラメーター、**Directory**、（このパラメーターは「haystack」です）。
2. 検索するファイル拡張子を指定する文字列パラメーター、**SearchFor**、（このパラメーターは「needle」です）。

マッピング上の入力または出力パラメーターのタイトルバーを右クリックして、設定を確認します。

関数には、MapForce ビルトイン関数 **contains** が接続されているフィルターコンポーネントが含まれています。**contains** 関数は、検索の値がソース構成内の"name"属性(ファイル名)に一致する場合、**true** を返します。**true** の値は、フィルターに現在のアイテムを出力にコピーするように命令します。それ以外の場合、スキップされます。フィルターに関する詳細は、フィルターと条件を参照してください。

関数の**directory** パラメーター(入力と出力の双方)同様、マッピングの、ソースとターゲットファイルは同じスキーマ**Directory.xsd** を持ちます。MapForce は、これらの型の割り当ては整合性があると検知し、入力パラメーターと関数間の接続の型は「全てコピーする」です。全てコピー接続を参照してください。

マッピングの実行

MapForce 内で、マッピングの実行をプレビューするには、「出力」タブをクリックします。デフォルトの入力パラメーター(".xml")を使用してマッピングは実行され、この結果、この検索結果に一致する結果のみが抽出されます。異なる検索の条件を提供するには、入力パラメーターへ接続されている定数を".xml"から".sps"に変更し、マッピングを再度実行します。

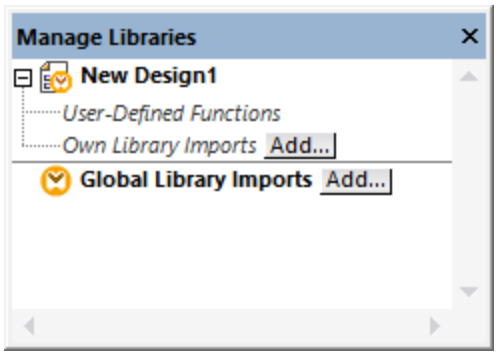
6.3 カスタム XSLT 1.0 または 2.0 関数のインポート

XSLT 1.0、XSLT 2.0 とXSLT3.0 関数ライブラリを MapForce 内で使用することのできる単純型を返すカスタム関数として拡張することができます。

> 簡単なデータ型（例えば、文字列）を返すカスタム関数のみがサポートされます。

XSLT ファイルから関数をインポートする

1. ライブラリ ウィンドウのベースの **ライブラリの追加/削除** ボタンをクリックします。ライブラリの管理ウィンドウが開かれます。



2. 以下の内の1つを行います:
   - （現在のマッピング内のスコープのみで）*ローカル*ライブラリとして関数をインポートする場合現在のマッピング名の下の「**追加**」ボタンをクリックします。
   - （プログラムレベルで）*グローバル*ライブラリとして関数をインポートする場合 **グローバルライブラリのインポート** の横の「**追加**」ボタンをクリックします。

メモ ライブラリを*ローカル*でインポートする場合、ライブラリファイルのパスをマッピングに対して相対的に設定することが可能です。グローバルにインポートされたライブラリでは、インポートされたライブラリへのパスは常に絶対的なパスです。

3. 関数が含まれる .xsl ファイルを参照し、「**開く**」をクリックします。メッセージボックスが表示され、新規のライブラリが追加されていることを通知します。

インポートされたXSLT ファイルは、ライブラリウィンドウ内のライブラリに表示され、全ての名前を持つテンプレートを関数として、ライブラリ名の下に表示されます。インポート済みのライブラリが表示されていない場合、XSLT が変換言語として選択されていることを確認してください。関数ライブラリの管理も参照してください。

以下の点に注意してください:

- MapForce にインポートが許可されている場合、関数はXSLT ファイル内でXSLT 仕様に準拠している名前が付けられたテンプレートとして宣言されている必要があります。XSLT 2.0 ドキュメント内で発生する関数を `<xsl:function name="MyFunction">` の書式でインポートすることもできます。インポートされたXSLT ファイル内で、他のXSLT ファイルをインポートまたは含むことができ、これらのXSLT ファイルと関数もインポートされることができます。
- インポートされたカスタム関数のマップすることのできる入力コネクタはテンプレート呼び出し内で使用されるパラメーターの数により異なります。任意のパラメーターもサポートされています。
- 名前空間はサポートされています。

- MapForce に既にインポートされている XSLT ファイルの更新を行う場合、変更は自動的に検知され、MapForce がファイルの再ロードを促します。
- 名前付きのテンプレートに書き込む場合、テンプレート内で使用されている XPath ステートメントが正しい名前空間を持つことを確認してください。マッピングの名前空間にバインドする名前空間を確認するには、 生成された XSLT コードをプレビューしてください。

XPath 2.0 内のデータ型

XML ドキュメントから XML スキーマが参照されており、スキーマに対してドキュメントが妥当である場合、演算にて目的のデータ型へ暗示的に変換されないデータ型を明示的に構築またはキャストする必要があります。

Altova XSLT 2.0 エンジンで使用される XPath 2.0 データモデルでは、XML ドキュメントから原子化された全てのノード値に `xs:untypedAtomic` データ型が割り当てられます。`xs:untypedAtomic` 型は、以下にあるように暗示的な型変換に使用されます:

例えば、

- `xdt:untypedAtomic` 値が加算演算子により `xs:double` に*暗黙的に*変換されるため式 `xs:untypedAtomic("1") + 1` は2の値を結果とします。
- 四則演算では、オペランドが暗黙的に `xs:double` へ変換されます。
- 値の比較を行う場合、比較の前にオペランドが `xs:string` へ変換されます。

次も参照してください

例: カスタム XSLT 関数の追加
例: ノードの値の集計
XSLT 1.0 エンジンの実装
XSLT 2.0 エンジンの実装

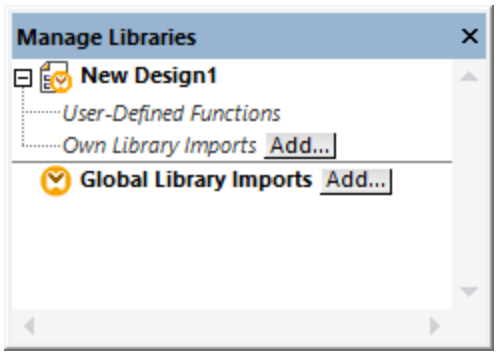
6.3.1 例: カスタム XSLT 関数の追加

この例は、カスタム XSLT 1.0 関数を MapForce にインポートする方法を説明しています。以下に示される単純なサンプルに必要なファイルは <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥ ディレクトリに収められています。

- **Name-splitter.xslt**. この XSLT ファイルは"string" のパラメーターを伴う "**tokenize**" という名前付きテンプレートを定義しています。テンプレートは入力文字列と作業し、それぞれの大文字の文字を空白文字で区切ります。
- **Name-splitter.xml** (処理されるソース XML インスタンスファイル)
- **Customers.xsd** (ソース XML スキーマ)
- **CompletePO.xsd** (ターゲット XML スキーマ)

カスタム XSLT 関数の追加方法:

1. ライブラリ ウィンドウのベースのライブラリの追加/削除 ボタンをクリックします。ライブラリの管理ウィンドウが開かれます。

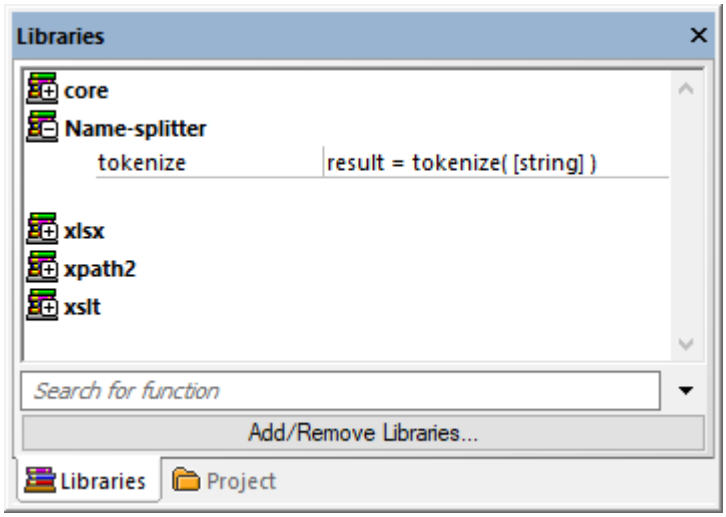


2. 以下の内の1つを行います:

   - (現在のマッピング内のスコープのみで) *ローカル*ライブラリとして関数をインポートする場合現在のマッピング名の下の「**追加**」ボタンをクリックします。
   - (プログラムレベルで) *グローバル*ライブラリとして関数をインポートする場合 **グローバルライブラリのインポート** の横の「**追加**」ボタンをクリックします。

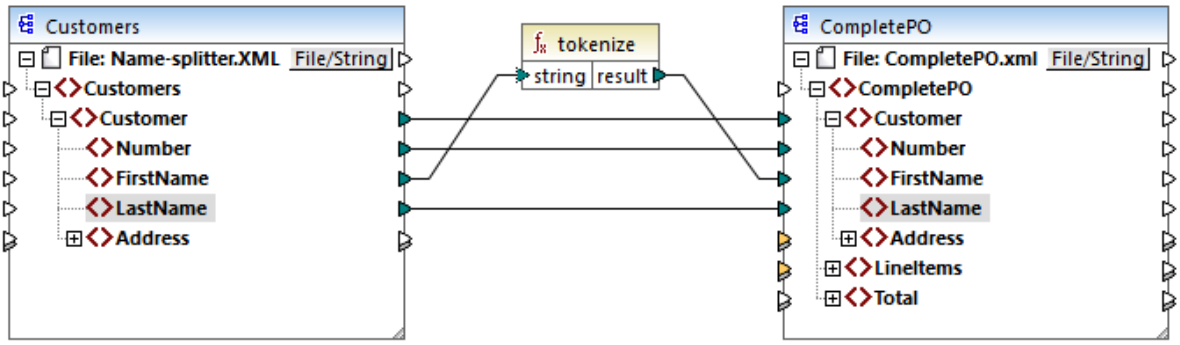
メモ ライブラリを*ローカル*でインポートする場合、ライブラリファイルのパスをマッピングに対して相対的に設定することが可能です。グローバルにインポートされたライブラリでは、インポートされたライブラリへのパスは常に絶対的なパスです。

3. 関数として振る舞う名前の付けられたテンプレートを含む.xsl または.xslt ファイルを参照します。この場合は**Name-splitter.xslt** です。「開く」をクリックします。新規のライブラリが追加されたことを通知するメッセージボックスが表示され、ライブラリウィンドウ内に名前の付けられたテンプレートとして定義された関数と共にXSLT ファイル名が表示されます。(この場合は、`tokenize` 関数を持つ**Name-splitter**)。



XSLT 関数を MapForce から使用する方法:

1. `tokenize` 関数をマッピングウィンドウにドラッグして、以下に示されるようにアイテムをマップします。



2. **XSLT** タブをクリックして生成されたXSLTコードを確認します。

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!--...-->
11  <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:xs="http
12      <xsl:include href="file:///C:/Users/altova/Documents/Altova/MapForce2020/MapForceExamples
13      <xsl:output method="xml" encoding="UTF-8" byte-order-mark="no" indent="yes"/>
14      <xsl:template match="/">
15          <CompletePO>
16              <xsl:attribute name="xsi:noNamespaceSchemaLocation" namespace="http://www.w3.org/
    CompletePO.xsd'"/>
17              <xsl:for-each select="Customers/Customer">
18                  <Customer>
19                      <Number>
20                          <xsl:sequence select="xs:string(xs:integer(fn:string(Number)))"/>
21                      </Number>
22                      <FirstName>
23                          <xsl:call-template name="tokenize">
24                              <xsl:with-param name="string" select="FirstName" as="item()"/>
25                          </xsl:call-template>
26                      </FirstName>
27                      <LastName>
28                          <xsl:sequence select="fn:string(LastName)"/>
29                      </LastName>
30                  </Customer>
31              </xsl:for-each>
32          </CompletePO>
33      </xsl:template>
34  </xsl:stylesheet>
35
```

メモ　マッピング内で名前の付けられたテンプレートが使用されると、名前の付けられたテンプレートを含むXSLTファイルは生成されたXSLTコード（**xsl:include href...**）内に含まれます。そしてコマンド **xsl:call-template** を使用して呼び出されます。

3. 「出力」タブをクリックして、マッピングの結果を確認します。

MapForce からカスタム XSLT ライブラリを削除する方法：

1. ライブラリ ウィンドウのベースのライブラリの追加/削除 をクリックします。ライブラリの管理ウィンドウが開かれます。
2. 削除するライブラリの横のライブラリの削除 ✕ をクリックします。

6.3.2　例: ノードの値の集計

このセクションではXML インスタンスドキュメント内にある複数のノードを処理し、その結果をターゲットアイテムにある単一のアイテムへマッピングする方法について記述します。この例で使用されるファイルは以下のとおりで、具体的には、マッピングの目的は、ソースXML ファイル内の全ての製品の値段を計算し、出力 XML ファイルに単一の値として書き込むことです。この例で使用されるファイルは以下のとおりで、<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥フォルダーに収められています:

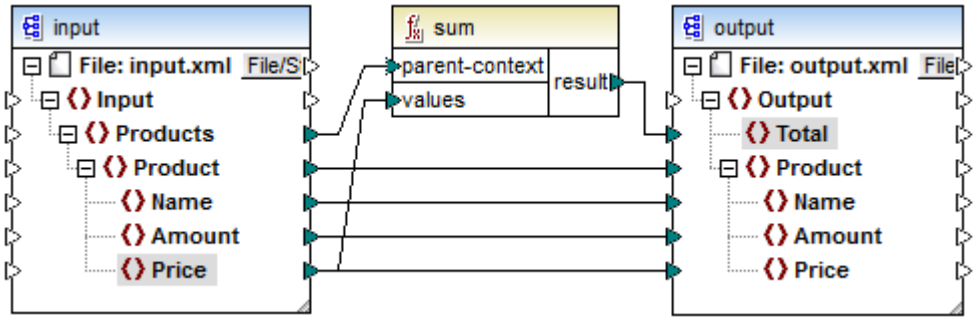
- **Summing-nodes.mfd** – マッピングファイル
- **input.xml** – ソースXML ファイル
- **input.xsd** – ソースXML スキーマ
- **output.xsd** – ターゲット XML スキーマ
- **Summing-nodes.xslt** – カスタムXSLT スタイルシートには個別のノードを集計する名前が付けられたテンプレートが含まれています。

以下の2 つの方法により、集計関数を使用することができます:

- sum 関数を使用します。このMapForce ビルトイン関数はライブラリウィンドウ内で使用することができます。
- カスタムXSLT スタイルシートをMapForce にインポートします。

ソリューション 1: "sum" 集計関数の使用

sum 関数をマッピング内で使用するには、関数をライブラリウィンドウからマッピングにドラッグします。ライブラリウィンドウ内で使用することのできる関数は、選択されたXSLT 言語バージョン(XSLT 1 またはXSLT 2)により異なります。次に、以下に表示されるようにマッピング接続を作成します。



コアライブラリの集計関数に関する情報に関しては core | 集計関数を参照してください。

ソリューション 2: カスタムXSLT スタイルシートの使用

上記の通りにこのサンプルの目的はソースXML ファイル内の製品の `Price` フィールドを集計することです。この場合製品 A とB です。

```
<?xml version="1.0" encoding="UTF-8"?>
<Input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="input.xsd">
   <Products>
      <Product>
         <Name>ProductA</Name>
         <Amount>10</Amount>
         <Price>5</Price>
      </Product>
```

```
      <Product>
         <Name>ProductB</Name>
         <Amount>5</Amount>
         <Price>20</Price>
      </Product>
   </Products>
</Input>
```

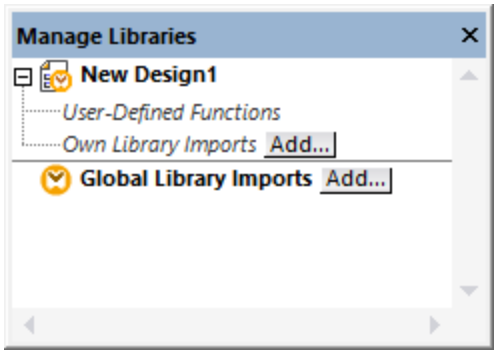
下のコードリスティングは、名前をつけられたテンプレート "Total" と単一パラメーター `string` を使用するカスタム XSLT スタイルシートを表示しています。テンプレートは、XML 入力ファイルを介して作動し、XPath 式 `/Product/Price` により取得された全ての値を合計します。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

   <xsl:template match="*">
      <xsl:for-each select=".">
      <xsl:call-template name="Total">
         <xsl:with-param name="string" select="."/>
      </xsl:call-template>
      </xsl:for-each>
   </xsl:template>

   <xsl:template name="Total">
   <xsl:param name="string"/>
      <xsl:value-of select="sum($string/Product/Price)"/>
   </xsl:template>
</xsl:stylesheet>
```

メモ XSLT 2.0 内のノードを合計するには、スタイルシートの宣言を `version="2.0"` に変更してください。

XSLT スタイルシートを MapForce にインポートする前に XSLT 1.0 を変換言語として選択してください。カスタム関数をインポートする準備が整いました。以下を行ってください:

1. ライブラリ ウィンドウのベースのライブラリの追加/削除 ボタンをクリックします。ライブラリの管理ウィンドウが開かれます。

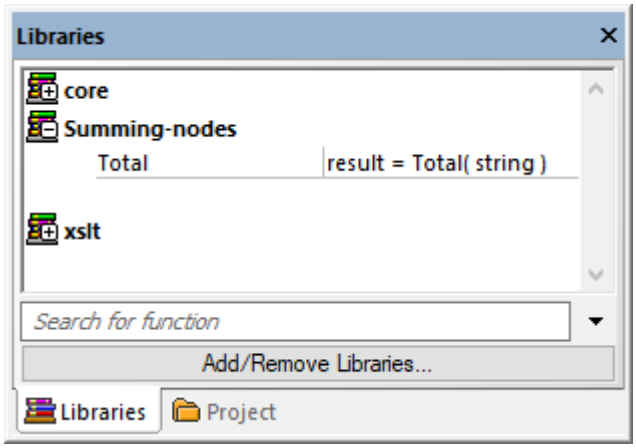


2. 以下の内の 1 つを行います:

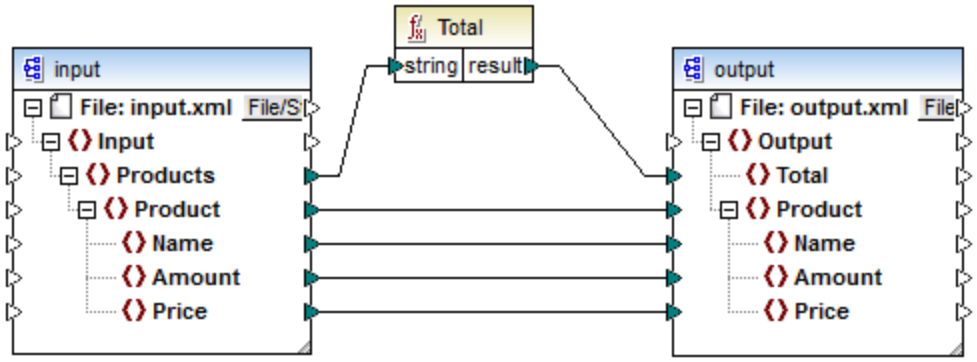
   - (現在のマッピング内のスコープのみで) ローカルライブラリとして関数をインポートする場合現在のマッピング名の下の「追加」ボタンをクリックします。
   - (プログラムレベルで) グローバルライブラリとして関数をインポートする場合 グローバルライブラリのインポート の横の「追加」ボタンをクリックします。

メモ　ライブラリをローカルでインポートする場合、ライブラリファイルのパスをマッピングに対して相対的に設定することが可能です。グローバルにインポートされたライブラリでは、インポートされたライブラリへのパスは常に絶対的なパスです。

3. <マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥Summing-nodes.xslt を参照して「開く」をクリックします。新規のライブラリが追加されたことを通知するメッセージボックスが表示され、新規のライブラリがライブラリウィンドウに表示されます。



4. ライブラリから `Total` 関数をマッピングにドラッグし、下に表示されるとおりマッピング接続を描きます。



マッピングの結果をプレビューするには、「出力」タブをクリックします。2つの `Price` フィールドの合計が `Total` フィールドに表示されます。

```
<?xml version="1.0" encoding="UTF-8"?>
<Output xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="output.xsd">
   <Total>25</Total>
   <Product>
      <Name>ProductA</Name>
      <Amount>10</Amount>
      <Price>5</Price>
   </Product>
   <Product>
      <Name>ProductB</Name>
      <Amount>5</Amount>
      <Price>20</Price>
   </Product>
</Output>
```

6.4 関数ライブラリの管理

MapForce では以下のライブラリの種類をインポートしマッピング内で使用することができます:

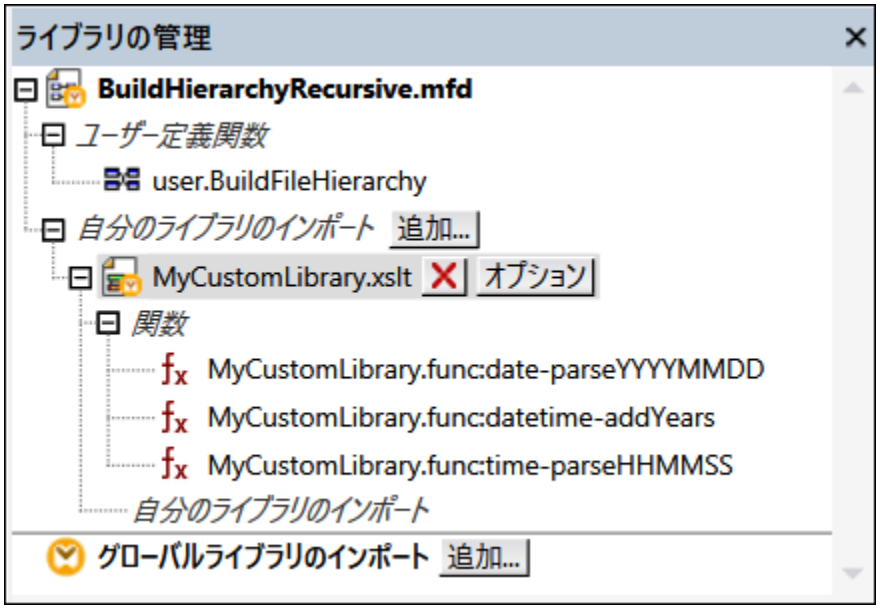
- ユーザー定義関数 (UDF) を含むマッピングデザインファイル(*.mfd)。これはビルドイン関数を使用しコンポーネントをビルディングブロックとして使用するMapForce を使用して作成されたユーザー定義関数を含むマッピングファイルを具体的に指しています。詳細に関しては、ユーザー定義関数の作成 を参照してください。
- 関数を含むXSLT ファイル。カスタムXSLT 関数のインポート内で説明されているとおりMapForce にインポートするためにMapForce 外部で作成されたXSLT 関数を指しています。

ライブラリの管理ウィンドウ

マッピングファイルにより使用されたすべてのライブラリをライブラリの管理ウィンドウからビューして管理することができます。これにはUDF とカスタムライブラリも含まれています。

デフォルトではライブラリの管理ウィンドウは表示されていません。表示するには以下の一つを行うことができます:

- 「**表示**」メニューから「**ライブラリの管理**」をクリックします。
- ライブラリ ウィンドウのベースの**ライブラリの追加/削除** をクリックします。



UDF とライブラリを現在アクティブなマッピングのためにのみ、またはすべての開かれているマッピングのために表示することを選択できます。現在開かれているすべてのマッピングのためにインポートされた関数とライブラリを表示するには、ウィンドウ内を右クリックし、コンテキストメニューから「**開かれているドキュメントの表示**」選択します。

名前の代わりに開かれているマッピングのパスを表示するには、コンテキストメニューから「**ファイルパスの表示**」を選択します。

ライブラリの管理ウィンドウ内に表示されているデータはツリー階層に以下のように整理されています:

- 現在開かれているマッピングドキュメントはトップレベルエントリとして表示されています。各エントリには2つのブランチが存在します: **ユーザー定義関数** と **自身のライブラリインポート**。
  - **ユーザー定義関数** ブランチはそのドキュメント内に含まれているユーザー定義関数を表示しています。

  - **自身のライブラリのインポート** ブランチは現在のマッピングドキュメントに*ローカルで*インポートされたライブラリを表示しています。「ライブラリ」という用語は他のマッピングドキュメント（ユーザー定義関数を含む.mfd ）またはXSLT 1,0 XSLT 2.0、XQuery 1.0*、Java*、C#* で作成されたカスタム外部ライブラリ、または前述の.mff ファイルを指します。マッピングは他のマッピングドキュメントをライブラリとしてインポートする可能性があるため **自身のライブラリのインポート** 構造は複数のレベルの深さになります。
- **グローバルライブラリのインポート** エントリは*アプリケーションレブルでグローバルに*インポートしたカスタムライブラリを含みます。.mfd ファイルの場合、構造は上記の理由から複数のレベルの深さになります。

＊MapForce Professional または Enterprise エディション内でのみこれらの言語はサポートされています

メモ XSLT、XQuery、C#、および Java ライブラリは自身の依存関係を持つ場合があります。このような依存関係はライブラリウィンドウに表示されません。

コンテキストメニューコマンド

オブジェクトを右クリックし、次のコンテキストメニューオプションを選択することにより、ライブラリの管理ウィンドウ内のオブジェクトに対して多種の操作を行うことができます:

コマンド	説明	適用可能範囲
開く	マッピングを開きます。	マッピング
追加	関数のカスタムライブラリを参照するダイアログボックスを開きます。	自身のライブラリのインポート
ライブラリウィンドウ内で関数をロケートする	ライブラリウィンドウのフォーカスを変更し、関数を選択します。	関数
切り取り、コピー、削除	これらの標準 Windows コマンドは MapForce ユーザー定義関数に対してのみ適用が可能です。外部 XSLT ファイルまたは他のライブラリの種類から関数をコピーして貼り付けることはできません。	ユーザー定義関数
貼り付け	以前にクリップボードにコピーされたユーザー定義関数を現在のライブラリに貼り付けることができます。	ライブラリ（UDF）
オプション	現在のライブラリのためにオプションを設定または変更することができるダイアログボックスを開きます。	ライブラリ
開かれているすべてのドキュメントを表示する	このオプションがオンに切り替えられていると、ライブラリの管理ウィンドウは現在開かれているすべてのマッピングを開きます。マッピング間で関数をコピーして貼り付ける場合この機能は通常役に立ちます。それ以外の場合、現在焦点の当たっているマッピングのみが表示されます。	常に
Windows ファイルパスの表示	このオプションがオンに切り替えられていると、ライブラリの管理ウィンドウ内のオブジェクトがフルパスと共に表示されています。それ以外の場合、オブジェクト名のみが表示されています。	常に

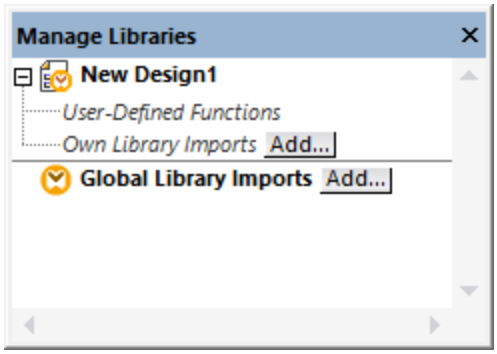
6.4.1 ローカルとグローバルライブラリ

ライブラリを *ローカル* または *グローバル* にインポートすることができます。グローバルなインポートはアプリケーションレベルです。ライブラリがグローバルにインポートされると、全てのマッピングからの関数を使用することができます。

ローカルなインポートはマッピングファイルレベルです。例えば、マッピング **A.mfd** で作業中にマッピング **B.mfd** からすべてのユーザー定義関数をインポートすると仮定します。この場合マッピング **B.mfd** は **A.mfd** にローカルなライブラリとしてインポートされると考えられ、**A.mfd** 内の **B.mfd** からも関数を使用することができます。同様にXSLT ファイルから関数を **A.mfd** にインポートすると、これもローカルなインポートと考えられます。

ライブラリの管理ウィンドウからすべてのローカルとグローバルのインポートを確認し管理することができます。ライブラリをインポートするには、以下を行います:

1. ライブラリ ウィンドウのベースの **ライブラリの追加/削除** ボタンをクリックします。ライブラリの管理ウィンドウが開かれます。



2. 以下の内の1つを行います:

   - (現在のマッピング内のスコープのみで) *ローカル* ライブラリとして関数をインポートする場合現在のマッピング名の下の「**追加**」ボタンをクリックします。
   - (プログラムレベルで) *グローバル* ライブラリとして関数をインポートする場合 **グローバルライブラリのインポート** の横の「**追加**」ボタンをクリックします。

メモ ライブラリを*ローカル*でインポートする場合、ライブラリファイルのパスをマッピングに対して相対的に設定することが可能です。グローバルにインポートされたライブラリでは、インポートされたライブラリへのパスは常に絶対的なパスです。

競合する関数名

以下のレベルで同じ名前を持つ関数が発生するシチュエーションに遭遇する可能性があります:

- メインマッピング内
- ライブラリがローカルでインポートとされている場合
- ライブラリがグローバルでインポートとされている場合

このようなシチュエーションに遭遇した場合、曖昧さを回避するためにMapForce は上記の順序で関数を呼び出す試みを行います。すなわち、ローカルでインポートされたライブラリ内に同じ名前の関数が存在する場合、マッピング内で直接定義されている関数は前例を取ります。また、(両方の関数が同じ名前を持つことを想定して)ローカルでインポートされている関数はグローバルでインポートされている関数よりも前例を優先します。

同じ名前が存在する複数の関数では、上記のルールに従い「優位な」関数が呼び出されます。その他の曖昧な関数名はブロックされます。このようなブロック関数はライブラリウィンドウ内で灰色表示され、マッピング内で使用することはできません。

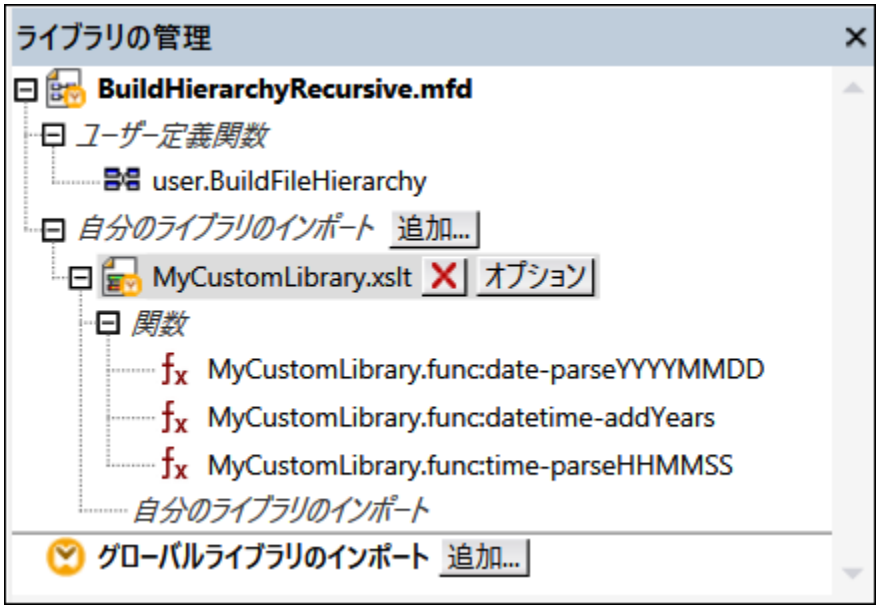
6.4.2 相対的なライブラリパス

ローカルとグローバルなライブラリで説明されているとおりライブラリが(グローバルではなく)ローカルにインポートされていることを前提として、インポートされたライブラリファイルのパスをマッピング デザインファイル(.mfd)に対して相対的に設定することができます。

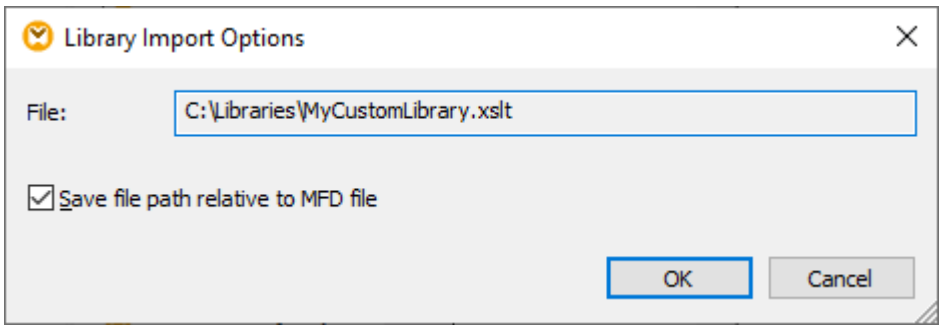
> ライブラリパスを相対的に設定するオプションはドキュメントレベルで *ローカル*にインポートされたライブラリに対してのみ適用することができます。マッピングがプログラムレベルで *グローバル*にインポートされる場合、パスは常に絶対的です。

マッピングデザインファイルに対して相対的にライブラリパスを設定する方法:

1. ライブラリ ウィンドウのベースの**ライブラリの追加/削除** をクリックします。ライブラリの管理ウィンドウが開かれます。



2. 注目するライブラリの横の「**オプション**」をクリックします。(または ライブラリを右クリックして、コンテキスト メニューから を「**オプション**」選択します 、



3. 「 MFD ファイルに対してファイル パスを相対的に保存する」チェックボックスを選択します。

メモ チェックボックスが灰色表示されている場合、ライブラリがグローバルではなくローカルにインポートされていることを確認してください。

チェックボックスが有効化されている場合、「名前を付けて保存」メニューコマンドを使用してマッピングファイルを新規のディレクトリに保存する際にMapForce は参照されるライブラリファイルへのパスを記録し更新します。また、ライブラリファイルがマッピングファイルと同じディレクトリに存在する場合、ディスク上の新しい場所にディレクトリ全体を移動してもパスの参照が壊されることはありません。コンポーネント上で相対的なパスを使用するを参照してください。

「MFD ファイルに対してファイルパスを相対的に保存する」チェックボックスはパスがマッピングファイルに対して相対的になることを指定し、生成されたコードには影響を与えません。生成されたコード内でどのようにライブラリ参照が処理されるかについては、多種の実行環境内のパスを参照してください。

6.5 正規表現

MapForce マッピングのデザイン時、以下のコンテキストで正規表現（"regex"）を使用することができます:

- tokenize-regexp 関数の**pattern** パラメーター内。

XSLT とXquery のため正規表現構文はXML スキーマパート 2: データベースセカンドエディションの付属 F データベースセカンドエディションの付属 F 内で定義されています。

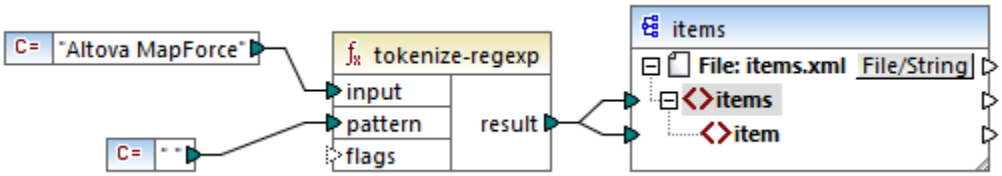
メモ C++、C#、またはJava コードを生成する際には、正規表現構文の高度な機能が異なる可能性があります。詳細に関しては各言語の正規表現ドキュメントを確認してください。

用語

tokenize-regexp 関数を基本の分析正規表現用語を検証してみましょう。この関数は正規表現の助けを借りてテキストを文字列のシーケンスに分割します。これを達成するには、関数は次の入力パラメーターを取ります:

input	関数により処理される入力文字列。正規表現はこの文字列上で操作されます。
pattern	適用される実際の正規表現パターン。
flags	これは正規表現がどのように解釈されるかを決定する追加オプション（フラグ）を定義する任意のパラメーターです。下の"フラグ"を確認してください。

下のマッピングでは、入力文字列は[Altova MapForce] です。**pattern** パラメーターは空白文字で、正規表現フラグは使用されていないことに注意してください。



これは空白文字が発生する都度テキストの分割を引き起こします。マッピングの出力:

```
<items>
   <item>Altova</item>
   <item>MapForce</item>
</items>
```

tokenize-regexp 関数は一致した文字を結果から除外することに注意してください。つまり、このサンプルの空白文字は出力から除外されています。

上のサンプルは基本で、同じ結果を正規表現を使用せず、tokenize 関数を使用して達成することができます。実際的なシナリオでは**pattern** パラメーターは更に複雑な正規表現を含んでいます。正規表現は次のどれかにより構成されることができます:

- リテラル
- 文字クラス
- 文字範囲
- 否定クラス

- メタ文字
- 限定子

リテラル

リテラルを使用して書かれている通り（文字通り）に文字を一致します。例えば、入力文字列が`abracadabra` の場合、**pattern** はリテラル`br` で出力は以下の通りです:

```
<items>
   <item>a</item>
   <item>acada</item>
   <item>a</item>
</items>
```

リテラル`br` は入力文字列 `abracadabra`内に2個の一致を持っています。出力から一致する文字を削除した後、上記の3個の文字列のシーケンスが生成されます。

文字クラス

角かっこ（`[` と`]`）内に文字のセットが含まれており、これが文字クラスを作成します。文字クラス内の文字の1つのみが一致します。例:

- パターン `[aeiou]` は小文字の母音アルファベットに一致します。
- パターン `[mj]ust` は"must" と"just" に一致します。

メモ　パターンに含まれる値は大文字と小文字で区別されるという点に注意してください。小文字の"a" は大文字の"A" にマッチしません。大文字と小文字を区別しないようにするには、**i** フラグを使用してください。以下を参照してください。

文字範囲

`[a-z]` を使用して2つの文字間の範囲を作成します。一度にマッチする文字は一文字だけである点に注意してください。例えば、パターン `[a-z]` は"a" から"z" までの小文字アルファベットに一致します。

否定クラス

角括弧の開始直後にハット記号（`^`）を使用することで、その文字クラスを否定したことになります。例えば、パターン `[^a-z]` は改行文字を含む文字列クラスに含まれていない文字に一致します。

任意の文字を一致する方法

改行文字を除く任意の単一文字に一致させるためにドット（`.`）メタ文字を使用します。例えば、パターン `.` は全ての単一文字に一致します。

限定子

正規表現内で限定子は一致が発生するために許可される文字またはサブ式の数量を定義します。

?	直前のアイテムのゼロまたは1度の発生に一致します。例えば、パターン `mo?` は"m" と"mo" に一致します。
+	直前のアイテムの1度または複数の発生に一致します。例えば、パターン `mo+` は"mo"、"moo"、"mooo" などに一致します。

*	直前のアイテムのゼロまたは複数の発生に一致します。
{min,max}	*min* と *max* の間の発生数を一致させます。例えば、パターン `mo{1,3}` は"mo"、"moo"、および"mooo"に一致します。

かっこ

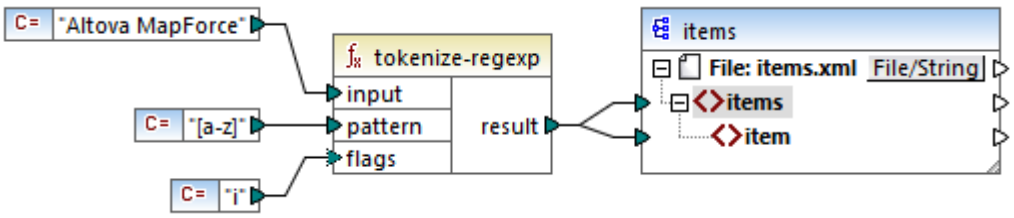
かっこ `(` と `)` は正規表現のパーツをグループ分けするために使用されます。(1文字とは異なり)または代替のある(以下参照)サブ式に限定式を適用するために使用されます。

代替

垂直バー(パイプ)文字 `|` は"or"を意味します。`|` により区切られたサブ式の一部に一致するための使用することができます。例えば、パターン `(horse|make) sense` は"horse sense"と"make sense"に一致します。

フラグ

以下に示されるのはオプションのパラメーターで、正規表現の解釈方法を指定するために使用されます。各フラグは文字に対応します。文字はどの順序でも使用することができ、繰り返して使用することもできます。

s	このフラグが存在する場合、「全てドット」モード内で一致プロセスが操作されます。 **s** フラグがセットされていれば、入力文字列に"hello"と"world"が2つの異なる行にある場合でも、`hello*world` という正規表現によりマッチが行われます。
m	このフラグが存在する場合、「複数行」モード内で一致プロセスが操作されます。 複数行ノードでは、ハット記号 `^` が全ての行頭(文字列全体の開始位置と改行文字の直後に来る文字)に対してマッチするようになります。 更にドル記号 `$` が全ての行末(文字列全体の終了位置と改行文字の直前に来る文字)に対してマッチするようになります。 改行に使用される文字は**#x0A**となります。
i	このフラグが存在する場合、「大文字と小文字を区別する」モード内で一致プロセスが操作されます。例えば、正規表現 `[a-z]` と**i** フラグはすべての文字 a-z と A-Z に一致します。 
x	このフラグが存在する場合、空白文字はマッチング処理が行われる前に正規表現から削除されます。空白文字は**#x09**、**#x0A**、**#x0D** および**#x20** です。 **メモ** 文字クラス内の空白文字は削除されません。例、`[#x20]`。

6.6 関数ライブラリレファレンス

このレファレンスチャプターでは、ライブラリ別に整理された「ライブラリ」ウィンドウ内で使用することのできるMapForce 内蔵の関数に関して説明されています。

メモ

- ライブラリウィンドウ内の関数のライブラリはマッピングの変換言語により異なります。詳細に関しては、下のテーブルを参照してください。
- **XQuery**、**C#**、**C++**、**Java**、および**Built-In** はMapForce Professional またはEnterprise Edition を必要とします。
- シーケンスを扱うXPath 2.0 関数の一部は現在使用することができません。

下のテーブルは各言語内でサポートされている関数をライブラリ別にリストしています。

core | aggregate functions (集計関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
avg		●	●	●	●	●	●	●
count	●	●	●	●	●	●	●	●
max		●	●	●	●	●	●	●
max-string		●	●	●	●	●	●	●
min		●	●	●	●	●	●	●
min-string		●	●	●	●	●	●	●
string-join		●	●	●	●	●	●	●
sum	●	●	●	●	●	●	●	●

core | conversion functions (変換関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
boolean	●	●	●	●	●	●	●	●
format-date		●	●		●	●	●	●
format-dateTime		●	●		●	●	●	●
format-number	●	●	●		●	●	●	●
format-time		●	●		●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
number	●	●	●	●	●	●	●	●
parse-date					●	●	●	●
parse-dateTime					●	●	●	●
parse-number					●	●	●	●
parse-time					●	●	●	●
string	●	●	●	●	●	●	●	●

core | file path functions (ファイルパス関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
get-fileext	●	●	●	●	●	●	●	●
get-folder	●	●	●	●	●	●	●	●
main-mfd-filepath	●	●	●	●	●	●	●	●
mfd-filepath	●	●	●	●	●	●	●	●
remove-fileext	●	●	●	●	●	●	●	●
remove-folder	●	●	●	●	●	●	●	●
replace-fileext	●	●	●	●	●	●	●	●
resolve-filepath	●	●	●	●	●	●	●	●

core | generator functions (ジェネレーター関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
auto-number	●	●	●	●	●	●	●	●

core | logical functions (論理関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
equal	●	●	●	●	●	●	●	●
equal-or-greater	●	●	●	●	●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
equal-or-less	●	●	●	●	●	●	●	●
greater	●	●	●	●	●	●	●	●
less	●	●	●	●	●	●	●	●
logical-and	●	●	●	●	●	●	●	●
logical-not	●	●	●	●	●	●	●	●
logical-or	●	●	●	●	●	●	●	●
not-equal	●	●	●	●	●	●	●	●

core | math functions (数学関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
add	●	●	●	●	●	●	●	●
ceiling	●	●	●	●	●	●	●	●
divide	●	●	●	●	●	●	●	●
floor	●	●	●	●	●	●	●	●
modulus	●	●	●	●	●	●	●	●
multiply	●	●	●	●	●	●	●	●
round	●	●	●	●	●	●	●	●
round-precision					●	●	●	●
subtract	●	●	●	●	●	●	●	●

core | node functions (ノード関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
is-xsi-nil	●	●	●	●	●	●	●	●
local-name	●	●	●	●	●	●	●	●
node-name		●	●	●	●	●	●	●
set-xsi-nil		●	●	●	●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
static-node-annotation	●	●	●	●	●	●	●	●
static-node-name	●	●	●	●	●	●	●	●
substitute-missing-with-xsi-nil		●	●	●	●	●	●	●

core | QName functions (QName 関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
QName		●	●	●	●	●	●	●
local-name-from-QName		●	●	●	●	●	●	●
namespace-uri-from-QName		●	●	●	●	●	●	●

core | sequence functions (シーケンス関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
distinct-values		●	●	●	●	●	●	●
exists	●	●	●	●	●	●	●	●
first-items		●	●	●	●	●	●	●
generate-sequence		●	●	●	●	●	●	●
group-adjacent		●	●		●	●	●	●
group-by		●	●		●	●	●	●
group-ending-with		●	●		●	●	●	●
group-into-blocks		●	●		●	●	●	●
group-starting-with		●	●		●	●	●	●
item-at		●	●	●	●	●	●	●
items-from-till		●	●	●	●	●	●	●
last-items		●	●	●	●	●	●	●
not-exists	●	●	●	●	●	●	●	●
position	●	●	●	●	●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
replicate-item		●	●	●	●	●	●	●
replicate-sequence		●	●	●	●	●	●	●
set-empty		●	●	●	●	●	●	●
skip-first-items		●	●	●	●	●	●	●
substitute-missing	●	●	●	●	●	●	●	●

core | string functions (文字列関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
char-from-code		●	●	●	●	●	●	●
code-from-char		●	●	●	●	●	●	●
concat	●	●	●	●	●	●	●	●
contains	●	●	●	●	●	●	●	●
normalize-space	●	●	●	●	●	●	●	●
starts-with	●	●	●	●	●	●	●	●
string-length	●	●	●	●	●	●	●	●
substring	●	●	●	●	●	●	●	●
substring-after	●	●	●	●	●	●	●	●
substring-before	●	●	●	●	●	●	●	●
tokenize		●	●	●	●	●	●	●
tokenize-by-length		●	●	●	●	●	●	●
tokenize-regexp		●	●	●	●	●	●	●
translate	●	●	●	●	●	●	●	●

db | null processing functions (NULL 処理関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
is-not-null					●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
is-null					●	●	●	●
set-null					●	●	●	●
substitute-null					●	●	●	●

edifact | datetime functions (日付時刻関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
auto-format					●	●	●	●
to-date					●	●	●	●
to-datetime					●	●	●	●
to-duration					●	●	●	●
to-time					●	●	●	●

lang | datetime functions (日付時刻関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
age					●	●	●	●
convert-to-utc					●	●	●	●
date-from-datetime					●	●	●	●
datetime-add					●	●	●	●
datetime-diff					●	●	●	●
datetime-from-date-and-time					●	●	●	●
datetime-from-parts					●	●	●	●
day-from-datetime					●	●	●	●
day-from-duration					●	●	●	●
duration-add					●	●	●	●
duration-from-parts					●	●	●	●
duration-subtract					●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
hour-from-datetime					●	●	●	●
hour-from-duration					●	●	●	●
leapyear					●	●	●	●
millisecond-from-datetime					●	●	●	●
millisecond-from-duration					●	●	●	●
minute-from-datetime					●	●	●	●
minute-from-duration					●	●	●	●
month-from-datetime					●	●	●	●
month-from-duration					●	●	●	●
now					●	●	●	●
remove-timezone					●	●	●	●
second-from-datetime					●	●	●	●
second-from-duration					●	●	●	●
time-from-datetime					●	●	●	●
timezone					●	●	●	●
weekday					●	●	●	●
weeknumber					●	●	●	●
year-from-datetime					●	●	●	●
year-from-duration					●	●	●	●

lang | file functions (ファイル関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
read-binary-file								●
write-binary-file								●

lang | generator functions (ジェネレーター関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
create-guid					●	●	●	●

lang | logical functions (論理関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
logical-xor					●	●	●	●
negative					●	●	●	●
numeric					●	●	●	●
positive					●	●	●	●

lang | math functions (数学関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
abs					●	●	●	●
acos					●	●	●	●
asin					●	●	●	●
atan					●	●	●	●
cos					●	●	●	●
degrees					●	●	●	●
divide-integer					●	●	●	●
exp					●	●	●	●
log					●	●	●	●
log10					●	●	●	●
max					●	●	●	●
min					●	●	●	●
pi					●	●	●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
pow					●	●	●	●
radians					●	●	●	●
random					●	●	●	●
sin					●	●	●	●
sqrt					●	●	●	●
tan					●	●	●	●
unary-minus					●	●	●	●

lang | QName functions (QName 関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
QName-as-string					●	●	●	●
string-as-QName					●	●	●	●

lang | string functions (文字列関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
capitalize					●	●	●	●
charset-decode								●
charset-encode								●
count-substring					●	●	●	●
empty					●	●	●	●
find-substring					●	●	●	●
format-guid-string					●	●	●	●
left					●	●	●	●
left-trim					●	●	●	●
lowercase					●	●	●	●
match-pattern					●		●	●

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
pad-string-left					●	●	●	●
pad-string-right					●	●	●	●
repeat-string					●	●	●	●
replace					●	●	●	●
reversefind-substring					●	●	●	●
right					●	●	●	●
right-trim					●	●	●	●
string-compare-ignore-case					●	●	●	●
string-compare-ignore-case					●	●	●	●
uppercase					●	●	●	●

mime | mime

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
content-encoding								●
content-type								●
decode-mime-entity								●
mime-entity								●

xbrl | Unit helpers

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
xbrl-measure-currency					●	●	●	●
xbrl-measure-pure					●	●	●	●
xbrl-measure-shares					●	●	●	●

xlsx | datetime functions (日付時刻関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
columnname-to-index		●	●		●		●	●
date-to-xlsx		●	●		●		●	●
datetime-to-xlsx		●	●		●		●	●
index-to-columnname		●	●		●		●	●
time-to-xlsx		●	●		●		●	●
xlsx-to-date		●	●		●		●	●
xlsx-to-datetime		●	●		●		●	●
xlsx-to-time		●	●		●		●	●

xpath2 | accessors

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
base-uri		●	●	●				
node-name		●	●	●				
string		●	●	●				

xpath2 | anyURI functions (anyURI 関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
resolve-uri		●	●	●				

xpath2 | boolean functions (ブール値関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
false		●	●	●				
true		●	●	●				

xpath2 | constructors

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
xs:ENTITY		●	●	●				
xs:ID		●	●	●				
xs:IDREF		●	●	●				
xs:NCName		●	●	●				
xs:NMTOKEN		●	●	●				
xs:Name		●	●	●				
xs:QName		●	●	●				
xs:anyURI		●	●	●				
xs:base64Binary		●	●	●				
xs:boolean		●	●	●				
xs:byte		●	●	●				
xs:date		●	●	●				
xs:dateTime		●	●	●				
xs:dayTimeDuration		●	●	●				
xs:decimal		●	●	●				
xs:double		●	●	●				
xs:duration		●	●	●				
xs:float		●	●	●				
xs:gDay		●	●	●				
xs:gMonth		●	●	●				
xs:gMonthDay		●	●	●				
xs:gYear		●	●	●				
xs:gYearMonth		●	●	●				
xs:hexBinary		●	●	●				

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
xs:int		●	●	●				
xs:integer		●	●	●				
xs:language		●	●	●				
xs:long		●	●	●				
xs:negativeInteger		●	●	●				
xs:nonNegativeInteger		●	●	●				
xs:nonPositiveInteger		●	●	●				
xs:normalizedString		●	●	●				
xs:positiveInteger		●	●	●				
xs:short		●	●	●				
xs:string		●	●	●				
xs:time		●	●	●				
xs:token		●	●	●				
xs:unsignedByte		●	●	●				
xs:unsignedInt		●	●	●				
xs:unsignedLong		●	●	●				
xs:unsignedShort		●	●	●				
xs:untypedAtomic		●	●	●				
xs:yearMonthDuration		●	●	●				

xpath2 | context functions (コンテキスト関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
current-date		●	●	●				
current-dateTime		●	●	●				
current-time		●	●	●				
default-collation		●	●	●				

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
implicit-timezone		●	●	●				
last		●	●	●				

xpath2 | durations, date and time functions (期間、日付、および時刻関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
adjust-date-to-timezone		●	●	●				
adjust-date-to-timezone		●	●	●				
adjust-dateTime-to-timezone		●	●	●				
adjust-dateTime-to-timezone		●	●	●				
adjust-time-to-timezone		●	●	●				
adjust-time-to-timezone		●	●	●				
day-from-date		●	●	●				
day-from-dateTime		●	●	●				
days-from-duration		●	●	●				
hours-from-dateTime		●	●	●				
hours-from-duration		●	●	●				
hours-from-time		●	●	●				
minutes-from-dateTime		●	●	●				
minutes-from-duration		●	●	●				
minutes-from-time		●	●	●				
month-from-date		●	●	●				
month-from-dateTime		●	●	●				
months-from-duration		●	●	●				
seconds-from-dateTime		●	●	●				
seconds-from-duration		●	●	●				
seconds-from-time		●	●	●				

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
subtract-dateTimes		●	●	●				
subtract-dates		●	●	●				
subtract-times		●	●	●				
timezone-from-date		●	●	●				
timezone-from-dateTime		●	●	●				
timezone-from-time		●	●	●				
year-from-date		●	●	●				
year-from-dateTime		●	●	●				
years-from-duration		●	●	●				

xpath2 | node functions (ノード関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
lang		●	●	●				
local-name		●	●	●				
local-name		●	●	●				
name		●	●	●				
name		●	●	●				
namespace-uri		●	●	●				
namespace-uri		●	●	●				
number		●	●	●				
number		●	●	●				

xpath2 | numeric functions (数値関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
abs		●	●	●				
round-half-to-even		●	●	●				

xpath2 | string functions (文字列関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
codepoints-to-string		●	●	●				
compare		●	●	●				
compare		●	●	●				
ends-with		●	●	●				
ends-with		●	●	●				
lower-case		●	●	●				
matches		●	●	●				
normalize-unicode		●	●	●				
replace		●	●	●				
starts-with		●	●	●				
string-to-codepoints		●	●	●				
substring-after		●	●	●				
substring-before		●	●	●				
upper-case		●	●	●				

xpath3 | external information functions (外部情報関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
available-environment-variables			●					
environment-variable			●					
unparsed-text			●					
unparsed-text-available			●					
unparsed-text-lines			●					

xpath3 | formatting functions (書式設定関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
format-date			●					
format-dateTime			●					
format-integer			●					
format-time			●					

xpath3 | math functions (数学関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
acos			●					
asin			●					
atan			●					
atan2			●					
cos			●					
exp			●					
exp10			●					
log			●					
log10			●					
pi			●					
pow			●					
sin			●					
sqrt			●					
tan			●					

xpath3 | URI functions (URI 関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
encode-for-uri			●					

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
escape-html-uri			●					
iri-to-uri			●					

xslt10 | xpath functions (xpath 関数)

関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
lang	●							
last	●							
local-name	●	●	●					
name	●	●	●					
namespace-uri	●	●	●					
position	●							

xslt10 | xslt functions (xslt 関数)

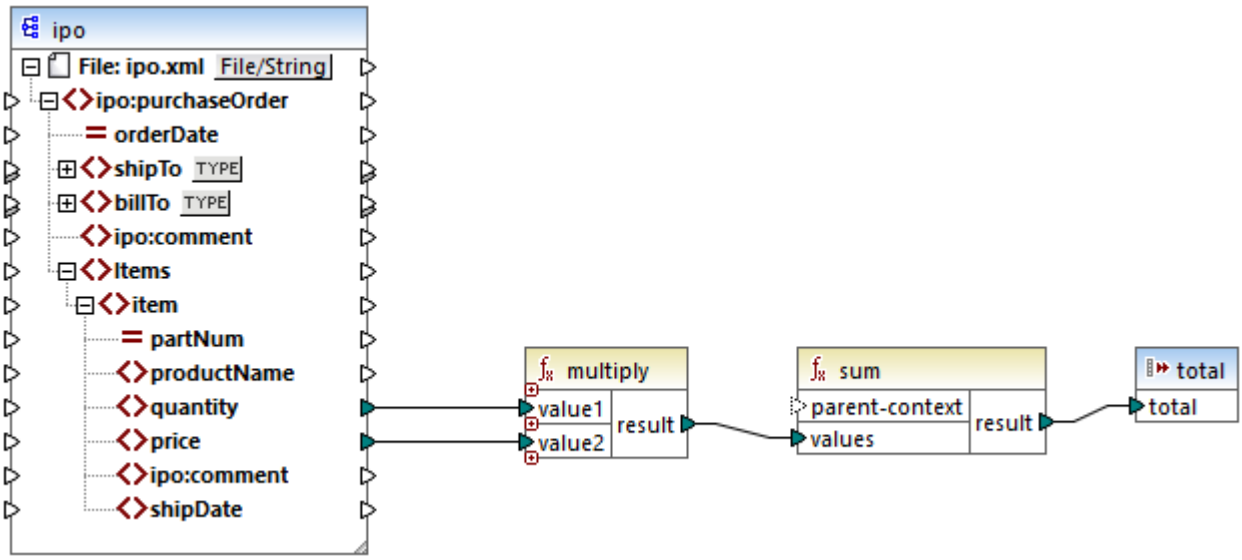
関数	XSLT 1.0	XSLT 2.0	XSLT 3.0	XQuery 1.0	C#	C++	Java	Built-In
currrent	●							
document	●							
element-available	●							
function-available	●							
generate-id	●	●	●					
system-property	●	●	●					
unparsed-entity-uri	●							

6.6.1 core | aggregate functions (集計関数)

「集計」は合計、カウント、平均など1つの結果を取得するために同じ型の複数の値を処理することを意味します。MapForce 内で `avg`、`count`、`max`、および他の集計関数を使用してデータの集計を行うことができます。

以下の2つの引数はすべての集計関数に共通します:

1. **parent-context**。この引数はオプションです。これによりデフォルトのマッピングコンテキストをオーバーライドすることができます（この結果、関数のスコープが、または、関数が反復する値が変更されます）。成功例に関しては例: 親コンテキストの変更で詳細に説明されています。
2. **values**。この引数は、処理する値を与えるソースアイテムに接続されている必要があります。例えば、下に示されるマッピング内で `sum` 関数はソースXMLファイルを基にする数値のシーケンスを入力としてとります。ソースXMLファイル内の各アイテムのために `multiply` 関数はアイテムの価格を個数と掛けた値を取得し `sum` 関数にパスします。`sum` 関数はすべての入力の値を集計し、マッピングの出力である総計結果を生成します。**...\MapForceExamples\Tutorial** ディレクトリ内でこのマッピングを見つけることができます。

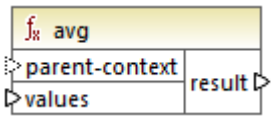


SimpleTotal.mfd

`min`、`max`、`sum` と `avg`、などの集計関数の一部は排他的に数値とのみ作業することができます。これらの関数の入力データが **decimal** データ型に変換され、処理されます。

6.6.1.1 avg

入力シーケンスから得られた値の平均値を返します。空のセットの平均は結果は空のセットになります。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

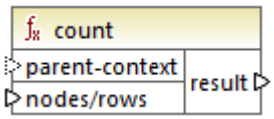
引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
values	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられる引数の値は数値である必要があります。

サンプル

サンプル: キー別にレコードをグループ分けする方法を参照してください

6.6.1.2 count

入力シーケンスを構成する個別のアイテムの数量を返します。入力が空セットのカウントは0 になります。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

この関数はXSLT 1.0 内では機能が制限されています。

パラメーター

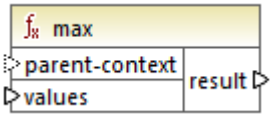
引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
nodes/rows	この引数は数えられるソースアイテムに接続されている必要があります。

サンプル

サンプル: 親コンテキストの変更。

6.6.1.3 max

入力シーケンス内の全ての数値の最小値を返します。空のセットの最大値は空のセットになります。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

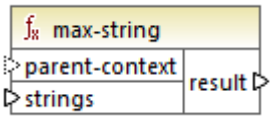
引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
values	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられる引数の値は数値である必要があります。文字列のシーケンスから最大値を得るには max-string 関数を使用してください。

サンプル

サンプル: キー別にレコードをグループ分けする方法を参照してください

6.6.1.4 max-string

入力シーケンス内の全ての文字列の値の最大値を返します。例えば、`max-string("a","b","c")` は `"c"` を返します。**strings** 引数が空のセットの場合、関数は空のセットを返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

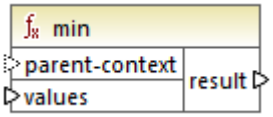
パラメーター

引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。

引数	説明
strings	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられた引数の値は `xs:string` のシーケンス(ゼロまたは複数)である必要があります。

6.6.1.5 min

入力シーケンス内の全ての数値の最小値を返します。空のセットの最小値は空のセットになります。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

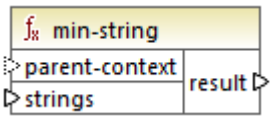
引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
values	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられる引数の値は数値である必要があります。文字列のシーケンスから最小値を得るには min-string 関数を使用してください。

サンプル

サンプル: キー別にレコードをグループ分けする方法を参照してください

6.6.1.6 min-string

入力シーケンス内の全ての文字列の値の最小値を返します。例えば、`min-string("a","b","c")` は `"c"` を返します。**strings** 引数が空のセットの場合、関数は空のセットを返します。



言語

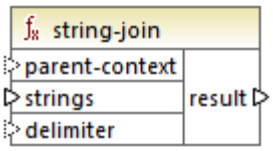
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
strings	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられた引数の値は `xs:string` のシーケンス(ゼロまたは複数)である必要があります。

6.6.1.7 string-join

入力シーケンスのすべての値を区切り文字として使用することを選択した文字列により区切られた一つの文字列に連結します。**strings** 引数が空に設定されている場合関数は空の文字列を返します。



言語

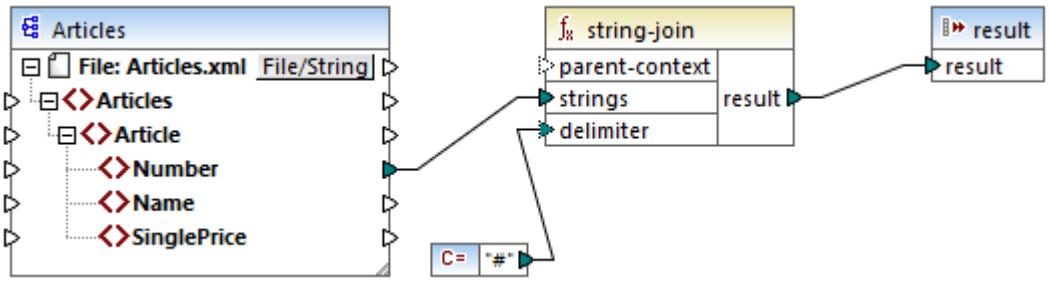
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
strings	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられた引数の値は `xs:string` のシーケンス(ゼロまたは複数)である必要があります。
delimiter	オプションの引数。2つの連続した文字列の間に区切り文字を挿入するように指定します。

サンプル

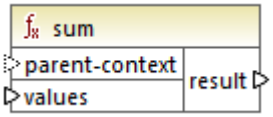
以下に示されるサンプルではソースXMLファイルには以下の番号を持つ4つの**Article**アイテムが含まれています: 1、2、3 と4。



定数は"#"を区切り文字として提供しています。マッピングの結果はこのため`1#2#3#4`です。区切り文字を提供しない場合、結果は`1234`になります。

6.6.1.8 sum

入力シーケンス内のすべての値の合計値を返します。空のセットの合計は0になります。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
parent-context	オプションの引数。親コンテキストを提供します。サンプル: 親コンテキストの変更で詳細に説明されています。
values	この引数は実際のデータを与えるソースアイテムに接続されている必要があります。与えられる引数の値は数値である必要があります。

サンプル

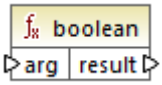
サンプル: ノードの値の集計を参照してください

6.6.2 core | conversion functions (変換関数)

明示的なデータ型変換をサポートするには、**conversion** ライブラリ内で複数の変換関数を使用することができます。多くの場合MapForceは必要な変換を自動的に作成するため、変換関数は常に必要ではありません。変換関数は日付および時刻の値を書式設定する場合、または値を比較する場合において通常役に立ちます。例えば、マッピングアイテムの一部が(整数と文字列など)異なる型の場合 number 変換関数を使用して数値の比較を強制することができます。

6.6.2.1 boolean

arg の値をブール値の値として変換します。(**equal**、**greater** や フィルターとif-else 条件 などの理論関数と作業する際に役に立ちます。ブール値 **false** を得るには空の文字列、または数値 0 を引数として与えます。ブール値 **true** を得るには空の文字列、または数値 1 を引数として与えます。



言語

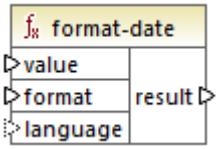
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
arg	必須の引数。変換される値を提供します。

6.6.2.2 format-date

型 `xs:date` の日付の値を特定のオプションに従い文字列と書式に変換します。



言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

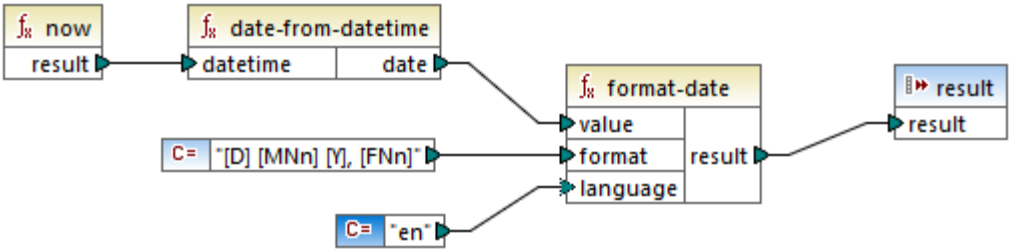
パラメーター

引数	説明
value	変換する `xs:date` 値です。
format	日付がフォーマットされる方法を識別する文字列をフォーマットします。この引数はformat-dateTime 関数内の **format** 引数と同じ方法で使用されます。
language	オプションの引数。与えられると、月の名前と週の曜日が特定の言語で返されます。有効な値:

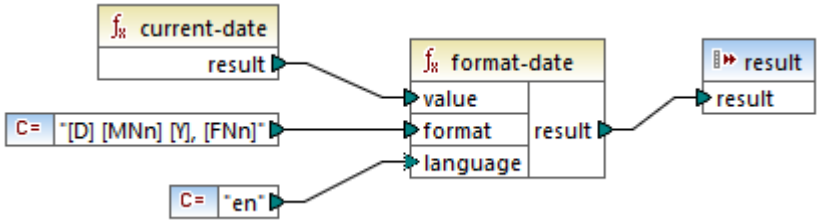
引数	説明	
	de	ドイツ語
	en (default)	英語
	es	スペイン語
	fr	フランス語
	ja	日本語

サンプル

以下のマッピングは以下のようなフォーマットで現在の日付を出力します: "25 March 2020, W ednesday"。この値をスペイン語に翻訳するには、**language** 引数の値を **es** に設定します。

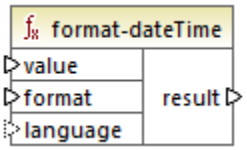


上のマッピングはBuilt-in、C++、C#、またはJava 変換言語のためにデザインされています。XSLT 2.0 では、同じ結果を以下のマッピングで得ることができます:



6.6.2.3 format-dateTime

型 `xs:dateTime` の値を文字列に変換します。日時の文字列の表示は **format** 引数の値に従いフォーマットされます。



言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value	変換する `xs:dateTime` 値です。
format	**value** がフォーマットされる方法を識別する文字列をフォーマットします。下の「説明」を確認してください。
language	オプションの引数。与えられると、月の名前と週の曜日が特定の言語で返されます。有効な値: **de** ドイツ語 **en (default)** 英語 **es** スペイン語 **fr** フランス語 **ja** 日本語

メモ 値はターゲットの型へキャストされるため、関数の出力 (result) が文字列以外のアイテムへ接続されている場合、書式が失われる可能性があります。自動的なキャストを無効化するには ターゲットコンポーネント内の[コンポーネント設定](#) 内の**ターゲット型にターゲット値をキャストする** チェックボックスをクリアします。

リマーク

format 引数 は、`[Y]/[M]/[D]` などの角括弧に囲まれた変数マーカーと呼ばれる文字列により構成されます。角括弧外の文字はリテラルな文字です。角括弧を定数文字として使用する場合、括弧を重ねあわせて記述する必要があります。

各変数マーカーは、表示する日付や時刻を表す識別子、オプションの書式修飾子、プレゼンテーション修飾子、幅修飾子により構成され、これらオプションの修飾子はコンマにより分割されます。

```
format := (literal | argument)*
argument := [component(format)?(presentation)?(width)?]
width := , min-width ("-" max-width)?
```

コンポーネントは以下のとおりです:

識別子	説明	デフォルトの表示
Y	年(絶対値)	4桁の数値 (2010)
M	その年の月	1-12
D	その月の日付	1-31
d	その年の日付	1-366

識別子	説明	デフォルトの表示
F	その週の曜日	曜日の名前(言語に依存)
W	その年の週	1-53
w	その月の週	1-5
H	時(24時間)	0-23
h	時(12時間)	1-12
P	A.M. またはP.M.	アルファベット(言語に依存)
m	その時間の分	00-59
s	その分の秒	00-59
f	小数点以下の秒	数値、小数点1つ
Z	UTC からのタイムゾーンオフセット	+08:00
z	GMT からのタイムゾーンオフセット	GMT+n

書式修飾子は以下であることができます:

文字	説明	サンプル
1	最初に0 を伴わない数値書式:	1, 2, 3
01	2 桁の数値書式:	01, 02, 03
N	コンポーネントの名前、大文字[1]	MONDAY, TUESDAY 1)
n	コンポーネントの名前、小文字[1]	monday, tuesday 1)
Nn	コンポーネントの名前、頭文字が大文字[1]	Monday, Tuesday

脚注:

1.**N**、**n**、および**Nn** 修飾子は以下のコンポーネントのみによりサポートされています: **M**、**d**、**D**。

幅の修飾子は必要な場合最小の幅を示す小数が次に続くコンマにより表示されます。オプションで、最大の幅を示す他の小数が次に表示されるダッシュ文字を追加することもできます。例:

- `[D,2]` は2桁で表示された先頭が0 で表示された月の日です。
- `[MNn,3-3]` は*Jan*、*Feb*、*Mar* など3文字で表示された月の名前です。

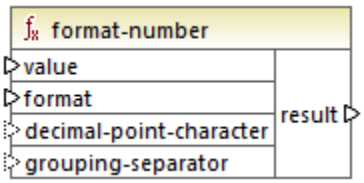
サンプル

format-dateTime 関数を使用して、下のテーブルは`xs:dateTime` 値のフォーマットのサンプルを表示しています。"「値」列は、**value** 引数に与えられる値を指定します。「書式」列は、**format** 引数の値を指定します。「結果」列は、関数により返される内容を表示しています。

値	書式	結果
2003-11-03T00:0:00	`[D]/[M]/[Y]`	2003/03/11
2003-11-03T00:0:00	`[Y]-[M,2]-[D,2]`	2003/11/03
2003-11-03T00:0:00	`[Y]-[M,2]-[D,2] [H,2]:[m]:[s]`	2003-11-03 00:00:00
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f]`	2010 June 02 Wed 153 8:02:12.054
2010-06-02T08:02	`[Y] [MNn] [D01] [F,3-3] [d] [H]:[m]:[s].[f] [z]`	2010 June 02 Wed 153 8:02:12.054 GMT+02:00
2010-06-02T08:02	`[Y] [MNn] [D1] [F] [H]:[m]:[s].[f] [Z]`	2010 June 2 Wednesday 8:02:12.054 +02:00
2010-06-02T08:02	`[Y] [MNn] [D] [F,3-3] [H01]:[m]:[s]`	2010 June 2 Wed 08:02:12

6.6.2.4 format-number

数値を文字列に変換し、特定のオプションに従い書式設定します。



言語

Built-in、C++、C#、Java、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value	必須の引数。フォーマットする数値を提供します。
format	必須の引数。数値がフォーマットされる方法を識別するフォーマット文字列を提供します。下の「説明」を確認してください。
decimal-point-format	オプションの引数。小数点文字として使用される文字を提供します。デフォルトの値はフルストップ（.）文字です。
grouping-separator	オプションの引数。数値グループを分割するために使用されるセパレーター/デミタ。デフォルトの値はコンマ（,）文字です。

メモ 値はターゲットの型へキャストされるため、関数の出力 (result) が文字列以外のアイテムへ接続されている場合、書式が失われる可能性があります。自動的なキャストを無効化するには ターゲットコンポーネント内の[コンポーネント設定] 内の**ターゲット型にターゲット値をキャストする** チェックボックスをクリアします。

リマーク

format 引数は以下の書式を取ります:

```
format := subformat (;subformat)?
  subformat := (prefix)? integer (.fraction)? (suffix)?
  prefix := any characters except special characters
  suffix := any characters except special characters
  integer := (#)* (0)* ( allowing ',' to appear)
  fraction := (0)* (#)* (allowing ',' to appear)
```

最初の *サブフォーマット* は正の数を書式設定するために使用され、2番目のサブフォーマット は負の数をフォーマットするために使用されます。1つの *サブフォーマット* のみが指定されている場合、同じサブフォーマットが負の数のためにも使用されますが、マイナスのシンボルが*プレフィックス*に追加されます。

特別な文字	デフォルト	説明
0 桁	0	結果内のこの時点で桁が常に表示されます。
桁	#	余剰な先行または最終的に0 になる場合を除き、結果の文字列にてこの場所に常に数値が表示されます。
小数点	。	数値の整数部と小数点以下を分けるために使用されます。
grouping-separator	,	桁のグループを区切ります。
パーセント記号	%	数値を100倍して、パーセンテージとして表示します。
パーミル	‰	数値を1000倍して、パーミルとして表示します。

下のテーブルでは書式設定文字列とその結果のサンプルが表示されています。

メモ `format-number` 関数により使用される端数処理メソッドは「ハーフアップ」です。これは値が0.5 よりも大きいまたは等しい場合丸められることを意味します。0.5 未満であれば切り捨てが行われます。このメソッドは、生成されたコードとビルトイン実行エンジンでのみ使用することができます。XSLT 1.0 では四捨五入のモードが定義されていません。XSLT 2.0 では端数処理モードは偶数への丸めです。

数値	書式設定文字列	結果
1234.5	#,##0.00	1,234.50
123.456	#,##0.00	123.46
1000000	#,##0.00	1,000,000.00
-59	#,##0.00	-59.00
1234	###0.0###	1234.0

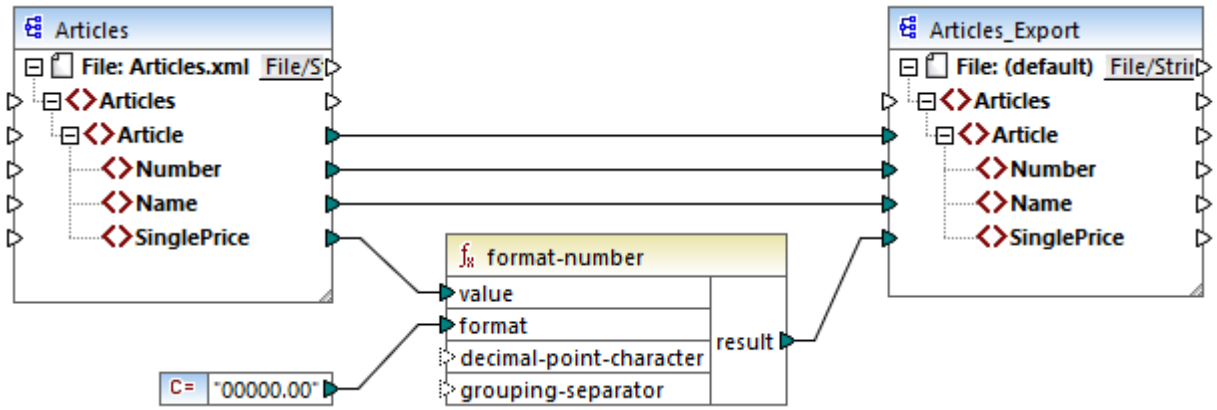
数値	書式設定文字列	結果
1234.5	###0.0###	1234.5
.00025	###0.0###	0.0003
.00035	###0.0###	0.0004
0.25	#00%	25%
0.736	#00%	74%
1	#00%	100%
-42	#00%	-4200%
-3.12	#.00;(#.00)	(3.12)
-3.12	#.00;#.00CR	3.12CR

サンプル

下で説明されるマッピングはソースXMLからデータを読み取りXMLターゲットに書き込みます。ソース内には以下の小数値を含む複数の **SinglePrice** 要素が存在します: **25**, **2.30**, **34**, **57.50**. このマッピングには2つのゴールがあります:

1. 全ての値の左側をゼロでパディングし、重要な部分が5桁になるようにします。
2. 全ての値の左側をゼロでパディングし、重要な部分が2桁になるようにします。

これを達成するために、フォーマット文字列 `00000.00` が `format-number` 関数の引数として提供されています。



PreserveFormatting.mfd

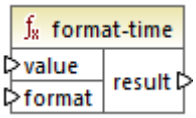
この結果、ターゲット内の値は以下のようになります:

```
00025.00
00002.30
00034.00
00057.50
```

マッピングデザインを次のパスを使用して見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\PreserveFormatting.mfd**。

6.6.2.5 format-time

`xs:time` 入力値を文字列へ変換します。



言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

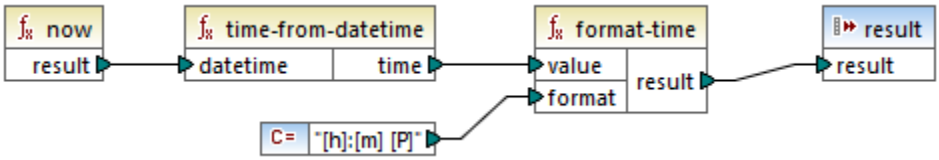
パラメーター

引数	説明
value	必須の引数。フォーマットする `xs:dateTime` 値を提供します。
format	必須の引数。フォーマット文字列を提供します。この引数はformat-dateTime 関数内の **format** 引数と同じ方法で使用されます。

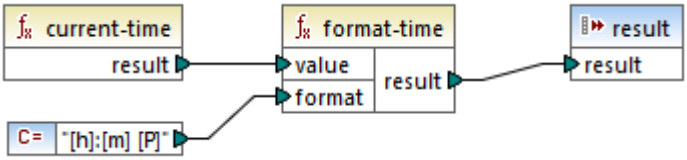
サンプル

以下のマッピングは以下の `2:15 p.m.` ような書式で現在の時刻を出力します。これを達成するには以下の書式のフォーマット文字列 `[h]:[m] [P]` を使用します:

- **[h]** は12 時間書式の現在の時間数です
- **[m]** は現在の分数です
- **[P]** は"a.m." または"p.m." の部分です。

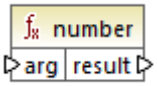


上のマッピングはBuilt-in、C++、C#、またはJava 変換言語のためにデザインされています。XSLT 2.0 では、同じ結果を以下のマッピングで得ることができます:



6.6.2.6 number

arg が文字列または**ブール値**である個所で **arg** の値を数値に変換します。**arg** は文字列で、MapForce は数値として解析を試みます。例えば、`"12.56"` などの文字列が小数の値 `12.56` に変換されます。**arg** が**ブール値 true** の場合、数値 **1** に変換されます。**arg** が**ブール値 false** の場合、数値 **0** に変換されます。



言語

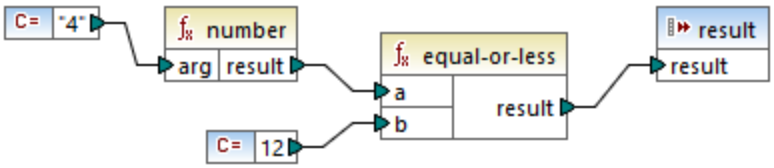
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
arg	必須の引数。変換される値を提供します。

サンプル

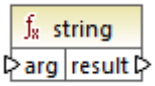
下のサンプルでは、最初の定数は型 `string` で、文字列 "4" を含んでいます。2番目の定数は、定数 12 を含んでいます。2つの値が数値として比較されるためには型が一致する必要があります。



number 関数を最初の定数に追加するには、文字列 "4" の定数を数値 4 に変換します。比較の結果は "true" になります。**number** 関数が使用されない場合結果が "false" の文字列の比較が発生します。(すなわち "4" が直接 **a** に接続されています)。

6.6.2.7 string

入力値を文字列に変換します。ノードのテキストコンテンツを抽出するためにこの関数を使用することもできます。XML 複合型が入力ノードとなる場合、全ての子要素も単一の文字列として出力されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

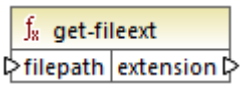
引数	説明
arg	必須の引数。変換される値を提供します。

6.6.3 core | file path functions (ファイルパス関数)

file path 関数によりマッピング内で更に処理するためにフォルダー、ファイル名、および拡張子などのファイル パスデータに直接アクセス、または操作することができます。これらの関数はMapForce にて使用可能な全ての言語で使用することができます。

6.6.3.1 get-fileext

ドット "." 文字を含むファイル パスの拡張子が削除されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

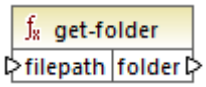
引数	説明
filepath	必須の引数。処理されるファイル パスを提供します。

サンプル

"c:¥data¥Sample.mfd" を引数として提供すると結果は `.mfd` です。

6.6.3.2 get-folder

ファイルパスから、スラッシュやバックスラッシュ(日本語環境では円マーク)を含むフォルダー名が返されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
filepath	必須の引数。処理されるファイルパスを提供します。

サンプル

"c:¥data¥Sample.mfd" を引数として提供すると結果は `c:\data\` です。

6.6.3.3 main-mfd-filepath

マッピングデザインファイル(.mfd) のフルパスを返します。mfd ファイルが現在保存されている場合、空の文字列が返されます。

言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

6.6.3.4 mfd-filepath

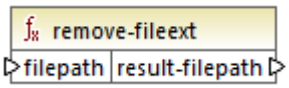
関数がメインマッピング内で呼び出された場合、main-mfd-filepath関数 と同一の値(メインマッピングが含まれている.mfd ファイルのフルパス) が返されます。mfd ファイルが現在保存されていない場合、空の文字列が返されます。.mfd ファイルにより*インポート済み*のユーザー定義関数を呼び出すと、ユーザー定義関数の定義を含む*インポート済みの*.mfd ファイルのフルパスが返されます。

言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

6.6.3.5 remove-fileext

ドット文字を含むファイル パスの拡張子が削除されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

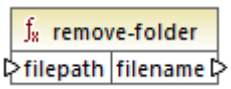
引数	説明
filepath	必須の引数。処理されるファイル パスを提供します。

サンプル

"c:¥data¥Sample.mfd" を引数として提供すると結果は `c:\data\Sample` です。

6.6.3.6 remove-folder

ファイル パスから、スラッシュやバックスラッシュ(日本語環境では円マーク)を含むフォルダー名が削除されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

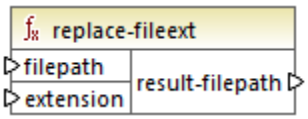
引数	説明
filepath	必須の引数。処理されるファイル パスを提供します。

サンプル

"c:¥data¥Sample.mfd" を引数として提供すると結果は `Sample.mfd` です。

6.6.3.7 replace-fileext

filepath パラメーターにより提供されたファイル パスの拡張子を **extension** パラメーターへの接続により提供されたものと置き換えます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

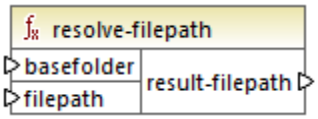
引数	説明
filepath	必須の引数。処理されるファイル パスを提供します。
extension	必須の引数。使用する新規の拡張子を提供します。

サンプル

"c:¥data¥Sample.log" を **filepath** として、".txt" を **extension** として提供すると、結果は `c:\data\Sample.txt` になります。

6.6.3.8 resolve-filepath

ベースフォルダーに対して相対的なファイル パスを解決します。関数は '.'（現在のディレクトリ）と '..'（親ディレクトリ）をサポートします。



言語

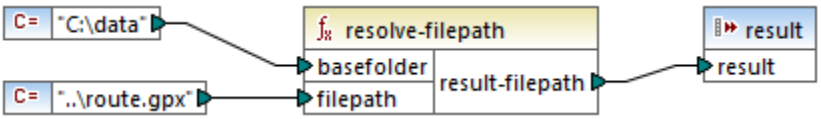
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
basefolder	必須の引数。パスが解決される相対的なベースディレクトリを提供します。これは絶対的または相対的なパスであることができます。
filepath	必須の引数。解決される相対的なファイルパスを提供します。

サンプル

下のマッピングでは相対的なファイルパス **..\route.gpx** は **C:\data** ディレクトリに対して解決されます。



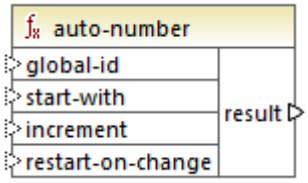
マッピングの結果は `C:\route.gpx` です。

6.6.4 core | generator functions (ジェネレーター関数)

core / generator 関数ライブラリには値を生成する関数が含まれています。

6.6.4.1 auto-number

(例えば、1,2,3,4, ... などの) シーケンスで整数を生成します。インクリメント値である最初の整数と他のオプションをパラメーターを使用して設定することが可能です。



生成されたマッピングコードにより呼び出される関数の順序は定義されていません。MapForce ではキャッシュされた計算結果を再利用するか、任意の順序で条件式を再評価する必要があります。同じ入力パラメーターを使用して複数回呼び出されると他の関数と異なり `auto-number` 関数は異なる結果を返します。このため `auto-number` 関数は注意して使用することが奨励されます。一部の場合、同じ結果を達成するために position 関数を代わりに使用することができます。

言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

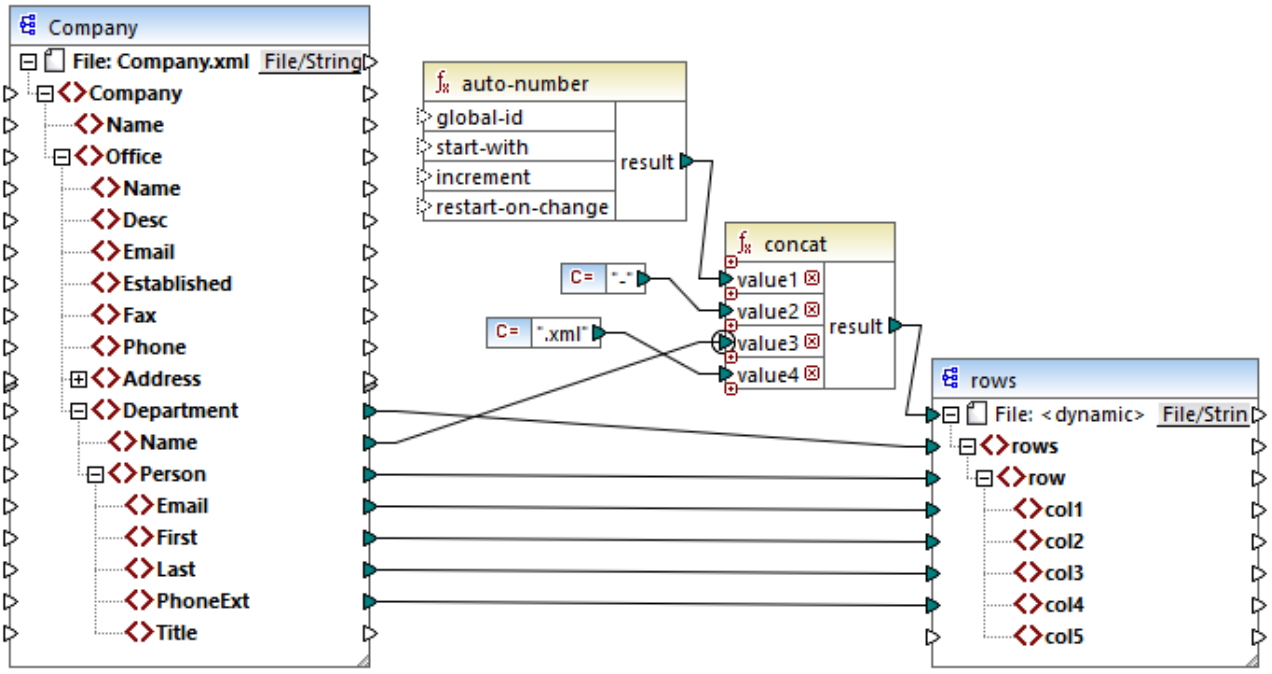
パラメーター

引数	説明
global-id	任意のパラメーターです。マッピングデザインが複数の`auto-number` 関数を含む場合複製（重複する）番号と共にシーケンスを生成します。全ての`auto-number` 関数がお互いを考慮するようにし、重複しないシーケンスを生成するには、各 `auto-number` 関数の（例えば、定数などの）**global-id** 入力に共通の文字列を接続します。
start-with	任意のパラメーターです。生成されたシーケンスが開始する整数を指定します。デフォルト の値は**1** です。
increment	任意のパラメーターです。インクリメント値を指定します。デフォルト の値は**1** です。
restart-on-change	任意のパラメーターです。接続されているアイテムのコンテンツが変更されると、**start-with** へのカウンターをリセットします。

サンプル

以下のマッピングは**ParentContext.mfd** マッピングのバリエーションで、サンプル: 親コンテキストの変更で詳細に説明されています。

下で説明されているマッピングの目的はソースXML ファイル内でそれぞれの部署に1つずつ複数のXML ファイルを生成することです。同じ名前を持つ部署が存在します（これはそれぞれが異なる親部署に属するためです）。このために、生成されたそれぞれのファイル名は連番で始まる必要があります、例えば、**1-Administration.xml**、**2-Marketing.xml** などです。



マッピングの目的を達成するには`auto-number` 関数が使用されます。この関数の結果はダッシュ文字の後に部署名および".xml" 文字列が表示されている形で連結されています。重要な点は`concat` 関数の3番目のパラメーター（部署名）にpriority context が適用されていることです。これには`auto-number` 関数が各部署のコンテキスト内で呼び出され、必要とされる連番の値を生成する効果があります

す。優先コンテキストが使用されない場合 `auto-number` 関数が(コンテキストが存在しない場合) 番号 1 を生成し続け、複製ファイル名が結果として生成されます。

6.6.5 core | logical functions (論理関数)

logical 関数は(通常) 入力されたデータを比較してブール値 [true] または [false] を返すために使用されます。通常、フィルターを使用してサブセットを渡す前段階で、データを評価するために使用されます。ほぼすべての理論関数には以下の構造が存在します:

入力パラメーター= **a** | **b** または **value1** | **value2**
出力パラメーター= 結果

評価結果は、入力の値と比較に使用されたデータ型により異なります。例: 整数の値 **4** と **12** の[より小さい]比較は、4 が 12 より小さいため、ブール値 [true] を返します。2 つの入力パラメーターが文字列の値 `[4]` と `[12]` を含む場合、[4] はアルファベットの順序でオペランド (12) の最初の文字 [1] よりも大きいため、構文的分析はの出力値 "false" を返します。

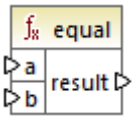
全ての入力値が同じデータ型の場合、比較は共通の型のために行われます。入力値が異なる型の場合、(例: 整数と文字列 または 文字列と日付)そして、使用されるデータ型は、2 つの入力型のうち 最も一般的な(最も制限のない) 入力データ型です。

具体的には、異なる型の2 つの値を比較する前に、すべての入力の値は、共通のデータ型に変換されます。前のサンプルで説明すると、データ型 文字列 は整数より制限が少ないことがわかります。整数の値 **4** を文字列 `[12]` と比較し、整数の値 **4** を文字列 `[4]` に変換し、文字列 `[12]` と比較します。

メモ Logical 関数を null 値の存在をテストするために使用することはできません。null 値を logical 関数の引数として与えると、null 値が返されます。null 値の扱い方に関しての詳しい情報は、Nil 値 / Nillable を参照してください。

6.6.5.1 equal

a が *b* と等価の場合、ブール値 **true** が返され、それ以外の場合 **false** となります。



言語

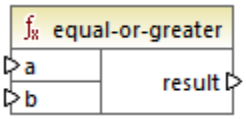
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
a	必須のパラメーターです。比較する最初の値を提供します。
b	必須のパラメーターです。比較する2 番目の値を提供します。

6.6.5.2 equal-or-greater

a が *b* と等価または大きい場合、ブール値 **true** が返され、それ以外の場合 **false** となります。



言語

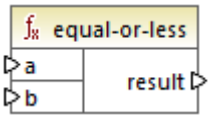
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
a	必須のパラメーターです。比較する最初の値を提供します。
b	必須のパラメーターです。比較する2番目の値を提供します。

6.6.5.3 equal-or-less

a が *b* と等価または小さい場合、ブール値 **true** が返され、それ以外の場合 **false** となります。



言語

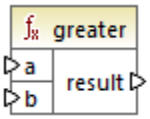
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
a	必須のパラメーターです。比較する最初の値を提供します。
b	必須のパラメーターです。比較する2番目の値を提供します。

6.6.5.4 greater

a が *b* より大きい場合、ブール値 **true** が返され、それ以外の場合 **false** となります。



言語

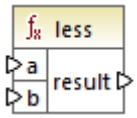
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
a	必須のパラメーターです。比較する最初の値を提供します。
b	必須のパラメーターです。比較する2番目の値を提供します。

6.6.5.5 less

a が *b* より小さい場合、ブール値 **true** が返され、それ以外の場合 **false** となります。



言語

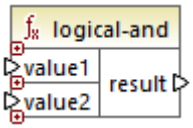
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
a	必須のパラメーターです。比較する最初の値を提供します。
b	必須のパラメーターです。比較する2番目の値を提供します。

6.6.5.6 logical-and

各入力値がtrue の場合のみブール値 **true** を返し、それ以外の場合は**false** を返します。結果を他の`logical-and` 関数に接続することができ、このために全てが**true** を返すようにテストするために、論理 AND と任意の数の条件をジョインすることができます。また、この関数は追加の引数を取るように拡張することができます。関数の引数の追加または削除を参照してください。



言語

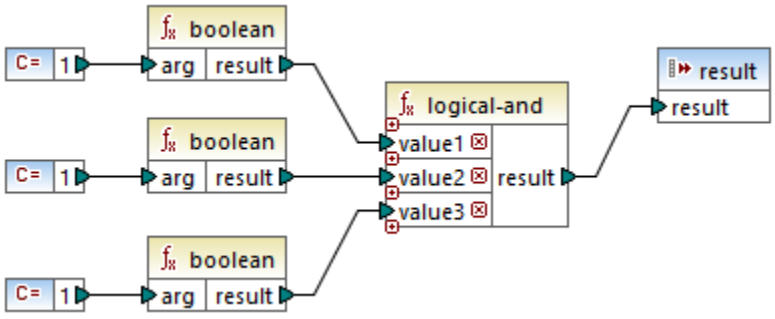
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。比較する最初の値を提供します。
value2	必須のパラメーターです。比較する2番目の値を提供します。

サンプル

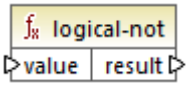
下で説明されているマッピングは`logical-and` への全ての入力値が**true** であるため**true** を返します。入力値のいずれかが**false** の場合、マッピングの結果は**false** になります。



サンプル: ルックアップと連結を参照してください。

6.6.5.7 logical-not

論理状態/結果を反転します。例えば *値* が**true** の場合、関数の結果はfalse です。*値* が**false** の場合、関数の結果はtrue です。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

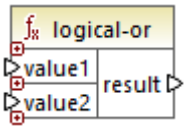
パラメーター

引数	説明
value	必須のパラメーターです。入力パラメーターを提供します。

6.6.5.8 logical-or

両方の入力値をブール型に揃える必要があります。少なくとも入力値の1つが**true** の場合、結果は**true** になります。それ以外の場合結果は**false** になります。

この関数は追加の引数を取るように拡張することができます。関数の引数の追加または削除を参照してください。



言語

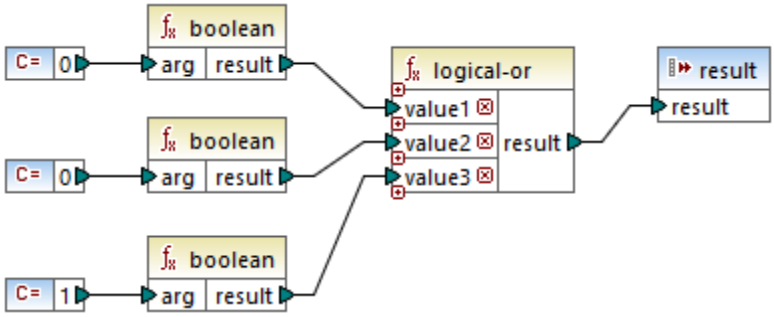
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。比較する最初の値を提供します。
value2	必須のパラメーターです。比較する2番目の値を提供します。

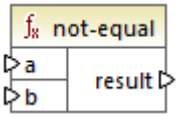
サンプル

少なくとも1つの関数の引数が**true** のため、下のマッピングの結果は**true** です。



6.6.5.9 not-equal

a が *b* と等価ではない場合、ブール値 **true** が返され、それ以外の場合 **false** となります。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

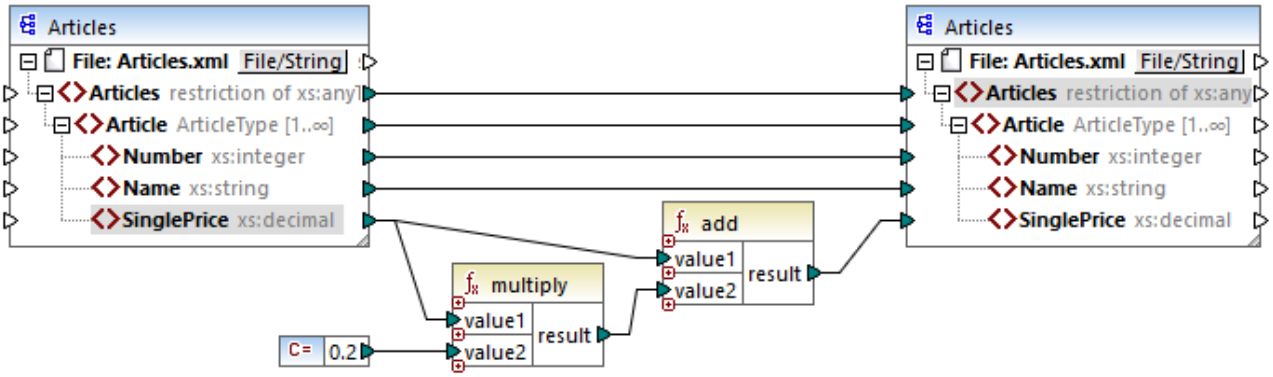
パラメーター

引数	説明
a	必須のパラメーターです。比較する最初の値を提供します。
b	必須のパラメーターです。比較する2番目の値を提供します。

6.6.6 core | math functions (数学関数)

データを使った初歩的な計算を行うために数学関数を利用することができます。これら関数を使用して期間や `datetime` (日時)の計算はできないという点に注意してください。

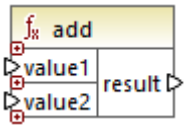
ほぼすべての数学関数は数学オペレーションのオペランドである2つの入力パラメーター(**value1**、**value2**) を取ります。更に処理するために入力値は `decimal` へ自動的に変換され、処理が行われます。数学関数の結果も `decimal` 型です。



上に示されるサンプルでは、ターゲットコンポーネントにマッピングされた各製品の価格に対して20% の消費税が上乗せされます。

6.6.6.1 add

value1 を **value2** に追加して結果を小数の値で返します。この関数は追加の引数を取るように拡張することができます。[関数の引数の追加または削除](#)を参照してください。



言語

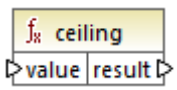
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。最初のオペランドを提供します。
value2	必須のパラメーターです。2番目のオペランドを提供します。

6.6.6.2 ceiling

value より大きい、または、等しい一番小さい整数を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

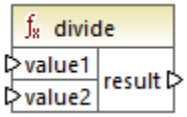
引数	説明
value	必須のパラメーターです。関数の入力パラメーターを提供します。

サンプル

入力値は**11.2** で`ceiling` 関数を適用して結果を**11.2** より大きい一番小さい整数である**12** にします。

6.6.6.3 divide

value1 を**value2** で割り、結果を小数として返します。除算結果の精度はターゲット言語により変化します。round-precision 関数を使用することで、結果の精度を定義することができます。



言語

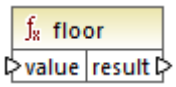
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。最初のオペランドを提供します。
value2	必須のパラメーターです。2番目のオペランドを提供します。

6.6.6.4 floor

value より小さい、または、等しい一番大きい整数を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

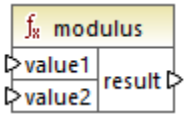
引数	説明
value	必須のパラメーターです。関数の入力パラメーターを提供します。

サンプル

入力値は**11.7** で`floor` 関数を適用して結果を**11.7** より小さい一番大きい整数である**11** にします。

6.6.6.5 modulus

value1 を**value2** で割った値の余りが返されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。最初のオペランドを提供します。
value2	必須のパラメーターです。2番目のオペランドを提供します。

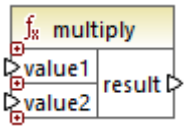
サンプル

入力値が**1.5** と**1** の場合、`modulus` 関数の結果は**0.5** です。説明は`1.5 / 1` はあまりとして**0.5** を返します。

入力値が**9** と**3** の場合 `9 / 3` が結果のため、余りはなく結果は**0** です。

6.6.6.6 multiply

value1 を **value2** と乗算し結果を小数として返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。最初のオペランドを提供します。
value2	必須のパラメーターです。2番目のオペランドを提供します。

6.6.6.7 round

一番近い整数に値を丸め処理します。値が2つの整数の中央にある場合、[正の無限大に向けて半分に丸める]アルゴリズムが使用されます。例えば、値 [10.5] は [11] に丸められ端数処理され、値 [-10.5] は [-10] に丸められ端数処理されます。

言語

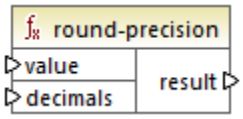
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value	必須のパラメーターです。関数の入力パラメーターを提供します。

6.6.6.8 round-precision

N が**小数**の引数である箇所で、入力値を *N* 小数の桁に丸め端数処理します。



言語

Built-in、C++、C#、Java。

パラメーター

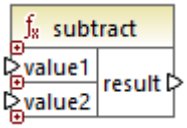
引数	説明
value	必須のパラメーターです。関数の入力パラメーターを提供します。
decimals	必須のパラメーターです。丸めて端数処理する小数の桁を指定します。

サンプル

値 **2.777777** を 2 桁の小数に丸めると **2.78** になります。値 **0.1234** を 3 桁の小数に丸めると **0.123** になります。

6.6.6.9 subtract

value2 を **value1** から減算し、結果を小数値として返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
value1	必須のパラメーターです。最初のオペランドを提供します。
value2	必須のパラメーターです。2番目のオペランドを提供します。

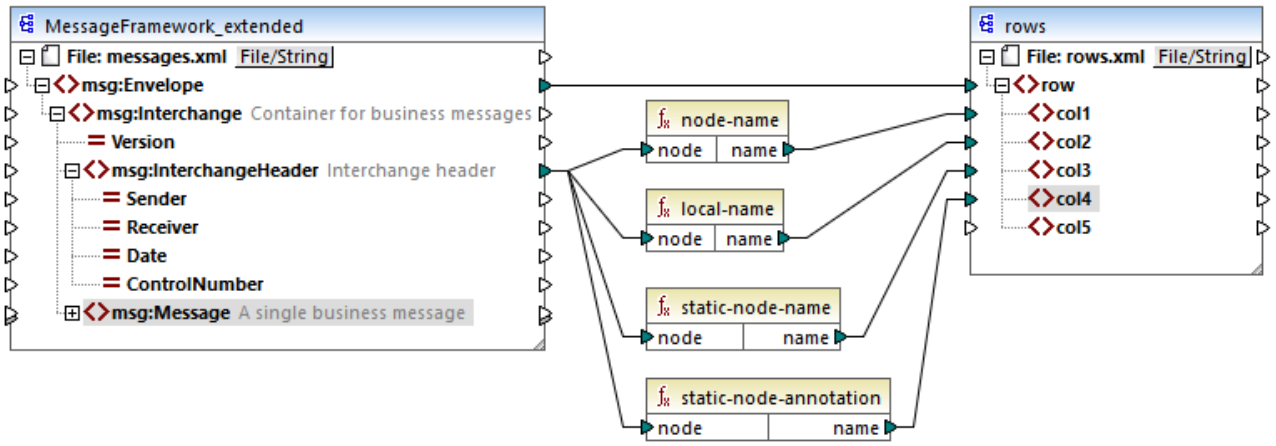
6.6.7 core | node functions (ノード関数)

core | node functions ライブラリからの関数は、名前または注釈などの、マッピングコンポーネント上のノードに関する情報についてのアクセスまたはnillable 要素の処理を許可します。Nil 値 / Nillable も参照してください。

ノード関数を必要とせずにノード名にアクセスする代替法が存在することに注意してください。ノード名のマッピングを参照してください。

下で表示されているマッピングはソースXML ファイルの**msg:InterchangeHeader** ノードから情報を取得するノード関数のいくつかを表示しています。具体的には、以下の情報が抽出されます:

1. `node-name` 関数はノードプレフィックスを含むノードの修飾名を返します。
2. `local-name` 関数はローカルのパートのみを返します。
3. `static-node-name` 関数は`node-name` 関数に類似したものを返します。XSLT 1.0 でも使用することができます。
4. `static-node-annotation` 関数はスキーマ内で定義された通り要素の注釈を取得します。

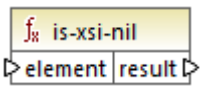


(XML と名前宣言を除く) マッピングの出力は以下の通りです:

```
<row>
   <col1>msg:InterchangeHeader</col1>
   <col2>InterchangeHeader</col2>
   <col3>msg:InterchangeHeader</col3>
   <col4>Interchange header</col4>
</row>
```

6.6.7.1 is-xsi-nil

element 要素の`xsi:nil` 属性が**[true]** に設定されている場合 **[true]** を返します。



言語

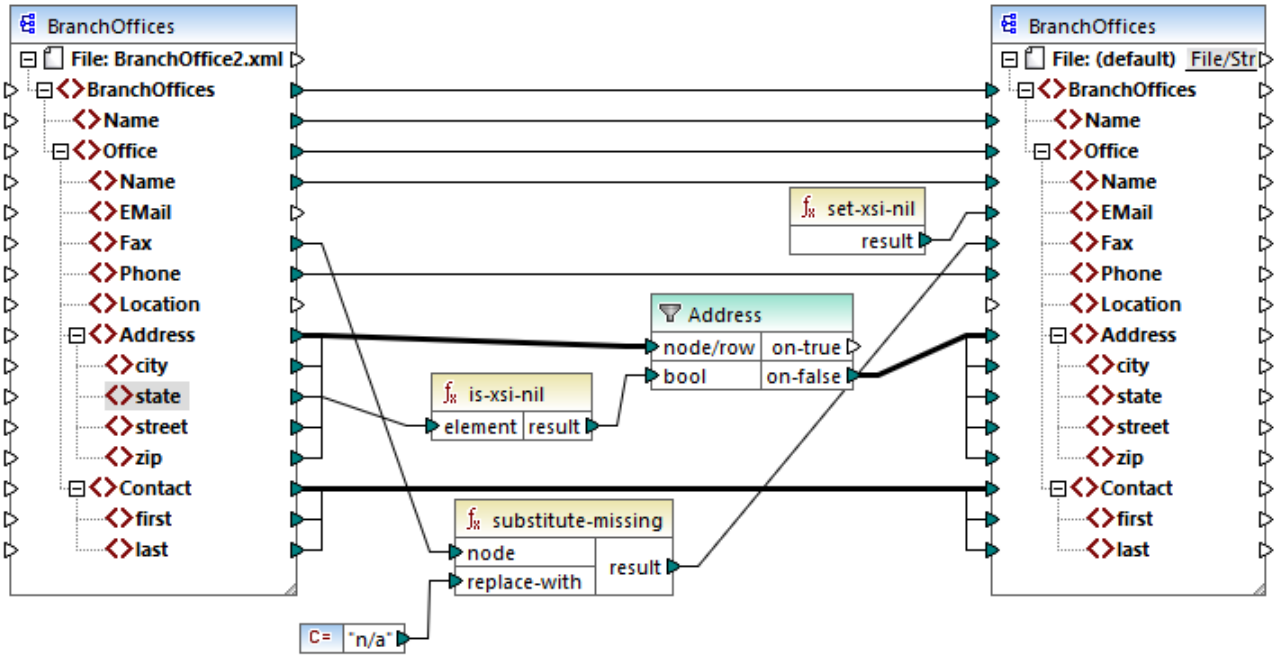
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
element	必須のパラメーターです。チェックされるソースノードに接続されている必要があります。

サンプル

下に表示されるマッピングデザインはデータをソースからターゲット XML ファイルに条件付きでコピーし `is-xsi-nil` を含む複数の関数の使用法を表示しています。このマッピングは**HandlingXsiNil.mfd** と呼ばれくマイドキュメント **>\Altova\MapForce2021\MapForceExamples** ディレクトリ内で見つけることができます。



上で説明される `is-xsi-nil` 関数は `xsi:nil` 属性がソースファイル内の**state** アイテムのために[true] かどうかをチェックします。この属性が[false] の場合フィルターは親 **Address** 要素をターゲットにコピーします。(XML と名前宣言を除く)ソース XML ファイルは以下の通りです:

```
<BranchOffices>
   <Name>Nanonull</Name>
   <Office>
      <Name>Nanonull Research Outpost</Name>
      <EMail>sp@nanonull.com</EMail>
      <Fax xsi:nil="true"/>
      <Phone>+8817 3141 5926</Phone>
      <Address>
         <city>South Pole</city>
```

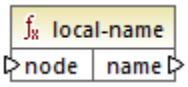
```
         <state xsi:nil="true"/>
         <street xsi:nil="true"/>
         <zip xsi:nil="true"/>
      </Address>
      <Contact>
         <first>Scott</first>
         <last>Amundsen</last>
      </Contact>
   </Office>
</BranchOffices>
```

ソース内にAddress が一つのみ存在し、xsi:nil 属性がstate 要素のために[true] に設定されているために、にマッピングの結果はAddress はターゲットにコピーされません。この結果、マッピングの出力は以下の通りです:

```
<BranchOffices>
   <Name>Nanonull</Name>
   <Office>
      <Name>Nanonull Research Outpost</Name>
      <EMail xsi:nil="true"/>
      <Fax>n/a</Fax>
      <Phone>+8817 3141 5926</Phone>
      <Contact>
         <first>Scott</first>
         <last>Amundsen</last>
      </Contact>
   </Office>
</BranchOffices>
```

6.6.7.2 local-name

ノードのローカル名を返します。node-name 関数とは異なりlocal-name はノードのプレフィックスを返しません。ノードがプレフィックスを持たない場合 local-name とnode-name は同じ値を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

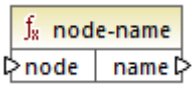
パラメーター

引数	説明
node	必須のパラメーターです。この入力をノード名を取得するノードに接続してください。

6.6.7.3 node-name

接続されているノードの修飾名（QName）を返します。ノードがXML **text()** ノードの場合、空のQname が返されます。この関数は名前付きのノードに対してのみ使用することができます。（ **fn:node-name** を呼び出す）XSLT 2.0 をターゲット言語としている場合、関数は名前を持っていないノードに対して空のシーケンスを返します。

メモ ノード名の取得はノード、データベーステーブル、またはフィールド、XBRL、Excel、JSON またはプロトコールバッファーフィールドは[ファイル入力]フィールドのためにはサポートされていません。



言語

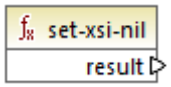
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
node	必須のパラメーターです。この入力をノード名を取得するノードに接続してください。

6.6.7.4 set-xsi-nil

ターゲットノードを xsi:nil にセットします。



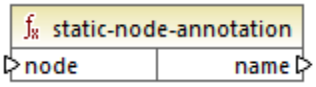
言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

6.6.7.5 static-node-annotation

接続されたノードの注釈と共に文字列を返します。入力は以下である必要があります: (i) ソースコンポーネントノード、または(ii) 呼び出しマッピング内のノードに直接接続されているパラメーターに直接接続されている型 インラインのユーザー定義関数。

接続は直接的である必要があります。フィルター、または標準（非[インライン]）ユーザー定義関数をパススルーすることはできません。これは、接続されたノードから取得されたテキストの生成時に置き換えられた擬似関数で、全ての言語で使用することができます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

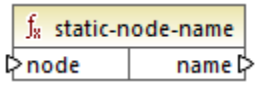
パラメーター

引数	説明
node	必須のパラメーターです。この入力を注釈を取得するノードに接続してください。

6.6.7.6 static-node-name

接続されたノードの名前と共に文字列を返します。入力は以下である必要があります: (i) ソースコンポーネントノード、または(ii) 呼び出しマッピング内のノードに直接接続されているパラメーターに直接接続されている型 インラインのユーザー定義関数。

接続は直接的である必要があります。フィルター、または 非インラインユーザー定義関数をパススルーすることはできません。これは、接続されたノードから取得されたテキストの生成時に置き換えられた擬似関数で、全ての言語で使用することができます。



言語

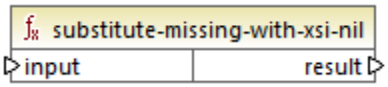
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

引数	説明
node	必須のパラメーターです。この入力をノード名を取得するノードに接続してください。

6.6.7.7 substitute-missing-with-xsi-nil

単純型コンテンツを持つノードでは、この関数はソースコンポーネントの不足する値(またはNULL 値) をターゲットノード内で `xsi:nil` 属性と置き換えます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

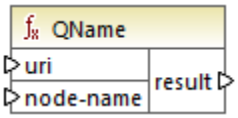
引数	説明
input	必須のパラメーターです。この入力をノード名を取得するノードに接続してください。

6.6.8 core | QName functions (QName 関数)

QName 関数は、XML ドキュメント内の修飾名 (QName) を操作する方法を提供します。

6.6.8.1 QName

名前空間 URI とローカルパートから QName を構築します。Qname をターゲットコンポーネント内で作成するためにこの関数を使用します。定数関数により **uri** と **node-name** パラメーターを提供することができます。



言語

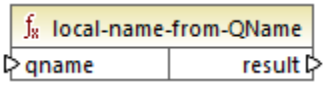
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
uri	必須。URI を提供します。
node-name	必須。ノードの名前を提供します。

6.6.8.2 local-name-from-QName

型 `xs:QName` の値からローカル名の部分が抽出されます。*node* のローカル名を返す local-name 関数とは異なり、この関数は **qname** 入力に接続されているアイテムの *content* を処理します。



言語

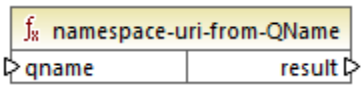
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
qname	必須。型 `xs:QName` の関数入力値を提供します。

6.6.8.3 namespace-uri-from-QName

引数として与えられている Qname 値の名前空間 URI 部分が返されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

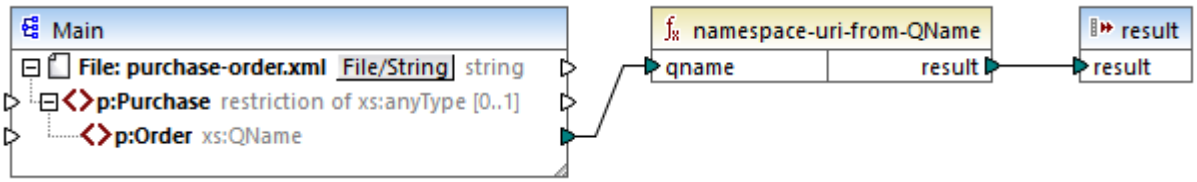
名前	説明
qname	必須。関数の入力パラメーターを提供します。

サンプル

次の XML ファイルには QName 値 **o:name** が含まれています。プレフィックス [o] は名前空間 **`http://NamespaceTest.com/Order`** にマップされています。

```
<?xml version="1.0" encoding="utf-8"?>
<p:Purchase xsi:schemaLocation="http://NamespaceTest.com/Purchase Main.xsd"
          xmlns:p="http://NamespaceTest.com/Purchase"
          xmlns:o="http://NamespaceTest.com/Order"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <p:Order>o:name</p:Order>
</p:Purchase>
```

QName 値を処理し、名前空間 URI を取得するマッピングは下で表示されています:



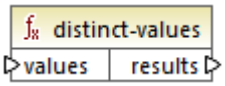
このマッピングの出力は`http://NamespaceTest.com/Order`です。

6.6.9 core | sequence functions（シーケンス関数）

シーケンス関数は入力 シーケンス の処理とそのコンテンツのグループ化を許可します。

6.6.9.1 distinct-values

values 入力に接続されている値のシーケンスを処理し、シーケンスとして一意の値のみを返します。これはシーケンスにて重複した値を取り除き、ユニークなアイテムをターゲットコンポーネントへコピーする際に役に立ちます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
values	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。

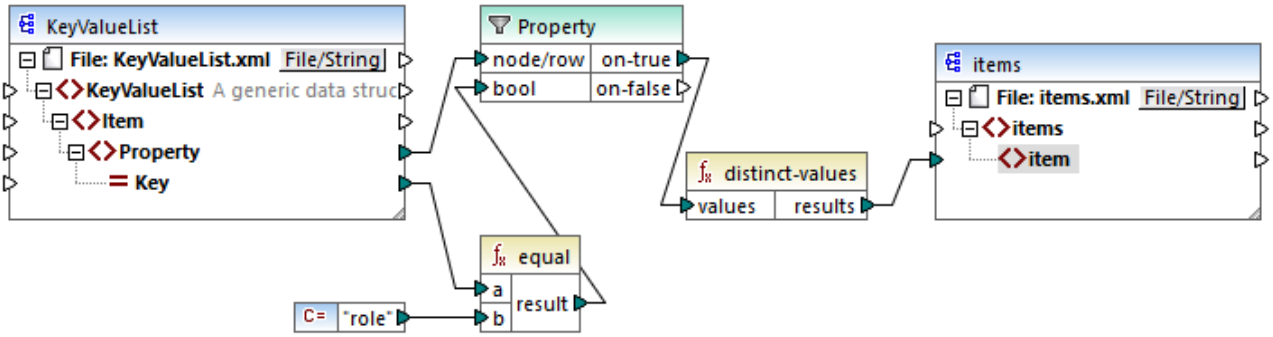
サンプル

次のXML ファイルには架空の会社の社員に関する情報が含まれています。社員の一部は同じロールを有しているため、[ロール]属性ロールには複数の値が含まれています。例えば[Loby Matise] と[Susi Sanna] は同じロール[Support] を持っています。

```
<?xml version="1.0" encoding="UTF-8"?>
<KeyValueList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="KeyValueList.xsd">
   <Item>
      <Property Key="role">Manager</Property>
      <Property Key="First">Vernon</Property>
      <Property Key="Last">Callaby</Property>
   </Item>
```

```
   <Item>
      <Property Key="role">Programmer</Property>
      <Property Key="First">Frank</Property>
      <Property Key="Last">Further</Property>
   </Item>
   <Item>
      <Property Key="role">Support</Property>
      <Property Key="First">Loby</Property>
      <Property Key="Last">Matise</Property>
   </Item>
   <Item>
      <Property Key="role">Support</Property>
      <Property Key="First">Susi</Property>
      <Property Key="Last">Sanna</Property>
   </Item>
</KeyValueList>
```

このXMLファイル内で発生するすべての一意のロール名のリストを抽出する必要があると仮定します。これは以下のマッピング内で達成することができます:



上記のマッピングでは次が発生します:

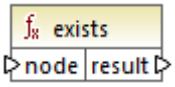
- ソースXMLファイルからの各 **Property** 要素はフィルターにより処理されます。
- フィルターの**bool** 入力への接続は**Key** 属性が[role] に等しい個所のみで **Property** 要素がターゲットコンポーネントに提供されることを保証します。文字列 [role] は定数により与えられます。(フィルターの条件を満たす [Support] プロパティが2つ存在するために)フィルターの出力はまだこの時点では複製を作成することに注意してください。
- フィルターにより作成されるシーケンスは複製値を除外する `distinct-values` 関数により処理されます。

この結果 (XMLと名前宣言を除く)マッピングの出力は以下の通りです:

```
<items>
   <item>Manager</item>
   <item>Programmer</item>
   <item>Support</item>
</items>
```

6.6.9.2 exists

接続済みのノードが存在する場合は**true** を返します。それ以外の場合は**false** を返します。ブール値を返すためこの関数は子要素または属性を持つ(または持たない)レコードのみをフィルターするために通常 フィルター と共に使用されます。



言語

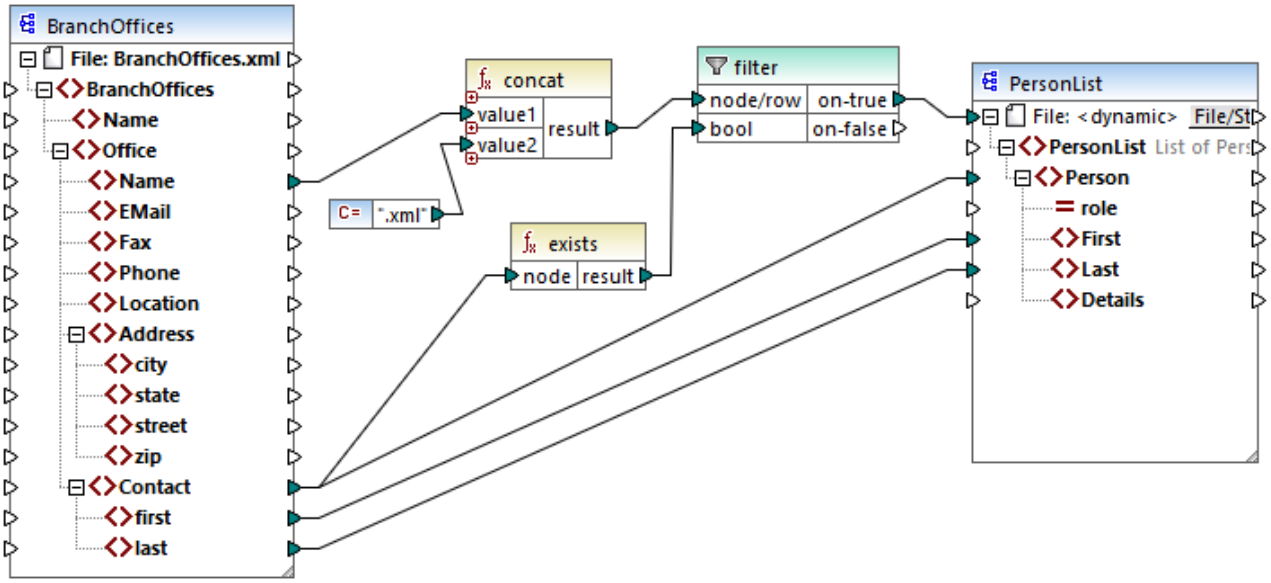
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
node	存在を確認するノード。

サンプル

次のマッピングは`exists` 関数を使用してデータをフィルターする過程を表示しています。このマッピングは **PersonListsForAllBranchOffices.mfd** と呼ばれ<マイドキュメント>¥Altova¥MapForce2021¥MapForceExamples¥ディレクトリ内で見つけることができます。



PersonListsForAllBranchOffices.mfd

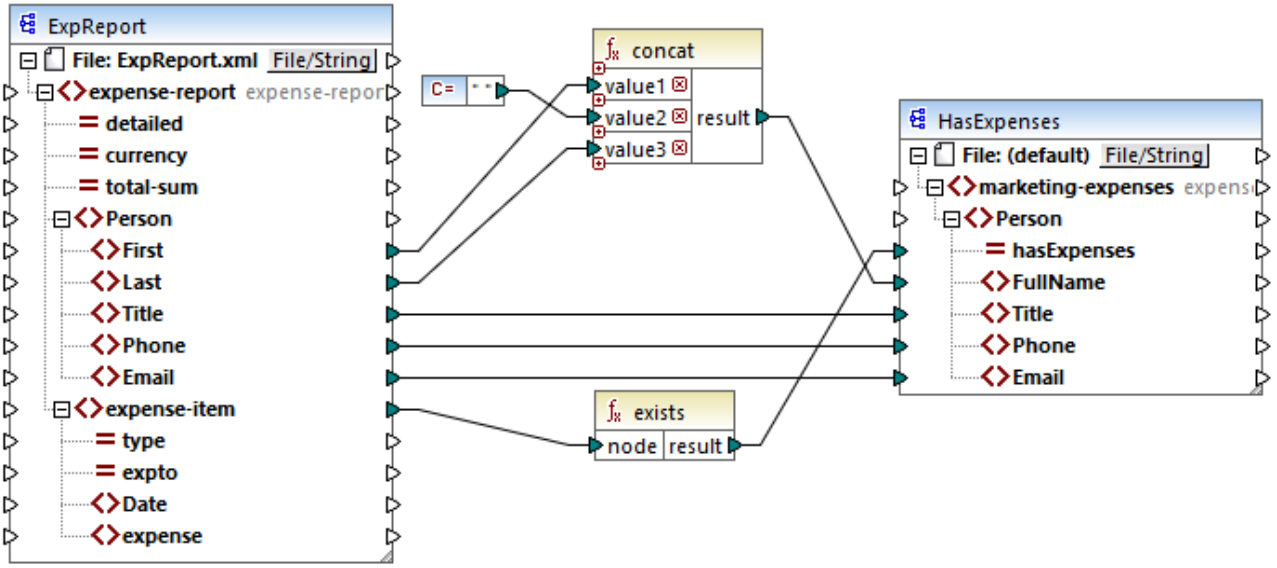
ソースファイル **BranchOffices.xml** 内には3つの **Office** 要素が存在します。オフィスの一つに **Contact** 子要素が存在しないことに気が付きます。マッピングのゴールは複数存在します:

a) 各オフィスのために、そのオフィス内に存在する連絡先のリストを抽出する
b) 各オフィスのために、オフィスと同じ名前を持つ個別のXML ファイルを作成する
c) オフィスに連絡先が存在しない場合 XML ファイルを生成しない

この目的を達成するために、フィルターがマッピングに追加されました。**Contact** アイテムが少なくとも1つが存在する箇所で フィルターは **Office** アイテムをターゲットにパスします。ブール値条件は`exists` 関数により提供されます。関数の結果がtrue の場合、ターゲットファ

イル名を生成するためにオフィスの名前は文字列 `.xml` と連結されます。マッピングからのファイル名の生成の詳細に関しては、複数の入力または出力ファイルを動的に処理を参照してください。

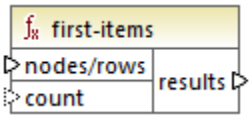
もう一つの例は次のマッピングです: <マイドキュメント>\Altova\MapForce2021\MapForceExamples\HasMarketingExpenses.mfd。**expense-item** がソースXML にて存在する場合、ターゲット XML/スキーマファイルの**hasExpenses** 属性が**true** となります。



HasMarketingExpenses.mfd

6.6.9.3 first-items

N が**count** パラメーターにより提供される個所で、入力シーケンスの最初の *N* 個のアイテムを返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

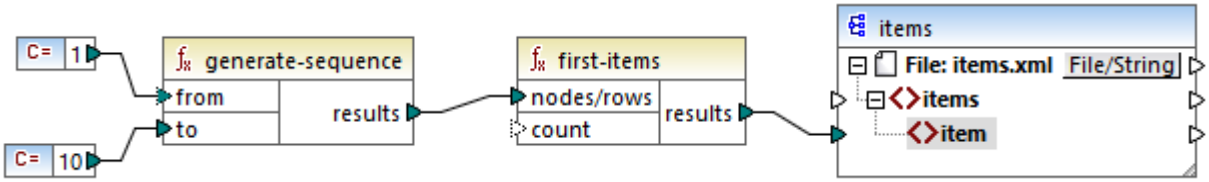
パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。

名前	説明
count	任意のパラメーターです。入力シーケンスから抽出されるアイテムの数量を指定します。デフォルトの値は**1** です。

サンプル

次の模擬マッピングは10 の値のシーケンスを生成します。`first-items` 関数によりシーケンスは処理され、結果はターゲット XML ファイルに書き込まれます。



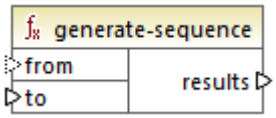
count 引数に値が存在しないため、**1** のデフォルトの値が適用されます。個の結果、シーケンスからの最初の値のみがマッピング出力内に生成されます:

```
<items>
   <item>1</item>
</items>
```

更に現実的なサンプルに関しては、マッピングにパラメーターを提供する内の**FindHighestTemperatures.mfd** マッピングを参照してください。

6.6.9.4 generate-sequence

[from] と[to] パラメーターを境界として使用し、整数のシーケンスを作成します。



言語

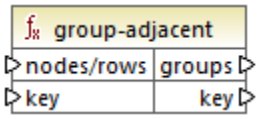
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
from	任意のパラメーターです。シーケンスが開始する整数(下限の数量)を指定します。デフォルト の値は**1** です。
宛先	必須のパラメーターです。シーケンスが終了する整数(上限の数量)を指定します。

6.6.9.5 group-adjacent

`group-adjacent` 関数は、**key** 入力に接続されているキーにより **nodes/rows** 入力に接続されているアイテムをグループ分けします。この関数は、アイテムが隣接しない場合、同じキー情報を共有するアイテムを個別のグループに分けます。複数の連続した（隣接した）アイテムが同じキーを共有する場合、これらのアイテムは同じグループに配置されます。



例えば、下記の抽象的な変換に示されるように、グループキーは「Department」です。このダイアグラムの左側は入力データを示し、右側はグループ分けの後の出力データを示しています。次は、変換の実行中に発生します:

- 最初は、最初のキー「Administration」は新規のグループを作成します。
- 次のキーは異なるため、2番目のグループである「Marketing」が作成されます。
- 3番目のキーもまた異なるための、「Engineering」と呼ばれる他のグループが作成されます。
- 4番目のキーは3番目と同じキーです。このため、レコードは既存のグループに配置されます。
- 最後に、5番目のキーは4番目のキーと異なるため、最後のグループを作成します。

下に説明されるとおり、「Michelle Butler」と「Fred Landis」は隣接する同じキーを持つために一緒にグループ分けされています。しかしながら、「Vernon Callaby」と「Frank Further」は同じキーを所有しているにもかかわらず隣接していないため、異なるグループに分けれています。

Department	Name
Administration	Vernon Callaby
Marketing	Susi Sanna
Engineering	Michelle Butler
Engineering	Fred Landis
Administration	Frank Further

group-adjacent()

Department	Name
Administration	Vernon Callaby

Department	Name
Marketing	Susi Sanna

Department	Name
Engineering	Michelle Butler
Engineering	Fred Landis

Department	Name
Administration	Frank Further

言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります

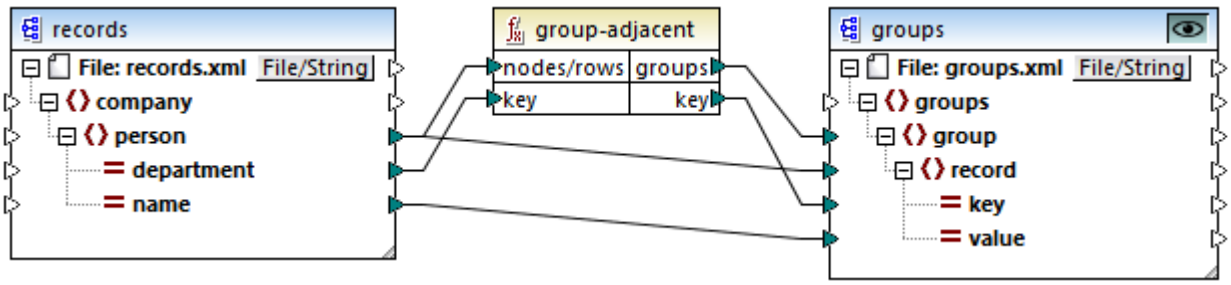
名前	説明
	す。例えば、接続はソースXML アイテムから始まっている場合があります。
key	アイテムをグループするキー

サンプル

使用するソースデータは次のコンテンツを持つXML ファイルです（下のコードリストでは名前空間とXML 宣言は簡素化のために削除されていることにご注意してください）。

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

ビジネスの条件は、隣接する部署の個人レコードをグループ分けします。これを達成するには、次のマッピングが`group-adjacent` 関数を呼び出し、**department** を **key** として提供します。



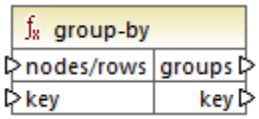
マッピングの結果は以下のとおりです:

```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
   </group>
   <group>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
   <group>
      <record key="Administration" value="Frank Further"/>
   </group>
</groups>
```

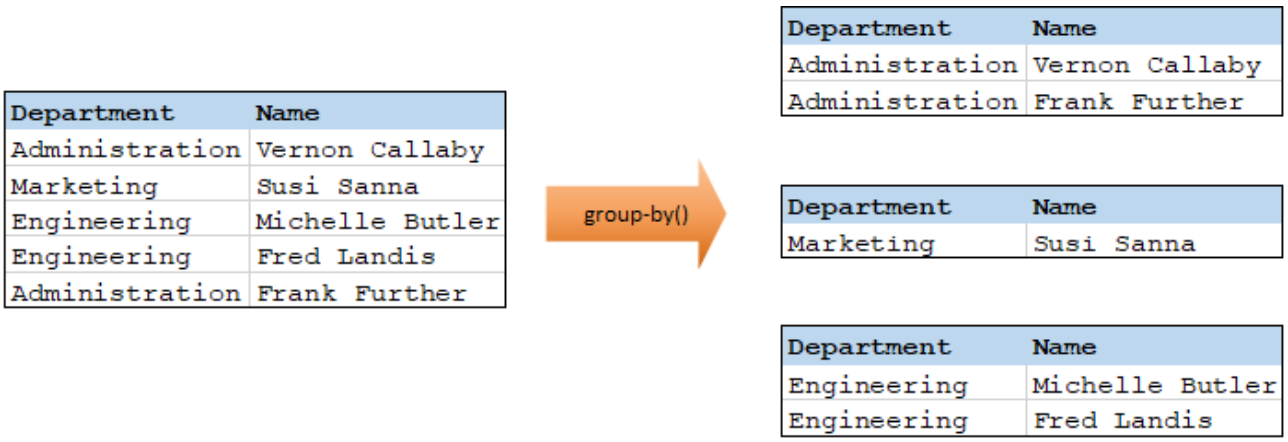
他のグループサンプルと共に、このサンプルは、次のマッピングファイルの一部です: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\GroupingFunctions.mfd**。出力 タブをクリックする前に、プレビューする関数に適用することのできる プレビュー ボタンをクリックします。

6.6.9.6 group-by

`group-by` 関数は、指定するグループキーに従いレコードのグループを作成します。



例えば、下で示される抽象的な変換では、グループキーは、「Department」です。トータルで3つの一意の部署が存在するため、`group-by` を適用すると3つのグループが作成されます:



言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

パラメーター

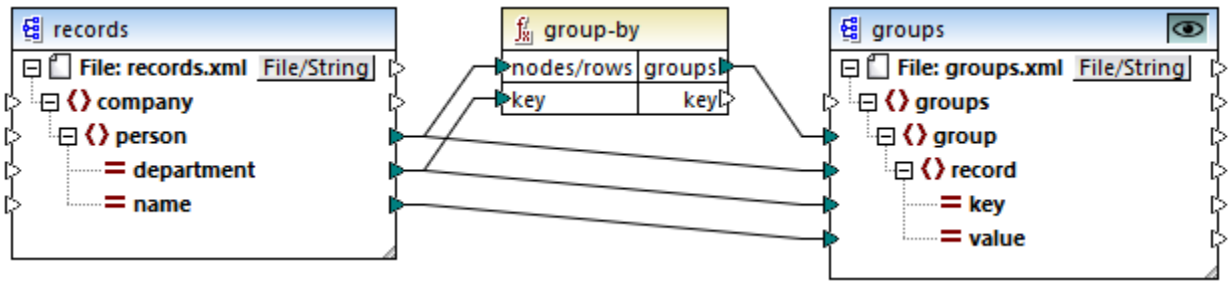
名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。
key	アイテムをグループするキー

サンプル

使用するソースデータは次のコンテンツを持つXML ファイルです（下のコードリストでは名前空間とXML 宣言は簡素化のために削除されていることに注意してください）。

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

ビジネスの条件は、部署別に個人のレコードをグループ分けします。これを達成するには、次のマッピングが`group-by` 関数を呼び出し **department** を提供します。



マッピングの結果は以下のとおりです:

```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
      <record key="Administration" value="Frank Further"/>
   </group>
   <group>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
</groups>
```

他のグループサンプルと共に、このサンプルは、次のマッピングファイルの一部です: **<マイドキュメント >\Altova\MapForce2021\MapForceExamples\Tutorial\GroupingFunctions.mfd**。**出力** タブをクリックする前に、プレビューする関数に適用することのできる **プレビュー** ボタンをクリックします。

6.6.9.7 group-ending-with

`group-ending-with` 関数はブール式の条件を引数として取ります。関数はブール式の条件がtrue の場合、条件を満たすレコードが最後になるように、新規のグループが作成されます。下のサンプルでは、「Key」が「trailing」と等価であることが条件です。3番目と5番目のレコードのために、この条件がtrue の場合、2つのグループが結果として作成されます:

メモ 条件を満たす最後のレコードの前にレコードが存在する場合、追加のグループが作成されます。例えば、「line」レコードが最後の「heading」レコードよりも多く存在する場合、これらは新しいグループ内に配置されます。

言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

パラメーター

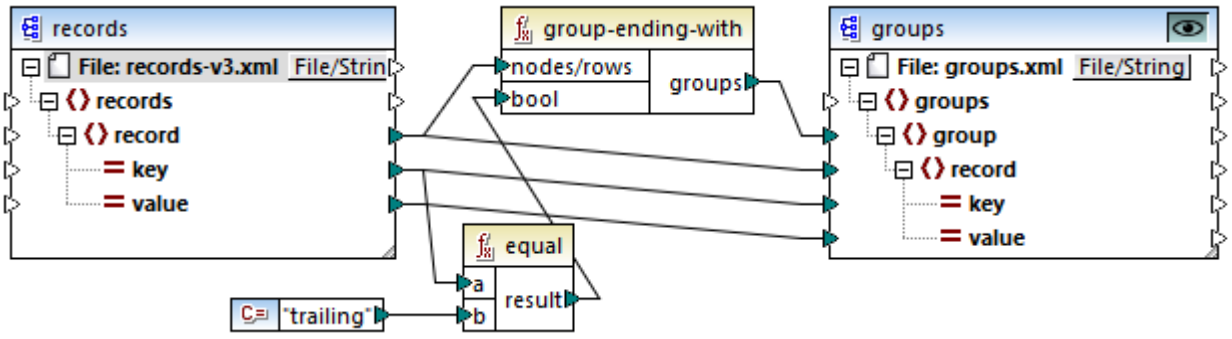
名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXMLアイテムから始まっている場合があります。
bool	**true** の場合、新規のグループを開始するブール式を提供します。

サンプル

使用するソースデータは次のコンテンツを持つXMLファイルです（下のコードリストでは名前空間とXML宣言は簡素化のために削除されていることにご注意ください）。

```
<records>
   <record key="line" value="A"/>
   <record key="line" value="B"/>
   <record key="trailing" value="Total 1"/>
   <record key="line" value="C"/>
   <record key="trailing" value="Total 2"/>
</records>
```

ビジネスの条件は、各[行末]レコードのためにグループを作成することです。各グループは[行末]レコードの前にある[line]レコードを含む必要があります。これを達成するには、次のマッピングが`group-ending-with`関数を呼び出します。下のマッピングでは、**key** 名が[行末]に等しい場合、**bool** に与えられている引数が**true** になり、新規のグループが作成されます。



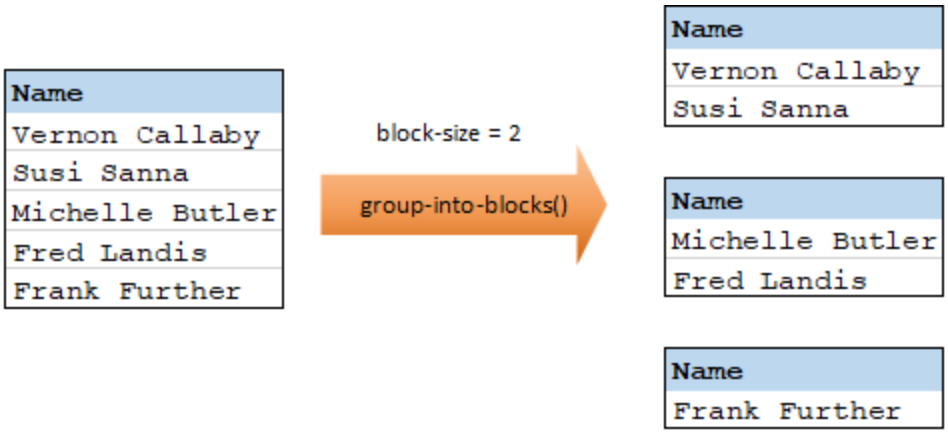
マッピングの結果は以下のとおりです:

```
<groups>
   <group>
      <record key="line" value="A"/>
      <record key="line" value="B"/>
      <record key="trailing" value="Total 1"/>
   </group>
   <group>
      <record key="line" value="C"/>
      <record key="trailing" value="Total 2"/>
   </group>
</groups>
```

他のグループサンプルと共に、このサンプルは、次のマッピングファイルの一部です: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\GroupingFunctions.mfd**。出力 タブをクリックする前に、プレビューする関数に適用することのできる プレビュー ボタンをクリックします。

6.6.9.8 group-into-blocks

group-into-blocks 関数は、N が`block-size` 引数に与えられる値である、N アイテムを含む等価のグループを作成します。最後のグループは、ソース内のアイテムの数に従い、N つのアイテム、または少ない数のアイテムを含むことができます。下のサンプルでは、`block-size` は2 です。総数で5 つのアイテムが存在し、最後のアイテムを除き、それぞれのグループは2 つのアイテムを含んでいます。



言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

パラメーター

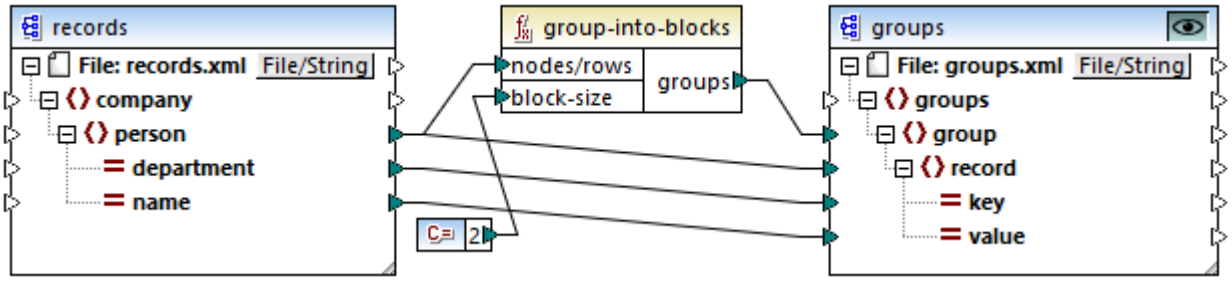
名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXMLアイテムから始まっている場合があります。
block-size	各グループのサイズを指定します。

サンプル

使用するソースデータは次のコンテンツを持つXMLファイルです（下のコードリストでは名前空間とXML宣言は簡素化のために削除されていることに注意してください）。

```
<company>
   <person department="Administration" name="Vernon Callaby"/>
   <person department="Marketing" name="Susi Sanna"/>
   <person department="Engineering" name="Michelle Butler"/>
   <person department="Engineering" name="Fred Landis"/>
   <person department="Administration" name="Frank Further"/>
</company>
```

個人のレコードを2つのアイテムを持つブロックにグループ分けすることがビジネスの必要条件とします。これを達成するために、以下のマッピングは`group-into-blocks`関数を呼び出し**block-size**として整数[2]を提供します。



マッピングの結果は以下のとおりです:

```
<groups>
   <group>
      <record key="Administration" value="Vernon Callaby"/>
      <record key="Marketing" value="Susi Sanna"/>
   </group>
   <group>
      <record key="Engineering" value="Michelle Butler"/>
      <record key="Engineering" value="Fred Landis"/>
   </group>
   <group>
```

```
      <record key="Administration" value="Frank Further"/>
   </group>
</groups>
```

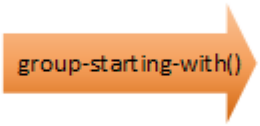
アイテムの総数（5）を 2 で割ることができないため、最後のグループに 1 つのアイテムのみ含まれていることにご注意してください。

他のグループサンプルと共に、このサンプルは、次のマッピングファイルの一部です: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\GroupingFunctions.mfd**。**出力** タブをクリックする前に、プレビューする関数に適用することのできる **プレビュー** ボタンをクリックします。

6.6.9.9 group-starting-with

group-starting-with 関数はブール式の条件を引数として取ります。関数はブール式の条件が true の場合、条件を満たすレコードから、新規のグループが作成されます。下のサンプルでは、「Key」が「heading」と等価であることが条件です。最初と4番目のレコードのために、この条件が true の場合、2 つのグループが結果として作成されます:

Key	Value
heading	Intro
line	A
line	B
heading	Body
line	C



Key	Value
heading	Intro
line	A
line	B

Key	Value
heading	Body
line	C

メモ 条件を満たす最初のレコードの前にレコードが存在する場合、追加のグループが作成されます。例えば、「line」レコードが最初の「heading」レコードより先に多く存在する場合、これらは新しいグループ内に配置されます。

言語

Built-in、C++、C#、Java、XSLT 2.0、XSLT 3.0。

パラメーター

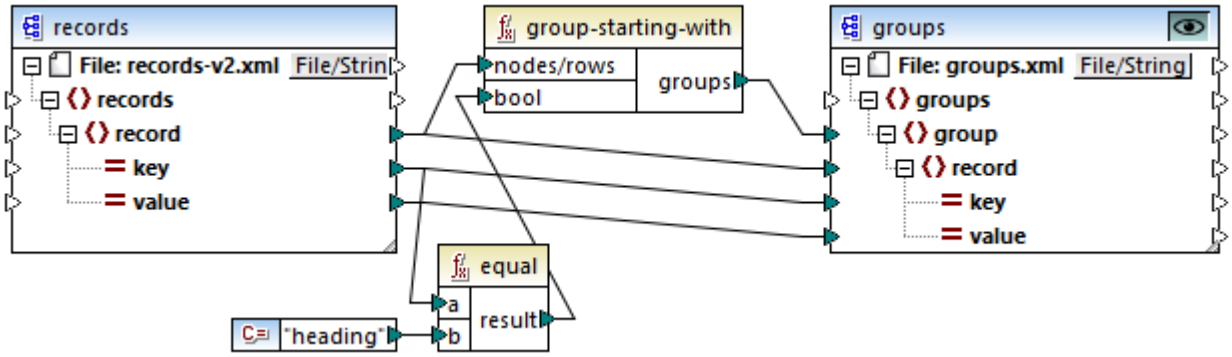
名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソース XML アイテムから始まっている場合があります。
bool	**true** の場合、新規のグループを開始するブール式を提供します。

サンプル

使用するソースデータは次のコンテンツを持つ XML ファイルです（下のコードリストでは名前空間と XML 宣言は簡素化のために削除されていることにご注意してください）。

```
<records>
   <record key="heading" value="Intro"/>
   <record key="line" value="A"/>
   <record key="line" value="B"/>
   <record key="heading" value="Body"/>
   <record key="line" value="C"/>
</records>
```

ビジネスの条件は、各 [heading] レコードのためにグループを作成することです。各グループは [heading] レコードの後にある [line] レコードを含んでいる必要があります。これを達成するには、次のマッピングが`group-starting-with` 関数を呼び出します。下のマッピングでは、**key** 名が「heading」に等しい場合、**bool** に与えられている引数が**true** になり、新規のグループが作成されます。



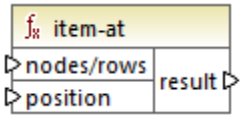
マッピングの結果は以下のとおりです:

```
<groups>
   <group>
      <record key="heading" value="Intro"/>
      <record key="line" value="A"/>
      <record key="line" value="B"/>
   </group>
   <group>
      <record key="heading" value="Body"/>
      <record key="line" value="C"/>
   </group>
</groups>
```

他のグループサンプルと共に、このサンプルは、次のマッピングファイルの一部です: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\GroupingFunctions.mfd**。**出力** タブをクリックする前に、プレビューする関数に適用することのできる **プレビュー** ボタンをクリックします。

6.6.9.10 item-at

position 引数により提供されたポジションで引数として提供された **nodes/rows** のシーケンスからアイテムを返します。最初のアイテムはポジション **1** にあります。



言語

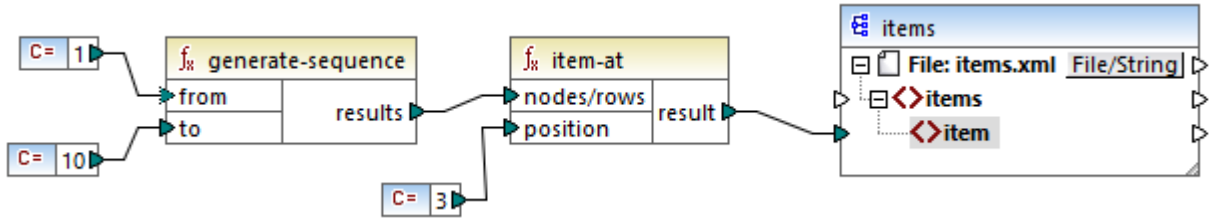
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。
position	個の整数はアイテムのシーケンスからどのアイテムが返されるかを指定します。

サンプル

次の模擬マッピングは10 の値のシーケンスを生成します。`item-at` 関数によりシーケンスは処理され、結果はターゲット XML ファイルに書き込まれます。

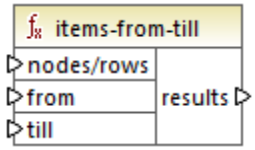


position 引数が**3** に設定されているため、シーケンスからの3 番目の値がターゲットにパスされます。そのため、（XML と名前宣言を除く）マッピングの出力は以下の通りです：

```
<items>
   <item>3</item>
</items>
```

6.6.9.11 items-from-till

[from] と[till] パラメーターを境界として使用し、**nodes/rows** のシーケンスを返します。最初のアイテムはポジション**1** にあります。



言語

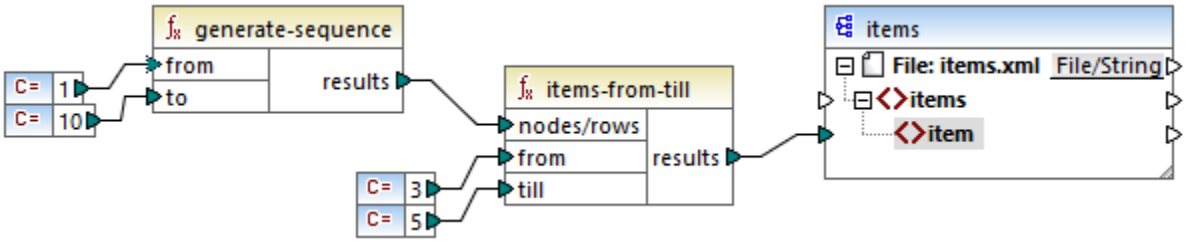
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。
from	この整数は抽出されるアイテムの開始のポジションを指定します。
till	この整数は抽出されるアイテムまでのポジションを指定します。

サンプル

次の模擬マッピングは10 の値のシーケンスを生成します。`items-from-till` 関数によりシーケンスは処理され、結果はターゲットXML ファイルに書き込まれます。

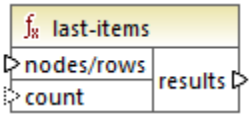


from と**till** 引数は**3** と**5** にそれぞれ設定されているため、**3** から**5** からの値のサブセットのみがターゲットにパスされます。そのため、(XML と名前宣言を除く) マッピングの出力は以下の通りです:

```
<items>
   <item>3</item>
   <item>4</item>
   <item>5</item>
</items>
```

6.6.9.12 last-items

N が**count** パラメーターにより提供される個所で、入力シーケンスの最後の *N* 個のアイテムを返します。最初のアイテムはポジション [1] にあります。



言語

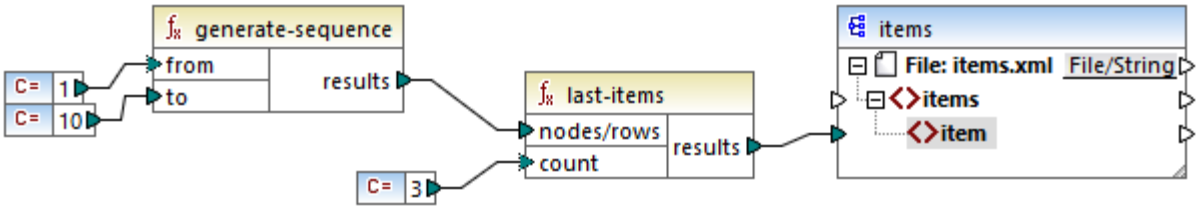
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。
count	任意のパラメーターです。入力シーケンスから抽出されるアイテムの数量を指定します。デフォルトの値は**1** です。

サンプル

次の模擬マッピングは10 の値のシーケンスを生成します。`last-items` 関数によりシーケンスは処理され、結果はターゲット XML ファイルに書き込まれます。

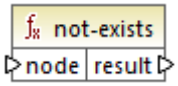


カウント引数が**3** に設定されているため、シーケンスからの最後の3 つの値がターゲットにパスされます。そのため、（XML と名前宣言を除く）マッピングの出力は以下の通りです：

```
<items>
   <item>8</item>
   <item>9</item>
   <item>10</item>
</items>
```

6.6.9.13 not-exists

接続済みのノードが存在する場合は**false** を返します。それ以外の場合は**true** を返します。この関数はexists 関数の逆ですが、使用方法は同じです。



言語

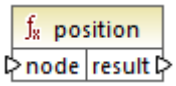
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
node	不在を確認するノード。

6.6.9.14 position

現在処理されているアイテムのシーケンス内のアイテムのポジションを返します。これは、例えば、アイテムに自動的に付番する場合に使用することができます。



言語

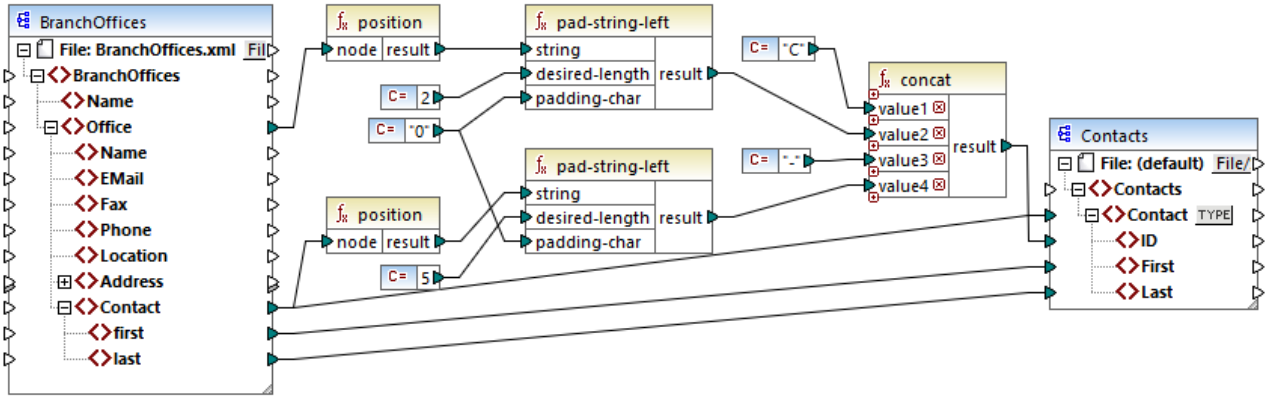
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
node	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXMLアイテムから始まっている場合があります。

サンプル

次のマッピングではマッピングにより生成されたデータ内で一意の識別地を生成するために`position`関数が使用されています。このマッピングには以下のパスで見つけることができるマッピングデザインファイルが存在します: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\ContactsFromBranchOffices.mfd**.



ContactsFromBranchOffices.mfd

上のマッピングではソースXMLファイル内に3つのブランチオフィスが含まれています。ブランチオフィスにはContact 子アイテムの任意の番号を含む場合があります。マッピングの目的は以下の通りです:

- ソースXMLファイルからすべてのContact アイテムを抽出し、ターゲット XMLファイルに書き込みます。
- 各連絡先に一意の識別番号が割り当てられる必要があります(ターゲット XML 内のID アイテム)。
- 各連絡先のID はX がオフィス番号を識別し、X が連絡先番号を識別するフォーム `CXX-YYYYY` を取る必要があります。オフィス番号が2文字以下の場合、左側はゼロで埋め込まれる必要があります。同様に、連絡先番号が5文字以下を取る場合、左側はゼロで埋め込まれる必要があります。この結果、最初のオフィスからの最初の連絡先の有効な識別番号は `C01-00001` のようになります。

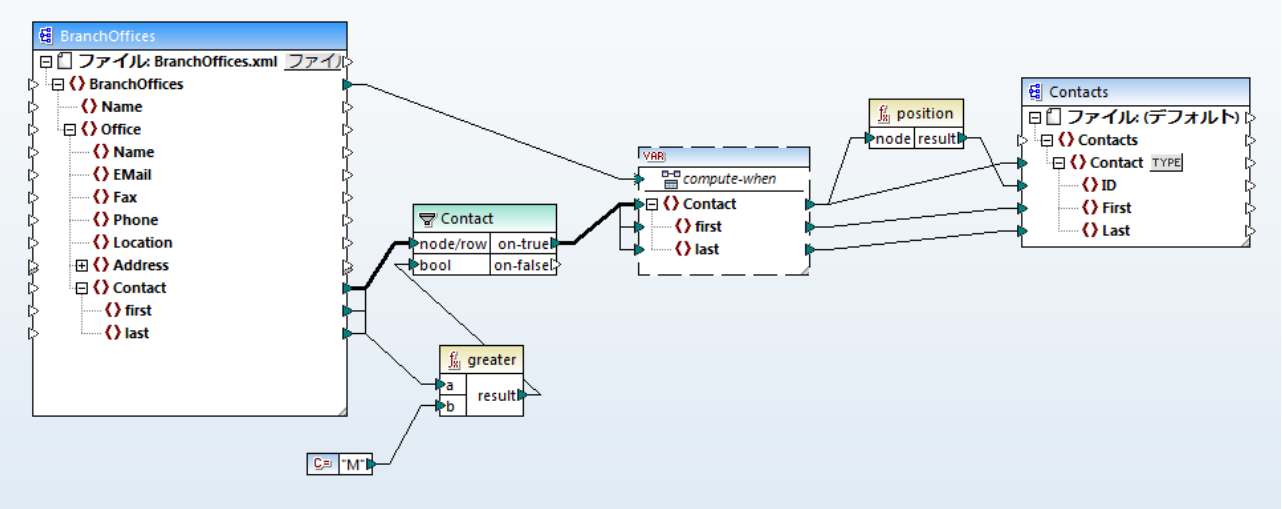
マッピングの目的を達成するには`position` 関数を含むいくつかのMapForce 関数が使用されます。上の`position` 関数は各オフィスのポジションを取得します。下の関数は各オフィスのコンテキストで各連絡先のポジションを取得します。

`position` 関数を使用する場合、現在のマッピングコンテキストを考慮してください。具体的には、マッピングの実行中に関数を介して間接的に最初のマッピングコンテキストが接続されるソースアイテムのターゲットコンポーネントのルートアイテムから確率されます。このサンプル内では上の`position` 関数は *全てのオフィスのシーケンス* を処理し、シーケンス内の最初のオフィスに対応する値 1 を最初に生成します。下の`position` 関数は(1、2、3 などの) *そのオフィスのコンテキスト* で連絡先のポジションに対応するシーケンシャル番号を生成します。次のオフィスが処理されるとこの[内部]シーケンスが設定され(このため、再度 1 から開始します)。前に記述された必要条件に従い`pad-string-left` 関数の両方はパディングを生成された番号に適用します。(ターゲット Contact へのソースからの親接続のため) `Concat` 関数は *各連絡先のコンテキスト* で作動します全ての値をジョインし、各連絡先の一意の識別番号を返します。

上のマッピングから生成有れた出力は以下に表示される通りです(いくつからのレコードは読みやすさのために削除されています):

```
<Contacts>
   <Contact>
      <ID>C01-00001</ID>
      <First>Vernon</First>
      <Last>Callaby</Last>
   </Contact>
   <Contact>
      <ID>C01-00002</ID>
      <First>Frank</First>
      <Last>Further</Last>
   </Contact>
   <!-- ...-->
   <Contact>
      <ID>C02-00001</ID>
      <First>Steve</First>
      <Last>Meier</Last>
   </Contact>
   <Contact>
      <ID>C02-00002</ID>
      <First>Theo</First>
      <Last>Bone</Last>
   </Contact>
   <!-- ...-->
</Contacts>
```

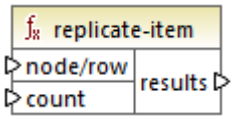
フィルターの適用後にアイテムのポジションを取得する必要がある場合があります。フィルターコンポーネントはシーケンス関数ではないため、フィルターされたアイテムのポジションを検索する関数と共には`position` 関数と共に *直接* 使用することはできません非間接的にこれは変数コンポーネントを追加することにより可能になります。例えば、下のマッピングは前のマッピングを簡素化したものです。マッピングデザインファイルは次のパスで検索することができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\PositionInFilteredSequence.mfd**.



MapForce 内の変数の結果は常にシーケンスです。このために上にマッピングでは`position` 関数は変数により作成されたシーケンスを反復し、そのシーケンス内の各アイテムのポジションを返します。サンプル: ノードのフィルターと番号付けで詳しく説明されています。

6.6.9.15 replicate-item

入力シーケンス内の全てのアイテムを **count** 引数内で指定された回数繰り返します。単一のアイテムを **node/row** 引数に接続すると、関数は、*N* が **count** 引数の値である箇所の *N* アイテムを返します。アイテムのシーケンスを **node/row** 引数に接続すると、1度に1つのアイテムを処理し、関数は、シーケンス内のそれぞれのアイテムを **count** 回繰り返します。例えば、カウントが **2** の場合、シーケンス `1,2,3` は `1,1,2,2,3,3` を生成します。下のサンプルで説明されている通り異なる **count** 値を入力シーケンス内の各アイテムに与えることができます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

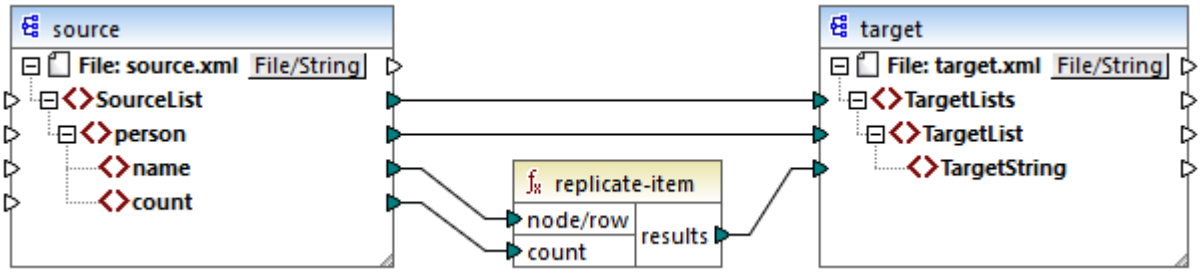
名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXMLアイテムから始まっている場合があります。
count	**node/row** に接続されている各アイテムまたはシーケンスを複数する回数を指定します。

サンプル

以下の構造を持つソースXMLファイルが存在すると想定します:

```
<SourceList>
   <person>
      <name>Michelle</name>
      <count>2</count>
   </person>
   <person>
      <name>Ted</name>
      <count>4</count>
   </person>
   <person>
      <name>Ann</name>
      <count>3</count>
   </person>
</SourceList>
```

replicate-item 関数を使用して各個人名を異なる回数ターゲットコンポーネント内で繰り返すことができます。これを達成するために各個人の**count** ノード を関数の replicate-item関数の**count** 入力に接続します:

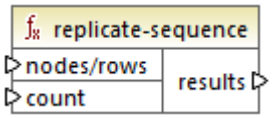


出力は以下のようになります:

```
<TargetLists>
   <TargetList>
      <TargetString>Michelle</TargetString>
      <TargetString>Michelle</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
      <TargetString>Ted</TargetString>
   </TargetList>
   <TargetList>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
      <TargetString>Ann</TargetString>
   </TargetList>
</TargetLists>
```

6.6.9.16 replicate-sequence

入力シーケンス内の全てのアイテムを **count** 引数内で指定された回数繰り返します。例えば、カウントが **2** の場合、シーケンス `1,2,3` は `1,2,3,1,2,3` を生成します。



言語

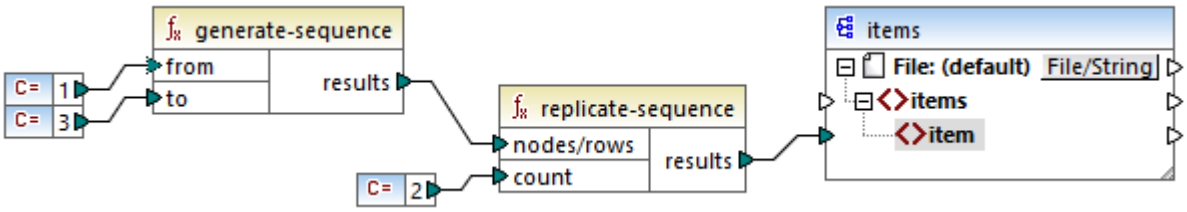
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソース XML アイテムから始まっている場合があります。
count	接続されているシーケンスを複数する回数を指定します。

サンプル

次の模擬マッピングはシーケンス `1,2,3` を生成します。`replicate-sequence` 関数によりシーケンスは処理され、結果はターゲット XML ファイルに書き込まれます。

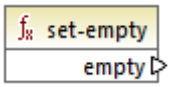


count 引数が **2** に設定されているため、シーケンスは2度複製されターゲットにパスされます。そのため、（XML と名前宣言を除く）マッピングの出力は以下の通りです：

```
<items>
   <item>1</item>
   <item>2</item>
   <item>3</item>
   <item>1</item>
   <item>2</item>
   <item>3</item>
</items>
```

6.6.9.17 set-empty

空のシーケンスを返します。

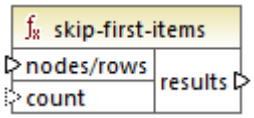


言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

6.6.9.18 skip-first-items

count 引数によりN が提供される個所で入力シーケンスの最初のN 個のアイテムをスキップし、残りのシーケンスを返します。



言語

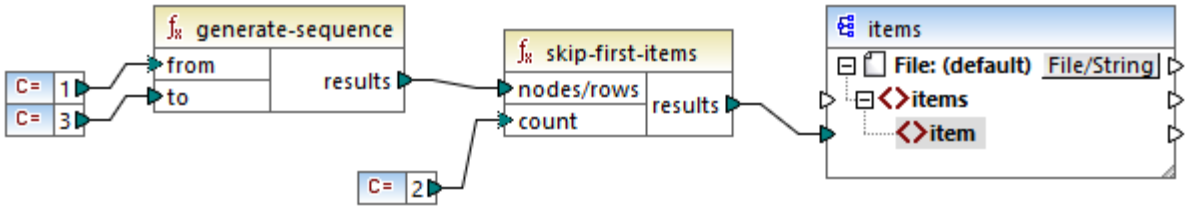
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
nodes/rows	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。
count	オプションの引数。スキップするアイテムの数量を指定します。デフォルト の値は**1** です。

サンプル

次の模擬マッピングはシーケンス`1,2,3`を生成します。`skip-first-items` 関数によりシーケンスは処理され、結果はターゲット XML ファイルに書き込まれます。

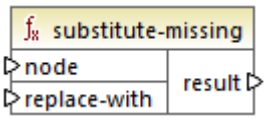


count 引数が**2** に設定されているため、最初の2 つのアイテムはスキップされ、残りのアイテムはターゲットにパスされます。そのため、（XML と名前宣言を除く）マッピングの出力は以下の通りです：

```
<items>
   <item>3</item>
</items>
```

6.6.9.19 substitute-missing

この関数はexists とif-else 条件 の組み合わせにより構成されます。**node** 入力に接続されているアイテムが存在すると、コンテンツはターゲットにコピーされます。それ以外の場合、**replace-with** 入力に接続されているアイテムのコンテンツはターゲットにコピーされます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

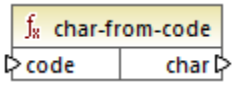
名前	説明
node	この入力はゼロまたは他の値を提供するマッピングアイテムからの接続を受け取る必要があります。例えば、接続はソースXML アイテムから始まっている場合があります。
replace-with	この入力は置換値を提供するマッピングアイテムから接続を受け取る必要があります。

6.6.10 core | string functions（文字列関数）

文字列関数はデータの一部分を取得、サブ文字列のテスト、文字列からの情報の取得、文字列の分割などを行うために文字列データを操作することができます。

6.6.10.1 char-from-code

引数として提供されている小数 Unicode 値（コード）の文字表示を返します。**ヒント**：文字のUnicode 小数コードを検索するには `code-from-char` 関数を使用することができます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

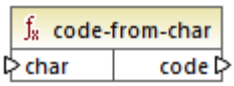
名前	説明
code	小数としてのUnicode 値。

サンプル

Unicode W eb サイトで使用可能なチャートによると（https://www.unicode.org/charts/）感嘆符文字は`0021` の16 進数値を持っています。小数フォーマットの対応する値は**33** です。このため、引数として`33` を提供すると、`char-from-code` 関数は`!` 文字を返します。

6.6.10.2 code-from-char

文字列として提供されている文字の小数 Unicode 値（コード）が返されます。文字列として提供されている文字列が複数の文字を持つ場合、最初の文字のコードが返されます。



言語

Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

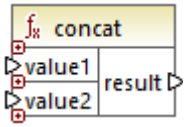
名前	説明
char	入力文字列の値。

サンプル

入力 **char** は`$`（ドルサイン）文字です。関数はこの文字のための小数 Unicode 値である）**36** を返します。

6.6.10.3 concat

2 個または以上の値を連結（追加）して、単一の結果文字列を生成します。全ての入力値は自動的に文字列型へ変換されます。デフォルトでは、この関数には2 つのパラメータのみ存在しますが、更にパラメーターを追加することができます。**パラメーターの追加**（⊡）または**パラメーターの削除**（⊠）をクリックしてパラメーターを追加または削除します。関数の引数の追加、または削除も参照してください。



メモ `concat` 関数へのすべての入力は値を必要とします。入力の1つが値を持たない場合、関数が呼び出されず、エラーが発生します。空の文字列は有効な入力値ですが、（ `set-empty` 関数の結果など）空のシーケンスは無効な値で、この結果、関数は失敗します。これを回避するには substitute-missing 関数を使用して値を最初に処理し `concat` 関数に結果を入力として提供します。

言語

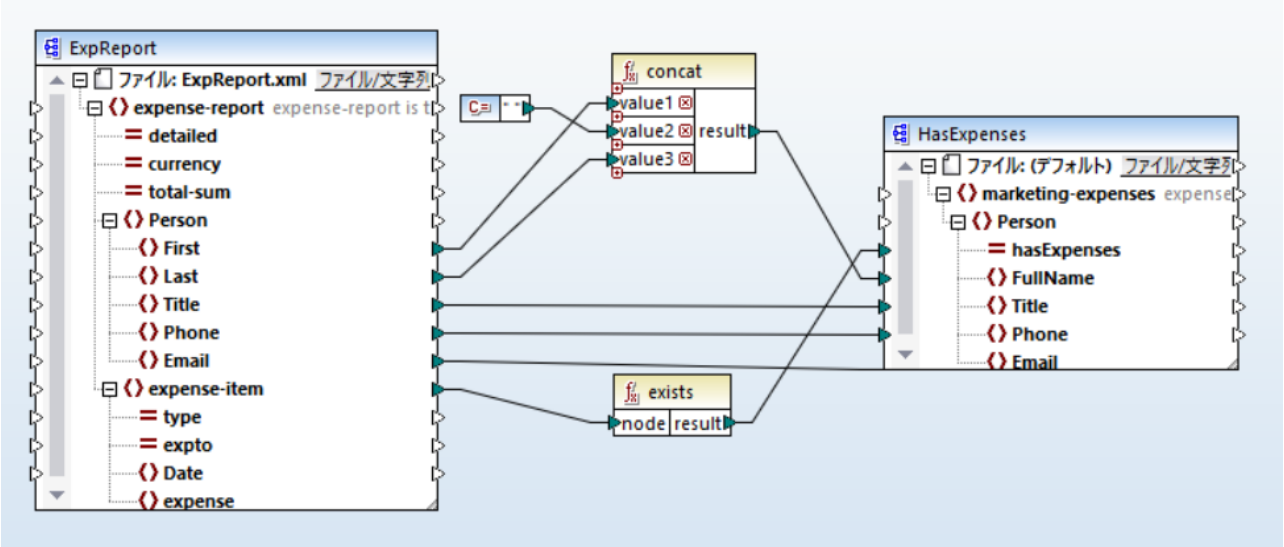
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
value1	最初の入力値。
value2	2番目の入力値。
valueN	*N* の入力値。

サンプル

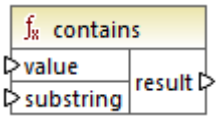
下で説明されるマッピングでは、 `concat` 関数は最初の名、定数 " "、そして姓を結合します。返される値は、 **FullName** ターゲットアイテムに書き込まれます。この関数のマッピングは次のパスで見つけることができます: **<マイドキュメント>\Altova\MapForce2021\MapForceExamples\HasMarketingExpenses.mfd**。



HasMarketingExpenses.mfd

6.6.10.4 contains

引数の値として提供されている文字列の値に引数として提供されているサブ文字列が含まれている場合、ブール値 **true** を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

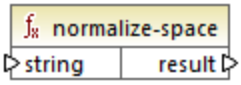
名前	説明
value	入力値（すなわち [haystack]）。
substring	（すなわち [needle]）を検索するサブ文字列。

サンプル

入力 **value** が [category] で **substring** が [cat] の場合、関数は **true** を返します。

6.6.10.5 normalize-space

正規化された入力文字列を返します。正規化は行頭および行末スペースが削除され、複数の連続した空白文字の各シーケンスが単一の空白スペースと置き換えられていることを意味します。空白スペースのための Unicode 文字は（U+0020）です。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

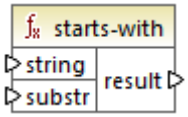
名前	説明
string	正規化するための入力文字列。

サンプル

入力文字列が `The quick  brown fox` の場合、関数は `The quick brown fox` を返します。

6.6.10.6 starts-with

引数として提供されているサブ文字列から開始する引数として文字列が提供されている場合、ブール値 **true** を返します。それ以外の場合 **false** を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

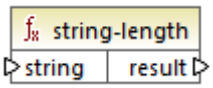
名前	説明
string	入力文字列。
substr	チェックするサブ文字列。

サンプル

入力 **string** が`category` で、**substr** が`cat` の場合、関数は**true** を返します。

6.6.10.7 string-length

引数として与えられている文字列内の文字の数量を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

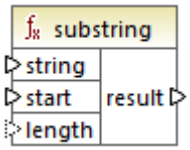
名前	説明
string	入力文字列。

サンプル

入力文字列が`car` の場合、関数は**3** を返します。入力文字列が空白の場合、関数は**0** を返します。

6.6.10.8 substring

start と**length** パラメーターにより指定されている文字列の部分を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

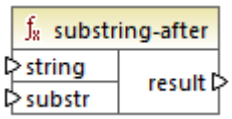
名前	説明
string	入力文字列。
start	サブ文字列が抽出される開始位置（*インデックス*）を指定します。最初の*インデックス*は**1** です。
length	任意。抽出される*アイテム*の数量を指定します。**length** パラメーターが指定されていない場合、結果は**start** から開始する文字列の最後までの*フラグメント*です。

サンプル

入力文字列が`MapForce` の場合、開始は**1** で、長さは**3** で、関数は`Map` を返します。入力文字列が`MapForce` の場合、開始は**4** で、長さが提供されていない場合、関数は`Force` を返します。

6.6.10.9 substring-after

substr の最初の発生後に発生する文字列の部分を返します。**substr** が**string** で発生しない場合、関数は空の文字列を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

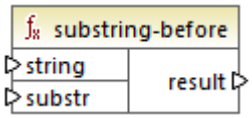
名前	説明
string	入力文字列。
substr	サブ文字列。**substr** の最初の発生の後の文字は関数の結果です。

サンプル

入力文字列が`MapForce` で**substr** が`Map` の場合、関数は`Force` を返します。入力文字列が`2020/01/04` で**substr** が`/` の場合、関数は`01/04` を返します。

6.6.10.10 substring-before

substr の最初の発生前に発生する文字列の部分を返します。**substr** が**string** で発生しない場合、関数は空の文字列を返します。



言語

Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

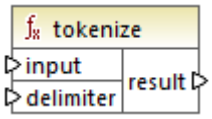
名前	説明
string	入力文字列。
substr	サブ文字列。**substr** の最初の発生の前の文字は関数の結果です。

サンプル

入力文字列が`MapForce` で**substr** が`Force` の場合、関数は`Map` を返します。入力文字列が`2020/01/04` で**substr** が`/` の場合、関数は`2020` を返します。

6.6.10.11 tokenize

引数として与えられた区切り文字を使用して入力文字列を文字列のシーケンスに分割します。



言語

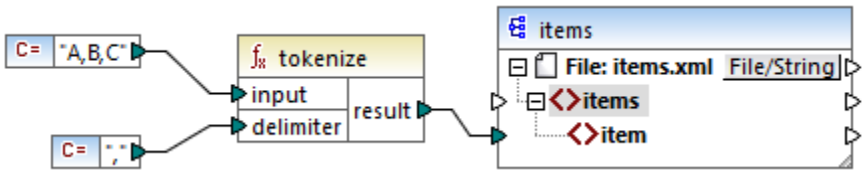
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
input	入力文字列。
delimiter	使用する区切り文字。

サンプル

入力文字列が`A,B,C` の場合、区切り文字が`,` の場合、関数は3つの文字列のシーケンスを返します: `A`、`B`、および`C`。

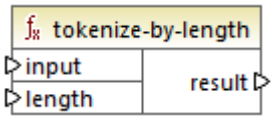


上の模擬マッピングでは関数の結果は文字列のシーケンスです。一般的なマッピング ルールによると、ソースシーケンス内の各アイテムのために新規 **item** がターゲットコンポーネント内に作成されます。この結果、マッピングの出力は以下のようになります:

```
<items>
   <item>A</item>
   <item>B</item>
   <item>C</item>
</items>
```

6.6.10.12 tokenize-by-length

入力文字列を文字列のシーケンスに分割します。結果する各文字列のサイズは**length** パラメーターにより決定されます。



言語

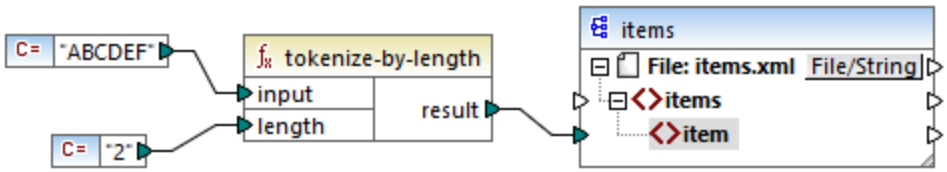
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
input	入力文字列。
length	文字列の生成されたシーケンス内の各文字列の長さを決定します。

サンプル

入力文字列が`ABCDEF` の場合、長さが**2** の場合、関数は3つの文字列のシーケンスを返します: `AB`、`CD`、および`EF`。



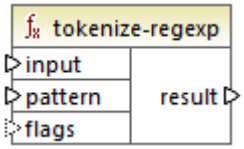
上の模擬マッピングでは関数の結果は文字列のシーケンスです。一般的なマッピング ルールによると、ソースシーケンス内の各アイテムのために新規 **item** がターゲットコンポーネント内に作成されます。この結果、マッピングの出力は以下のようになります:

```
<items>
   <item>AB</item>
   <item>CD</item>
   <item>EF</item>
</items>
```

6.6.10.13 tokenize-regexp

入力文字列を文字列のシーケンスに分割します。引数として与えられている正規表現 **pattern** に一致するサブ文字列はセパレーターを定義します。一致する(セパレーター) 文字列は関数により返された結果に含まれていません。

メモ C++、C#、またはJava コードを生成する際には、正規表現構文の高度な機能が異なる可能性があります。詳細に関しては各言語の正規表現ドキュメントを確認してください。



言語

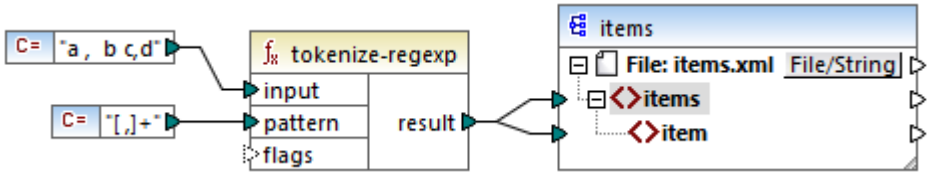
Built-in、C++、C#、Java、XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
input	入力文字列。
pattern	正規表現パターンを提供します。パターンに一致するサブ文字列は区切り文字として扱われます。正規表現に関する更に詳しい情報については正規表現のセクションを参照ください。
flags	任意のパラメーターです。使用される正規表現 flags を提供します。例えば、フラグ "i" はマッピングに大文字と小文字を区別して処理するように命令します。

サンプル

下で紹介されているマッピングのゴールは文字列 `a , b c,d` を各アルファベット文字がシーケンス内のアイテムである個所で文字列のシーケンスに分割することです。冗長空白スペースまたはコンマを削除する必要があります。



このゴールを達成するには正規表現パターン `[ ,]+` が`tokenize-regexp` 関数のパラメーターとして提供される必要があります。このパターンは以下を意味します:

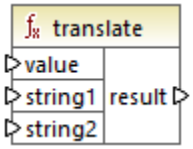
- 文字クラス `[ ,]` 内の文字のいずれかに一致します。このために、入力文字列内でコンマまたは空白文字が発生すると分割が行われます。
- 量指定子 `+` は一致する先頭の文字クラスの1つまたは複数の発生を指定します。量指定子がない場合は文字列の結果シーケンス内に各空白文字またはコンマが個別のアイテムを作成します。これは期待されない結果です。

マッピングの出力は以下のようになります:

```
<items>
   <item>a</item>
   <item>b</item>
   <item>c</item>
   <item>d</item>
</items>
```

6.6.10.14 translate

文字の置換による文字を実行します。**string1** 内に含まれている文字のために**value** を確認し **string2** 内の同じポジションにあるもののために各文字を置き換えます。**string2** に対応する文字が無い場合、その文字は削除されます。



言語

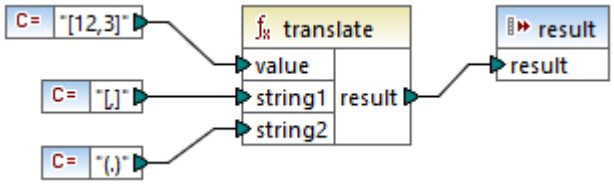
Built-in、C++、C#、Java、XQuery、XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	説明
value	入力文字列。
string1	検索文字のリストを提供します。文字列内の各文字のポジションは重要です。
string2	置換文字のリストを提供します。各置換文字のポジションは**string1** 内のものに対応する必要があります。

サンプル

文字列 `[12,3]` を `(12.3)` に変換すると仮定します。具体的には、角かっこが丸かっこと置き換えられる必要があり、コンマはピリオドと置き換えられる必要があります。これを達成するには `translate` 関数を以下のように呼び出します:



上のマッピングでは、最初の定数は処理される入力文字列を提供します。2番目と3番目の定数はそれぞれ**string1** と**string2** として文字のリストを提供します。

string1 `[,]`

string2 `(.)`

string1 と**string2** には同じ数の文字が存在することにご注意してください。**string1** 内の各文字のために、同じポジションの**string2** からの等価の文字が置換として使用されます。この結果、次の置換が発生します:

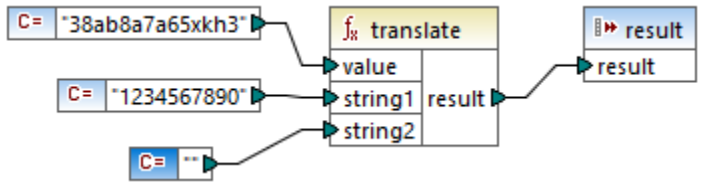
- 各 `[` は `(` と置き換えられます

- 各 , は . と置き換えられます
- 各] は] と置き換えられます

マッピングの出力は以下のようになります:

```
(12.3)
```

この関数は文字列から特定の文字を削除するために使用することができます。これを達成するには**string1** パラメーターを削除する文字に設定し、を空の文字列を **string2** に設定します。例えば下のマッピングは文字列 `38ab8a7a65xkh3` からすべての小数を削除します。



マッピングの出力は以下のようになります:

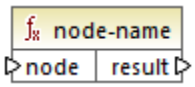
```
abaaxkh
```

6.6.11 xpath2 | accessors

xpath2 | accessors サブライブラリからの関数はXML ノードまたはアイテムに関する情報を抽出します。これらは関数はXSLT2 またはXQuery 言語が選択されている状態で利用することができます。

6.6.11.1 base-uri

`base-uri` 関数はノードを入力として取り、ノードを含むXML リソースのURI を返します。出力は型 `xs:string` 型になります。



言語

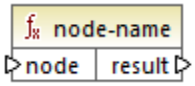
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
node	`mf:node`	入力ノード。

6.6.11.2 node-name

node-name 関数は入力パラメーターとしてノードを受け取り、そのQName を返します。QName が文字列として表示され、ノードにプレフィックスがある場合は`prefix:localname` 形式の入力を受け取り、ノードにプレフィックスが無い場合は`localname` 形式の入力が受け取られます。ノードの名前空間 URI を取得するには、(qnameに関連したfunctions 以下にある) **namespace-uri-from-QName** 関数を使用してください。



言語

XQuery、XSLT 2.0、XSLT 3.0。

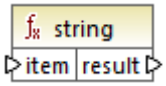
パラメーター

名前	型	説明
node	**mf:node**	入力ノード。

6.6.11.3 string

string 関数は、`xs:string` のように作動し、入力された値を`xs:string` 型に変換します。

入力が(例えば`xs:decimal` の)原子型である場合、原子型の値が`xs:string` 型の値に変換されます。入力引数がノードの場合、ノードの文字列値が抽出されます。(ノードの文字列値とはノードの子孫ノードから得られた値を連結したものになります)。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
item	**mf:item**	入力値。

6.6.12 xpath2 | anyURI functions (anyURI 関数)

xpath2 | anyURI サブライブラリは `resolve-uri` 関数を含んでいます。この関数はXSLT2 またはXQuery 言語が選択されている状態で利用することができます。

6.6.12.1 resolve-uri

`resolve-uri` 関数は最初のパラメーターとして相対的なURI を取り、2 番目の引数内のベースURI に対して解決します。結果は型 `xs:string` になります。関数の実装は両方の入力を文字列として扱います。これらのURI により識別されるリソースが実在するかチェックされません。

言語

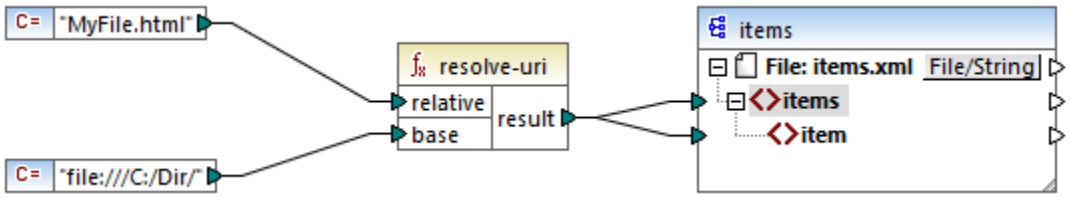
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
relative	`xs:string`	ベースに対して解決される相対的なURI。
base	`xs:string`	ベースURI。

サンプル

下で表示されているマッピング内では、最初の引数は相対的なURI `MyFile.html` を提供しており、2番目の引数はベースURI `file:///C:/Dir/` を提供しています。解決されたURI は2つを連結したもので `file:///C:/Dir/MyFile.html` になります。

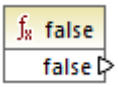


6.6.13 xpath2 | boolean functions (ブール値関数)

boolean 関数 `true` ならびに `false` 関数は引数を受け取ること無く、boolean の定数値である **true** と **false** をそれぞれ返します。定数のboolean 値が必要な場所で使用することができます。

6.6.13.1 false

boolean 値の**false** を挿入します。

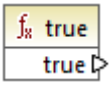


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.13.2 true

boolean 値の**true** を挿入します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.14 xpath2 | constructors

XPath 2.0 ライブラリの[constructors] サブライブラリ内の関数は入力テキストから特定のデータ型をコンストラクトします。以下のテーブルは使用可能なすべてのコンストラクタ関数をリストしています。

xs:ENTITY	xs:double	xs:nonPositiveInteger
xs:ID	xs:duration	xs:normalizedString
xs:IDREF	xs:float	xs:positiveInteger
xs:NCName	xs:gDay	xs:short
xs:NMTOKEN	xs:gMonth	xs:string
xs:Name	xs:gMonthDay	xs:time
xs:QName	xs:gYear	xs:token
xs:anyURI	xs:gYearMonth	xs:unsignedByte
xs:base64Binary	xs:hexBinary	xs:unsignedInt
xs:boolean	xs:int	xs:unsignedLong

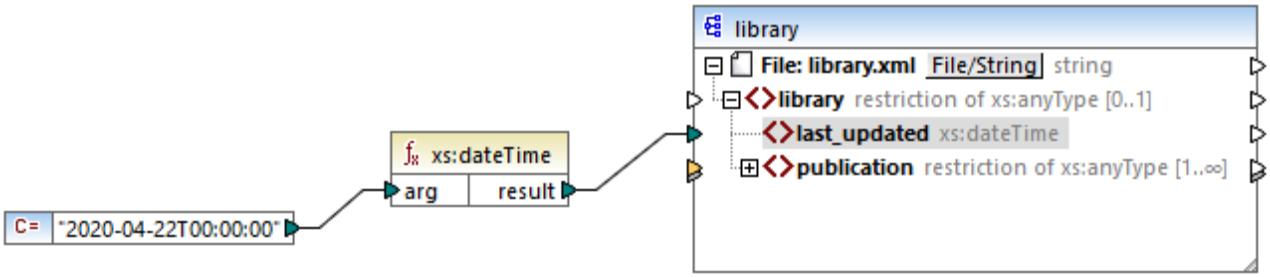
xs:byte	xs:integer	xs:unsignedShort
xs:date	xs:language	xs:untypedAtomic
xs:dateTime	xs:long	xs:yearMonthDuration
xs:dayTimeDuration	xs:negativeInteger	
xs:decimal	xs:nonNegativeInteger	

言語

XQuery、XSLT 2.0、XSLT 3.0。

サンプル

通常、入力されたテキストはコンストラクトされるデータ型に対応した書式で記述されている必要があります。そうでない場合は変換に失敗します。例えば、 xs:dateTime 値を **xs:dateTime** コンストラクタ関数を使用してコンストラクトする場合、入力テキストは xs:dateTime データ型の書式である **YYYY-MM-DDTHH:mm:ss** の形式で記述されている必要があります。



上で説明されているマッピング内では（"2020-04-28T00:00:00"）は関数の入力引数を提供するために使用されています。入力はソースドキュメント内のアイテムから取得することもできます。 **xs:dateTime** 関数は型 xs:dateTime の値 2020-04-28T00:00:00 を返します。

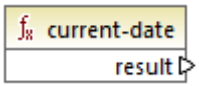
（関数引数のデータ型を含む）マッピングアイテムの期待されるデータ型を確認するには マウスカーソルを対応する入力または出力コネクタにポイントします。

6.6.15 xpath2 | context functions（コンテキスト関数）

xpath2 ライブラリからのコンテキスト関数は現在の日時のその他の情報、以前に使用されたデフォルトの照合順序、現在のシーケンスのサイズ、現在のノードのポジションを提供します。

6.6.15.1 current-date

システムクロックから得られた現在の日付（`xs:date`）を返します。

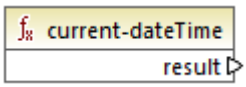


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.15.2 current-dateTime

システムクロックから得られた現在の日時（`xs:dateTime`）を返します。

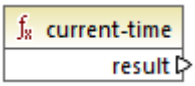


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.15.3 current-time

システムクロックから得られた現在の時刻（`xs:time`）を返します。



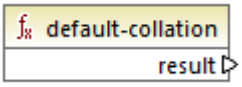
言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.15.4 default-collation

`default-collation` 関数は入力パラメーターを取らず、デフォルトの照合、つまり指定することができる関数に対して特定の照合が指定されなかった場合に使用される照合を返します。

`max-string` と**`min-string`** 関数を含む比較はデフォルトの照会順序をベースにしています。

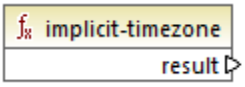


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.15.5 implicit-timezone

評価コンテキストからの[implicit timezone] プロパティの値を返します。

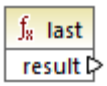


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.15.6 last

現在処理されているアイテムのシーケンス内のアイテムの数量を返します。下のサンプルで説明される通り、重要な点はアイテムのシーケンスは現在の[マッピングコンテキスト]により決定されます。



言語

XQuery、XSLT 2.0、XSLT 3.0。

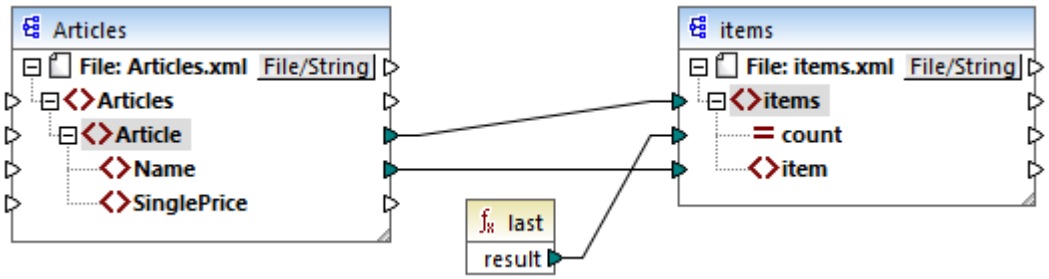
サンプル

次のソースXML ファイルが存在すると仮定します:

```
<Articles>
   <Article>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
   <Article>
      <Name>Socks</Name>
      <SinglePrice>2.30</SinglePrice>
   </Article>
```

```
   <Article>
      <Name>Jacket</Name>
      <SinglePrice>57.50</SinglePrice>
   </Article>
</Articles>
```

目的はデータを異なるスキーマを持つXML ファイルにコピーすることです。また、全てのアイテムのカウントはターゲット XML ファイルに保存される必要があります。これは以下のようなマッピング内で達成することができます:



上のサンプルでは`last` 関数は現在の親コンテキスト内と最後のノードの位置を返し、値 **3** を持つ**count** 属性を生成します。

```
<items count="3">
   <item>T-Shirt</item>
   <item>Pants</item>
   <item>Jacket</item>
</items>
```

値 **3** は**Article** と**items** の接続により作成されたマッピングコンテキスト内の最後のアイテム(すなわちすべてのアイテムのカウント)のポジションです。この接続が存在しない場合でもアイテムはターゲットにコピーされますが反復する 親コンテキスト が存在しないため`last` 関数は値 **1** を間違って返します。(具体的には、両方のコンポーネントのルートアイテム間で作成され、期待される通り3個ではなく1個のアイテムのシーケンスを作成する)デフォルトの明示的なコンテキストを使用します。

一般的には`last` 関数の代わりに**core** からの`count` 関数を使用することが奨励されます。これは前者がマッピングコンテキストを明示的に変更することのできる**parent-context** 引数を持っているからです。

6.6.16 xpath2 | durations, date and time functions (期間、日付、および時刻関数)

xpath2 ライブラリからの期間、日付、および時刻関数により日付と時刻の値内でタイムゾーンを調整することができ、特定のコンポーネントを日付、時刻、および期間値から抽出し、日付時刻の値を減算します。

タイムゾーンの調整

日付および時刻の値内でタイムゾーンを調節するには、以下の関数を使用することができます:

- `adjust-date-to-timezone`
- `adjust-date-to-timezone` (タイムゾーン引数を持つ)
- `adjust-dateTime-to-timezone`
- `adjust-dateTime-to-timezone` (タイムゾーン引数を持つ)
- `adjust-time-to-timezone`
- `adjust-time-to-timezone` (タイムゾーン引数を持つ)

これら関数はそれぞれ xs:date、xs:time、または xs:dateTime 値を最初の引数として取り、(存在する場合)2 番目のパラメーターから得られた値により、タイムゾーンの削除や編集を行います。

次のシチュエーションは最初の引数にタイムゾーンが含まれない場合のみ可能です(例えば、日付 2020-01 または時刻 14:00:00)。

- **timezone** 引数が存在する場合、結果には2番目の引数により指定されたタイムゾーンが含まれています。2番目の引数内のタイムゾーンが追加されます。
- **timezone** 引数が不在の場合、結果にはシステムのタイムゾーンである明示的なタイムゾーンが含まれます。システムのタイムゾーンが追加されます。
- **timezone** 引数が不在の場合、結果にはタイムゾーンは含まれません。

次のシチュエーションは最初の引数にタイムゾーンが含まれない場合のみ可能です(例えば、日付 2020-01-01+01:00 または時刻 14:00:00+01:00)。

- **timezone** 引数が存在する場合、結果には2番目の引数により指定されたタイムゾーンが含まれています。オリジナルのタイムゾーンが2番目の引数のタイムゾーンに置き換えられます。
- **timezone** 引数が不在の場合、結果にはシステムのタイムゾーンである明示的なタイムゾーンが含まれます。オリジナルのタイムゾーンがシステムのタイムゾーンに置き換えられます。
- **timezone** 引数が不在の場合、結果にはタイムゾーンは含まれません。

日付と時刻のコンポーネントの抽出

時刻、分数、日数、月数などの数値を日付と時刻の値から抽出するには、以下の関数を使用することができます:

- **day-from-date**
- **day-from-dateTime**
- **hours-from-dateTime**
- **hours-from-time**
- **minutes-from-dateTime**
- **minutes-from-time**
- **month-from-date**
- **month-from-dateTime**
- **seconds-from-dateTime**
- **seconds-from-time**
- **timezone-from-date**
- **timezone-from-dateTime**
- **timezone-from-time**
- **year-from-date**
- **year-from-dateTime**

これらの関数はそれぞれ xs:date、xs:time、xs:dateTime、および xs:duration 値から特定のコンポーネントを抽出します。結果は xs:integer または xs:decimal です。

期間からコンポーネントを抽出する

時刻コンポーネントを期間から抽出するには、以下の関数を使用することができます:

- **days-from-duration**
- **hours-from-duration**
- **minutes-from-duration**
- **months-from-duration**
- **seconds-from-duration**

- **years-from-duration**

期間は(年数と月数を抽出するための) `xs:yearMonthDuration` または(日数、時間数、分数、秒数を抽出するための) `xs:dayTimeDuration` として指定される必要があります。`xs:decimal` を返す **seconds-from-duration** 関数の例外と共に型 `xs:integer` の結果を返します。

日付と時刻の値の減算

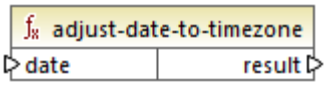
日付と時刻の値を減算するには、以下の関数を使用することができます:

- **subtract-dateTimes**
- **subtract-dates**
- **subtract-times**

それぞれの減算関数により、時間に関する値を他の値から差し引き、期間の値を得ることができます。

6.6.16.1 adjust-date-to-timezone

`xs:date` 値を評価コンテキスト(システムのタイムゾーン)内で明示的なタイムゾーンに調節します。



言語

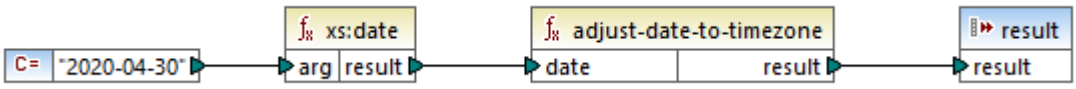
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date	**xs:date**	型 `xs:date` の入力値。

サンプル

次のマッピングは `xs:date` から文字列をコンストラクトし **adjust-date-to-timezone** 関数の引数として提供します。

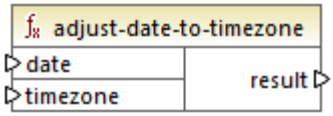


XSLT 2.0 マッピング

システムのタイムゾーンが+02:00 のコンピューターでマッピングが作動する場合、関数は日付の値がシステムのタイムゾーンを含むように調整します。この結果、マッピングの出力は `2020-04-30+02:00` になります。

6.6.16.2 adjust-date-to-timezone

`xs:date` 値を特定のタイムゾーンまたはタイムゾーン無しに調整します。**timezone** 引数が空のシーケンスの場合、関数はタイムゾーン無しで `xs:dateTime` を返します。それ以外の場合、タイムゾーン付きで `xs:date` を返します。



言語

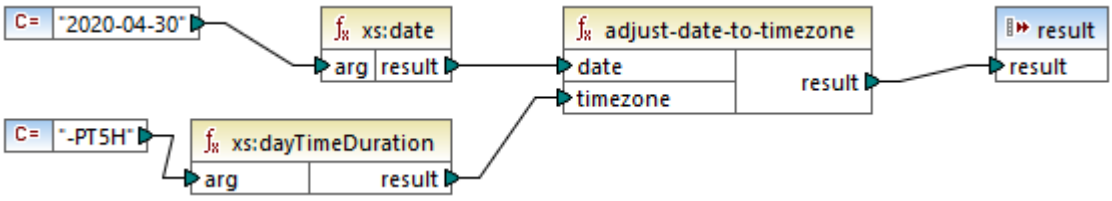
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date	`xs:date`	型 `xs:date` の入力値。
timezone	`xs:dayTimeDuration`	タイムゾーンは `xs:dayTimeDuration` 値として表示されます。この値は負の値であることができます。例えば -5 時間のタイムゾーンの値は `-PT5H` と表記することができます。

サンプル

XPath 2 constructor 関数を使用して次のマッピングは `adjust-date-to-timezone` 関数への両方のパラメーターを文字列からコンストラクトします。このマッピングの目的はタイムゾーンを -5 時間に調整することです。このタイムゾーンは `-PT5H` と表記することができます。

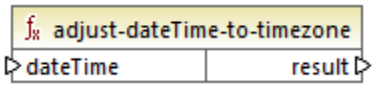


XSLT 2.0 マッピング

この関数は引数として与えられたタイムゾーンに日付の値を調整します。この結果、マッピングの出力は `2020-04-30-05:00` になります。

6.6.16.3 adjust-dateTime-to-timezone

xs:dateTime 値を評価コンテキスト(システムのタイムゾーン)内で明示的なタイムゾーンに調節します。



言語

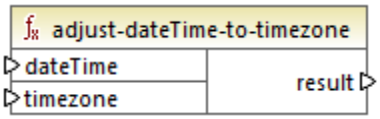
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 xs:dateTime の入力値。

6.6.16.4 adjust-dateTime-to-timezone

xs:dateTime 値を特定のタイムゾーンまたはタイムゾーン無しに調整します。**timezone** 引数が空のシーケンスの場合、関数はタイムゾーン無しで xs:dateTime を返します。それ以外の場合、タイムゾーン付きで xs:dateTime を返します。



言語

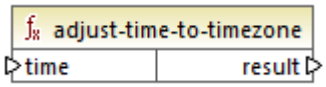
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 xs:dateTime の入力値。
timezone	**xs:dayTimeDuration**	タイムゾーンは xs:dayTimeDuration 値として表示されます。この値は負の値であることができます。例えば -5 時間のタイムゾーンの値は **-PT5H** と表記することができます。

6.6.16.5 adjust-time-to-timezone

xs:time 値を評価コンテキスト(システムのタイムゾーン)内で明示的なタイムゾーンに調節します。



言語

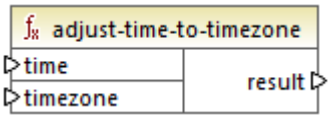
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time	**xs:time**	型 xs:time の入力値。

6.6.16.6 adjust-time-to-timezone

xs:time 値を特定のタイムゾーンまたはタイムゾーン無しに調整します。**timezone** 引数が空のシーケンスの場合、関数はタイムゾーン無しで xs:time を返します。それ以外の場合、タイムゾーン付きで xs:time を返します。



言語

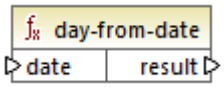
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time	**xs:time**	型 xs:time の入力値。
timezone	**xs:dayTimeDuration**	タイムゾーンは xs:dayTimeDuration 値として表示されます。この値は負の値であることができます。例えば -5 時間のタイムゾーンの値は **-PT5H** と表記することができます。

6.6.16.7 day-from-date

`xs:date` 値の日付の部分を表す `xs:integer` を返します。



言語

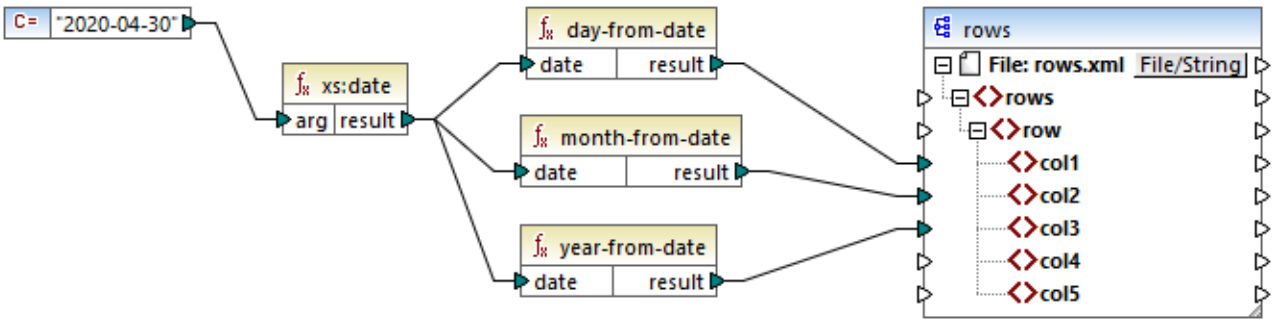
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date	**xs:date**	型 `xs:date` の入力値。

サンプル

xs:date コンストラクタ関数を使用して次のマッピングは文字列を `xs:date` に変換します。**day-from-date**、**month-from-date**、および **year-from-date** 関数は日付の関連する部分を抽出しターゲット XML ファイル内の個別のアイテムに書き込みます。



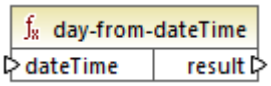
XQuery 1.0 マッピング

マッピングの出力は以下のようになります:

```
<rows>
   <row>
      <col1>30</col1>
      <col2>4</col2>
      <col3>2020</col3>
   </row>
</rows>
```

6.6.16.8 day-from-dateTime

`xs:dateTime` 値の日付の部分を表す `xs:integer` を返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 `xs:dateTime` の入力値。

6.6.16.9 days-from-duration

引数として与えられた期間の値の正規表記の日付コンポーネントの部分を表す `xs:integer` を返します。

言語

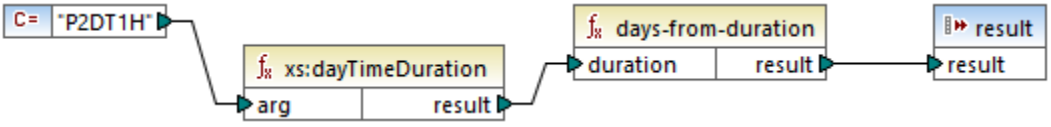
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
duration	**xs:duration**	型 `xs:duration` の入力値。

サンプル

下で表示されるマッピングは **P2DT1H**（2 日と1 時間）の `xs:dayTimeDuration` をコンストラクトし、入力として **days-from-duration** 関数に提供します。結果は **2** です。



XSLT 2.0 マッピング

メモ 期間が **P1DT24H**（1 日と24 時間）の場合、関数は **1** ではなく **2** を返します。これは **P1DT24H** の正規表現が実際 **P2D**（2 日）であるからです。

6.6.16.10 hours-from-dateTime

`xs:dateTime` 値の時間の部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**`xs:dateTime`**	型 `xs:dateTime` の入力値。

6.6.16.11 hours-from-duration

引数として与えられた期間の値の正規表記の時間コンポーネントの部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
duration	**`xs:duration`**	型 `xs:duration` の入力値。

サンプル

期間が`PT1H60M`（1 時間と60 分）の場合、関数は**1**ではなく**2**を返します。これは`PT1H60M` の正規表現が実際には`PT2H`（2 時間）であるからです。

6.6.16.12 hours-from-time

`xs:time` 値の時間の部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time	**`xs:time`**	型 `xs:time` の入力値。

6.6.16.13 minutes-from-dateTime

xs:dateTime 値の分数の部分を表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 xs:dateTime の入力値。

6.6.16.14 minutes-from-duration

引数として与えられた期間の値の正規表記の分数コンポーネントの部分を表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
duration	**xs:duration**	型 xs:duration の入力値。

サンプル

期間が **PT1M60S** (1 分 60 秒) の場合 **1** ではなく **2** を返します。これは **PT1M60S** の正規表現は **PT2M** (2 分)だからです。

6.6.16.15 minutes-from-time

xs:time 値の分数の部分を表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time	**xs:time**	型 xs:time の入力値。

6.6.16.16 month-from-date

xs:date 値の月数の部分を表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date	**xs:date**	型 xs:date の入力値。

6.6.16.17 month-from-dateTime

xs:dateTime 値の月数の部分を表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 xs:dateTime の入力値。

6.6.16.18 months-from-duration

引数として与えられた期間の値の正規表記の月数コンポーネントの部分を表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
duration	**xs:duration**	型 xs:duration の入力値。

6.6.16.19 seconds-from-dateTime

dateTime のローカライズされた値内の秒コンポーネントを表す xs:integer を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	

6.6.16.20 seconds-from-duration

引数として与えられた期間の値の正規表記の秒数コンポーネントの部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
duration	**xs:duration**	型 `xs:duration` の入力値。

6.6.16.21 seconds-from-time

引数として提供された `xs:time` 値の分数の部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time	**xs:time**	型 `xs:time` の入力値。

6.6.16.22 subtract-dateTimes

dateTime1 の正規化された値と**dateTime2** の正規化された値間の差分に対応する`xs:dayTimeDuration` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime1	**xs:dateTime**	最初の入力値。
dateTime2	**xs:dateTime**	2番目の入力値。

6.6.16.23 subtract-dates

date1 の正規化された値と**date2** の正規化された値間の差分に対応する`xs:dayTimeDuration` を返します。

言語

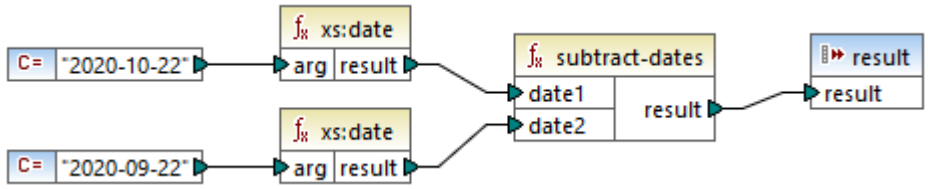
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date1	**xs:date**	最初の入力値。
date2	**xs:date**	2番目の入力値。

サンプル

下に示されているマッピングは2つの日付の減算を表しています（2020-10-22 から2020-09-22 を差し引く減算）。結果は30日を表す型 `xs:dayTimeDuration` の値 **`P30D`** です。



6.6.16.24 subtract-times

time1 の正規化された値と**time2** の正規化された値間の差分に対応する `xs:dayTimeDuration` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time1	**xs:time**	最初の入力値。
time2	**xs:time**	2番目の入力値。

6.6.16.25 timezone-from-date

引数として提供された日付のタイムゾーンコンポーネントを返します。結果はUTC からの偏差を示す `xs:dayTimeDuration` 。この値は+14:00 から-14:00 時間の両方の値が含まれる範囲です。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date	**xs:date**	型 `xs:date` の入力値。

6.6.16.26 timezone-from-dateTime

引数として提供された `xs:dateTime` 値のタイムゾーンコンポーネントを返します。結果はUTC からの偏差を示す `xs:dayTimeDuration` 。この値は+14:00 から-14:00 時間の両方の値が含まれる範囲です。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 `xs:dateTime` の入力値。

6.6.16.27 timezone-from-time

引数として提供された `xs:time` 値のタイムゾーンコンポーネントを返します。結果はUTC からの偏差を示す `xs:dayTimeDuration` 。この値は+14:00 から-14:00 時間の両方の値が含まれる範囲です。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
time	**xs:time**	型 `xs:time` の入力値。

6.6.16.28 year-from-date

`xs:date` 値の年数の部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
date	**xs:date**	型 `xs:date` の入力値。

6.6.16.29 year-from-dateTime

`xs:dateTime` 値の年数の部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
dateTime	**xs:dateTime**	型 `xs:dateTime` の入力値。

6.6.16.30 years-from-duration

引数として与えられた期間の値の正規構文表記の年数コンポーネントの部分を表す `xs:integer` を返します。

言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
duration	**xs:duration**	型 `xs:duration` の入力値。

6.6.17 xpath2 | node functions (ノード関数)

xpath2 ライブラリからのノード関数はマッピングコンポーネント上のノード (アイテム) の情報を与えます。

lang 関数は ([en] などの) 言語コードを識別する文字列の引数を取ります。関数はコンテキストノードに関数の引数に一致する値を持つ `xml:lang` 属性があるかに従い **true** または **false** を返します。

local-name、**name**、および **namespace-uri** 関数はそれぞれローカル名、名前、入力ノードの名前空間 URI を返します。例えば、ノード **altova:Products** のローカル名は **Products**、名前は **altova:Products**、名前空間 URI は altova: prefix がバウンドされる名前空間の URI です (**local-name** 関数のためのサンプルを参照してください)。これら3個の関数には2つのバリアントが存在します:

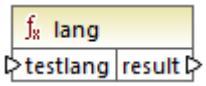
- 引数無し: 関数はコンテキストノードに適用されます (コンテキストノードのサンプルのためには **lang** 関数のために与えられたサンプルを参照してください)。
- ノードである必要がある引数: 関数は接続済みのノードに適用されます。

number 関数はノードを入力として取り、ノードを自動化し (すなわち、コンテンツを抽出し)、および値を小数に変換し、変換された値を返します。**number** 関数の2つのバリアントが存在します:

- 引数無し: 関数はコンテキストノードに適用されます (コンテキストノードのサンプルのためには **lang** 関数のために与えられたサンプルを参照してください)。
- ノードである必要がある引数: 関数は接続済みのノードに適用されます。

6.6.17.1 lang

コンテキストノードが **testlang** 引数に完全一致する、またはサブセットである値を持つ `xml:lang` 属性を持つ場合 **true** を返します。それ以外の場合関数は **false** を返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

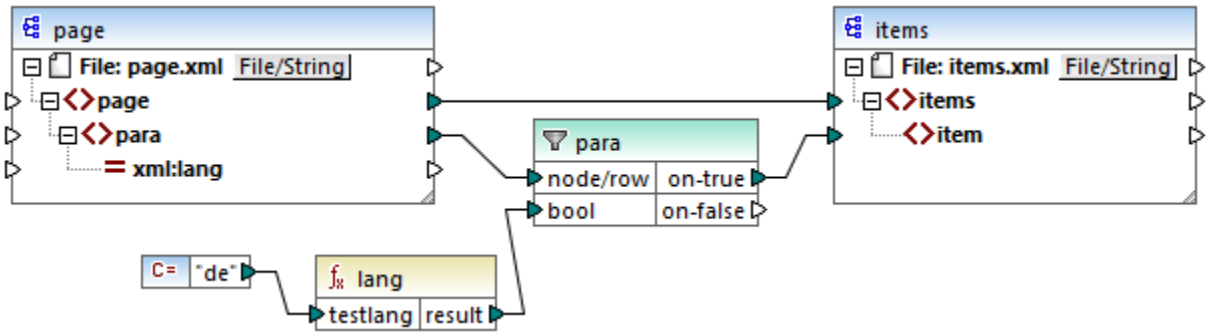
名前	型	説明
testlang	`xs:string`	チェックする言語コードは、例えば[en] です。

サンプル

以下のXML は`xml:lang` 属性のために異なる値を持つ**para** 要素を含んでいます。

```
<page>
   <para xml:lang="en">Good day!</para>
   <para xml:lang="fr">Bonjour!</para>
   <para xml:lang="de-AT">Grüss Gott!</para>
   <para xml:lang="de-DE">Guten Tag!</para>
   <para xml:lang="de-CH">Grüezi!</para>
</page>
```

下に示されてるマッピングは`lang` 関数を使用して国名のバリアントに関わらずドイツ語の段落のみをフィルターします。



XSLT 2.0 マッピング

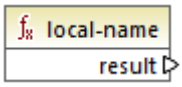
上のマッピング内では、ソース内の各 **para** 内で**item** はターゲット内に条件付けられて作成されます。`lang` 関数が**true** を返す箇所のノードのみをターゲットにパスするフィルターにより条件は提供されています。すなわち`xml:lang` 属性を[de]（[de]のサブセット）に設定しているこれらのノードはフィルターの条件を満たします。この結果、マッピングの出力は以下の通りです:

```
<items>
   <item>Grüss Gott!</item>
   <item>Guten Tag!</item>
   <item>Grüezi!</item>
</items>
```

para と**item** の間の親接続のため`lang` 関数は各 **para** のコンテキスト内で操作されます。マッピングコンテキストを参照してください。

6.6.17.2 local-name

引数として提供されたコンテキストノードの名前のローカルの部分を `xs:string` として返します。これは **local-name** 関数のパラメーター無しのバリアントです。ノードを明示的に指定するには入力ノードをパラメーターとして取る **local-name** 関数を使用してください。

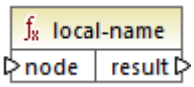


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.17.3 local-name

node の名前のローカルの部分を `xs:string` として返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

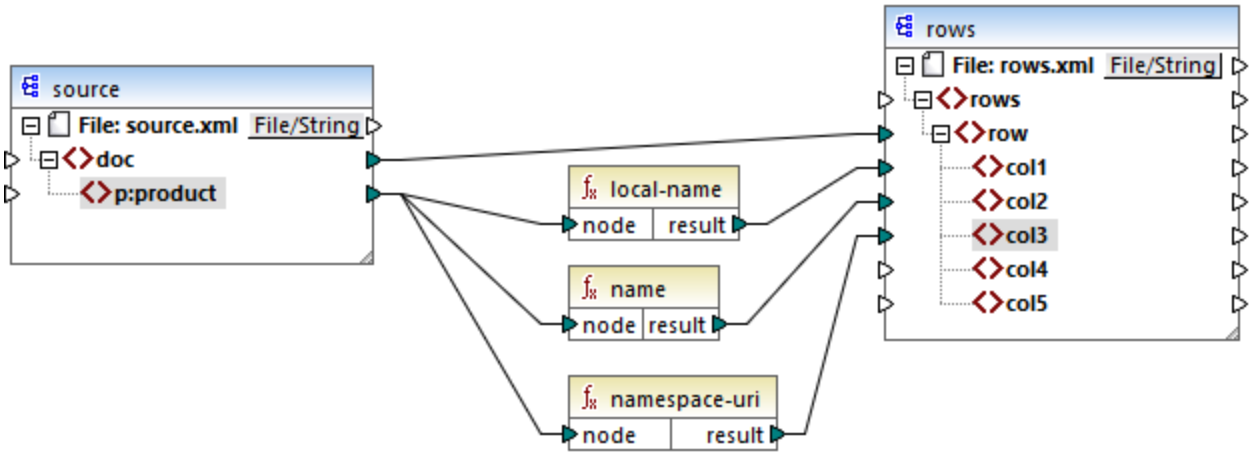
名前	型	説明
node	**node()**	入力ノード。

サンプル

次のXMLファイル内では `p:product` 要素の名前はプレフィックスされた装飾名（QName）です。プレフィックス [p] は名前空間 http://mycompany.com にマップされています。

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:p="http://mycompany.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="source.xsd">
   <p:product/>
</doc>
```

次のマッピングはローカル名、名前、およびノードの名前空間 URI を抽出し、これらの値をターゲットファイルに書き込みます:



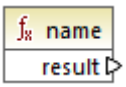
XSLT 2.0 マッピング

マッピングの出力は以下のように表示します: 各 **col** アイテムは`local-name`、`name`、および`namespace-uri` 関数の結果をリストしています。

```
<rows>
   <row>
      <col1>product</col1>
      <col2>p:product</col2>
      <col3>http://mycompany.com</col3>
   </row>
</rows>
```

6.6.17.4 name

コンテキストノードの名前を返します。これは`name` 関数のパラメーター無しのバリアントです。ノードを明示的に指定するには入力ノードを引数としてとる`name` 関数を使用してください。

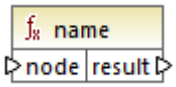


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.17.5 name

ノードの名前を返します。



言語

XQuery、XSLT 2.0。

パラメーター

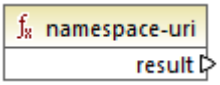
名前	型	説明
node	`node()`	入力ノード。

サンプル

`local-name` 関数のために与えられたサンプルを参照してください。

6.6.17.6 namespace-uri

コンテキストノードのQName の名前空間 URI を `xs:string` として返します。コンテキストノードがマッピング内の接続により決定される個所の`namespace-uri` 関数のパラメーター無しのバリアントです。ノードを明示的に指定するには入力ノードをパラメーターとして取る `namespace-uri` 関数を使用してください。



言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.17.7 namespace-uri

`xs:string` としての**node** のQName 名前空間 URI を返します。

言語

XQuery、XSLT 2.0。

パラメーター

名前	型	説明
node	`node()`	入力ノード。

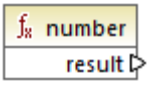
サンプル

`local-name` 関数のために与えられたサンプルを参照してください。

6.6.17.8 number

`xs:double` に変換されたコンテキストノードの値を返します。これは `number` 関数のパラメーター無しのバリアントです。ノードを明示的に指定するには入力ノードを引数として取る `number` 関数を使用してください。

数値に変換可能な方はBoolean、数値文字列、他の数値型 のみです。(非数値文字列などの)非数値入力値 は(数字ではない) NaN 内に結果します。

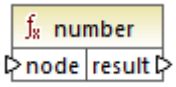


言語

XQuery、XSLT 2.0、XSLT 3.0。

6.6.17.9 number

`xs:double` に変換された **node** の値を返します。数値に変換可能な方はBoolean、数値文字列、他の数値型 のみです。(非数値文字列などの)非数値入力値 は(数字ではない) NaN 内に結果します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

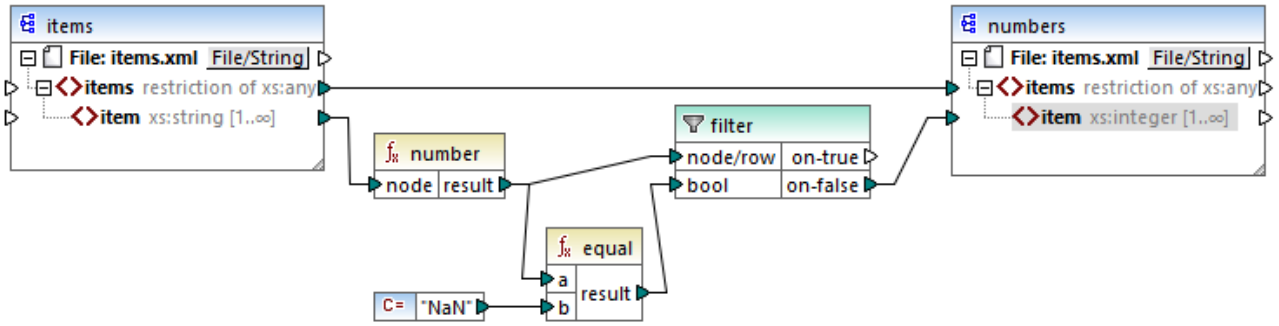
名前	型	説明
node	`mf:atomic`	入力ノード。

サンプル

次のXML は型 `string` のアイテムを含んでいます:

```
<items>
   <item>1</item>
   <item>2</item>
   <item>Jingle Bells</item>
</items>
```

下で説明されているマッピングはこれらすべての文字列を数値に変換しターゲット XML ファイルに書き込もうとします。ターゲット XML コンポーネント内の**item** のデータ型は`xs:integer` ですが、ソース**item** は`xs:string` データソースです。変換に成功しない場合、アイテムはスキップされなければならず、ターゲットファイルにコピーされません。



XSLT 2.0 マッピング

マッピングの目的を達成するためにフィルターが使用されています。`equal` 関数は変換の結果が[NaN] であるかをチェックします。これが false の場合、これは変換の成功を示し、アイテムはターゲットにコピーされます。マッピングの出力は以下のようになります:

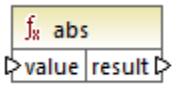
```
<items>
   <item>1</item>
   <item>2</item>
</items>
```

6.6.18 xpath2 | numeric functions (数値関数)

xpath2 ライブラリの数値関数には`abs` と`round-half-to-even` 関数が含まれます。

6.6.18.1 abs

引数として提供された絶対値を返します。例えば入力引数が**-2** または**2** の場合、関数は**2** を返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

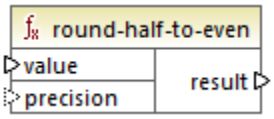
パラメーター

名前	型	説明
value	`xs:decimal`	入力値。

6.6.18.2 round-half-to-even

`round-half-to-even` 関数は任意の2番目の引数内で提供された数値（最初の引数）を精度（小数の数）の度数に提供します。例えば、最初の引数が**2.141567** で2番目の引数が**3** の場合、最初の引数（数値）は3番目の小数で端数処理され、結果は**2.141** になります。精度（2 番目パラメーター）が指定されていない場合、番号は小数が無いように端数処理され与えられた値は整数になります。

与えられている数値内の数字が2つの値の中間にある場合関数の名前内の[偶数]は数値を偶数に端数処理することを指しています。例えば、`round-half-to-even(3.475, 2)` は**3.48** という値を返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

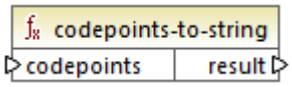
名前	型	説明
value	`xs:decimal`	端数処理される入力値を提供する必須の引数。
precision	`xs:integer`	端数処理する小数点の数値を指定する任意の引数。デフォルト の値は**0** です。

6.6.19 xpath2 | string functions (文字列関数)

xpath2 ライブラリの文字列関数により文字列を処理することができます (これには文字列の比較、文字列を大文字または小文字への変換、文字列またはその他からサブ文字列の抽出などが含まれます)。

6.6.19.1 codepoints-to-string

Unicode コードポイントのシーケンスから文字列を作成します。この関数は `string-to-codepoints` 関数の反対です。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

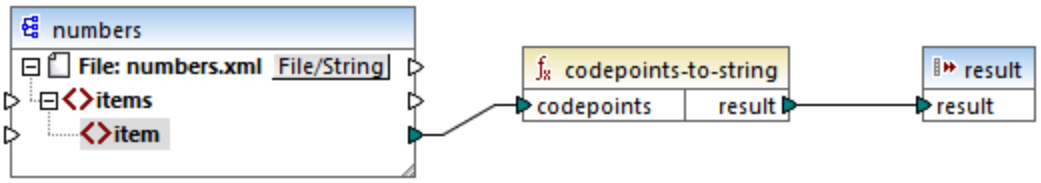
名前	型	説明
codepoints	`ZeroOrMore xs:integer`	この入力は各整数がUnicode コードポイントを指定する箇所で整数型のアイテムのシーケンスに接続されている必要があります。

サンプル

次のXML はUnicode コードポイント値を保管する複数の **item** 要素を含んでいます。

```
<items>
   <item>77</item>
   <item>97</item>
   <item>112</item>
   <item>70</item>
   <item>111</item>
   <item>114</item>
   <item>99</item>
   <item>101</item>
</items>
```

下で示されているマッピングは `codepoint-to-string` 関数への引数としてアイテムのシーケンスを提供します。



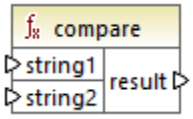
XSLT 2.0 マッピング

マッピング出力は`MapForce` です。

6.6.19.2 compare

`Compare` 関数は2つの文字列を引数としてとり、等価とアルファベット順に比較します。**string1** が**string2** よりもアルファベット順に比較して小さい場合(例えば、2つの文字列が[A] と[B] の場合) 関数は**-1** を返します。2つの文字列が等価の場合(例えば[A] と[A]) 関数は**0** を返します。**string1** が**string2** よりも大きい場合(例えば[B] と[A]) 関数は**1** を返します。

関数のこのバリアントはUnicode であるデフォルトの照会順序を使用しています。この関数の他の variant は照会順序を引数として提供可能な箇所で存在しています。



言語

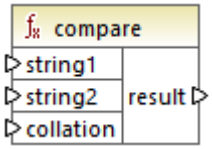
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string1	`xs:string`	最初の入力文字列。
string2	`xs:string`	2番目の入力文字列。

6.6.19.3 compare

引数として提供された照会順序を使用して `compare` 関数は2つの文字列を引数としてとり、等価とアルファベット順に比較します。**string1** が**string2** よりもアルファベット順に比較して小さい場合(例えば、2つの文字列が[A] と[B] の場合) 関数は**-1** を返します。2つの文字列が等価の場合(例えば[A] と[A])関数は**0** を返します。**string1** が**string2** よりも大きい場合(例えば[B] と[A]) 関数は**1** を返します。



言語

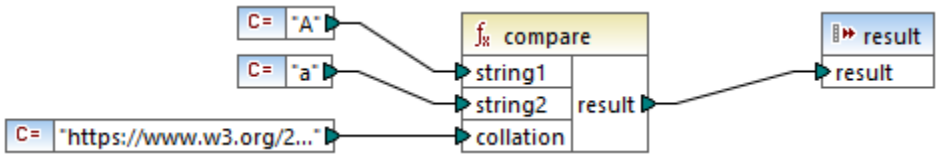
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string1	`xs:string`	最初の入力文字列。
string2	`xs:string`	2番目の入力文字列。
collation	`xs:string`	文字列比較のために使用する照会順序を指定します。この入力は`default-collation` 関数の出力を元にする可能性があります、または `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive` などの照会順序の場合があります。

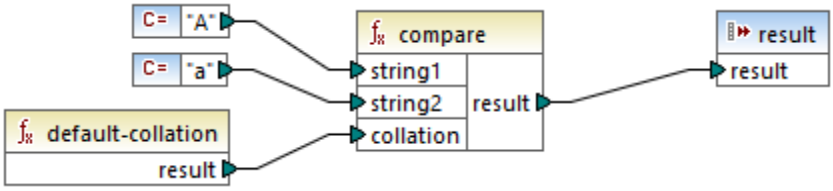
サンプル

以下のマッピングは文字列 [A] と[a] を定数により提供される大文字と小文字を区別した照会順序 `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive` を使用して比較しています。



XSLT 2.0 マッピング

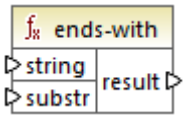
上のマッピングの結果は **0** です。これは両方の文字列が同様に扱われることを意味します。`default-collation` 関数により与えられている照会順序を置き換えると照会順序はデフォルトのUnicode コードポイント照会順序に変更され、マッピングの結果は **-1** になります。([A] は[a] よりもアルファベット順では小さいからです)。



6.6.19.4 ends-with

string が**substr** で終わる場合は**true** を返し、それ以外の場合は**false** を返します。戻り値は`xs:boolean` です。

関数のこのバリアントはUnicode であるデフォルトの照会順序を使用しています。この関数の他のvariant は照会順序を引数として提供可能な箇所で存在しています。



言語

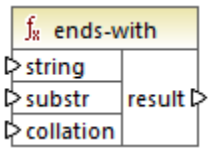
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string	`xs:string`	入力文字列（すなわち[haystack]）。
substr	`xs:string`	サブ文字列（すなわち[needle]）。

6.6.19.5 ends-with

string が**substr** で終わる場合は**true** を返し、それ以外の場合は**false** を返します。戻り値は`xs:boolean` です。



言語

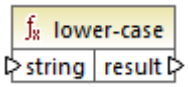
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string	`xs:string`	入力文字列（すなわち [haystack]）。
substr	`xs:string`	サブ文字列（すなわち [needle]）。
collation	`xs:string`	文字列比較のために使用する照会順序を指定します。この入力は `default-collation` 関数の出力を元にする可能性があります、または `http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive` などの照会順序の場合があります。

6.6.19.6 lower-case

全ての文字を対応する小文字に変換後 **string** の値を返します。



言語

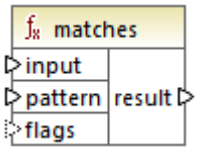
XQuery、XSLT 2.0。

パラメーター

名前	型	説明
string	`xs:string`	入力値。

6.6.19.7 matches

`matches` 関数により、(input パラメーターにより) 与えられた文字列が、(pattern パラメーターから得られる) 正規表現にマッチするかがチェックされます。正規表現の構文は、XML スキーマの `pattern` ファセットにて定義する必要があります。文字列が正規表現にマッチすれば **true** が返され、それ以外の場合は **false** が返されます。



言語

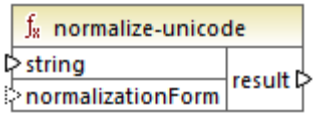
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
input	`xs:string`	入力文字列。
pattern	`xs:string`	一致する正規表現に関しては正規表現を参照してください。
flags	`xs:string`	一致に影響を与える任意の引数。この引数は以下のフラグの任意の組み合わせを与える場合があります: `i`、`m`、`s`、`x`。例えば `imx` のように、複数のフラグを使用することもできます。フラグが使用されていない場合、4 つのフラグのデフォルト値が使用されます。4 種類あるフラグは以下の通りです: `i` 大文字と小文字を区別するモードを使用します。デフォルトでは大文字と小文字を区別します。 `m` 複数行モードを使用して、改行文字 (`x0a`) により複数行に分けられている入力文字列を、1 つの文字列として扱います。メタ文字の `^` と `$` が、それぞれ行頭と行末を表します。デフォルトは文字列モードになっており、個々の文字列が `^` から始まり `$` で終わります。 `s` dot-all モードを使用します。デフォルトではメタ文字は新規のライン文字 (`x0a`) 以外のすべての文字に一致する not-dot-all モードです。dot-all mode ではドットはすべての新規ライン文字に一致します。 `x` 空白文字を無視する。デフォルトでは空白文字は無視されません。

6.6.19.8 normalize-unicode

指定されている正規化書式のルールに従い正規化されている **string** の値 (2番目の引数) を返します。Unicode 正規化に関する詳細は §2.2 of https://www.w3.org/TR/charmod-norm/ を参照してください。



言語

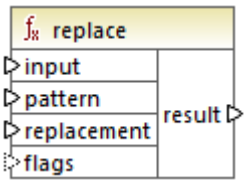
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string	`xs:string`	正規化される文字列の値
normalizationForm	`xs:string`	正規化フォームを提供するオプションの引数。デフォルトはUnicode 正規化形式 C（NFC）です。 フォームNFC、NFD、NFKC、および NFKD の正規化は、、サポートされています。

6.6.19.9 replace

この関数は入力文字列、正規表現、置換文字列を引数としてとります。置換文字列を使用して入力文字列内の正規表現のすべての一致を置き換えます。入力文字列内で正規表現が2つのオーバーラップする文字列に一致する場合、最初の一致のみが置き換えられます。



言語

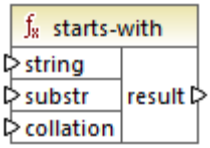
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
input	`xs:string`	入力文字列。
pattern	`xs:string`	一致する正規表現に関しては正規表現を参照してください。
replacement	`xs:string`	置換文字列。
flags	`xs:string`	一致に影響を与える任意の引数。この引数は`matches` 関数内の**flags** 引数と同じ方法で使用されます。

6.6.19.10 starts-with

string が**substr** で始まる場合は**true** を返し、それ以外の場合は**false** を返します。戻り値は`xs:boolean` です。文字列の比較は指定されている照会順序に従い行われます。



言語

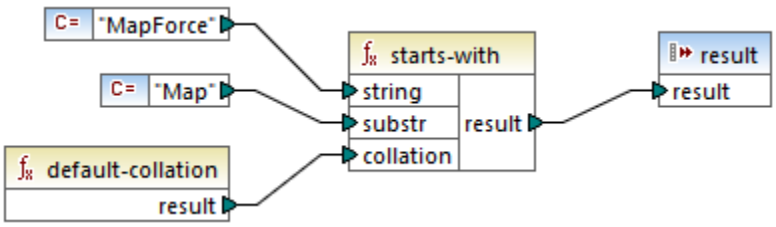
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string	`xs:string`	入力文字列（すなわち[haystack]）。
substr	`xs:string`	サブ文字列（すなわち[needle]）。
collation	`xs:string`	文字列比較のために使用する照会順序を指定します。この入力は`default-collation` 関数の出力を元にする可能性があります、または`http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive` などの照会順序の場合があります。

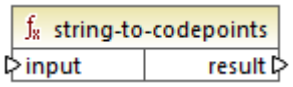
サンプル

デフォルトのUnicode 照会順序が使用されるため入力文字列 [MapForce] はサブ文字列 [Map] から開始するため次のマッピングは値 `true` を返します。



6.6.19.11 string-to-codepoints

引数として提供された文字列を構成するUnicode コードポイント(整数値) のシーケンスを返します。この関数は`codepoints-to-string` 関数の反対です。



言語

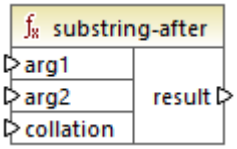
XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
input	`xs:string`	入力文字列。

6.6.19.12 substring-after

文字列 **arg2** の後に発生する文字列 **arg1** の部分を返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
arg1	`xs:string`	入力文字列 (すなわち [haystack])。
arg2	`xs:string`	サブ文字列 (すなわち [needle])。
collation	`xs:string`	文字列比較のために使用する照会順序を指定します。この入力は`default-collation` 関数の出力を元にする可能性があります、または `http://www.w3.org/2005/xpath-functions/collation/html-`

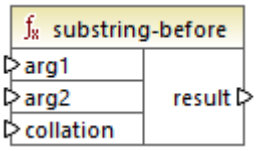
名前	型	説明
		ascii-case-insensitive などの照会順序の場合があります。

サンプル

arg1 が[MapForce]、**arg2** が[Map]の場合、**collation** は`default-collation`で、関数は[Force]を返します。

6.6.19.13 substring-before

文字列 **arg2** の前に発生する文字列 **arg1** の部分を返します。



言語

XQuery、XSLT 2.0、XSLT 3.0。

パラメーター

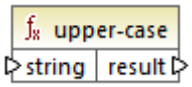
名前	型	説明
arg1	`xs:string`	入力文字列（すなわち[haystack]）。
arg2	`xs:string`	サブ文字列（すなわち[needle]）。
collation	`xs:string`	文字列比較のために使用する照会順序を指定します。この入力は`default-collation`関数の出力を元にする可能性があります、または http://www.w3.org/2005/xpath-functions/collation/html-ascii-case-insensitive などの照会順序の場合があります。

サンプル

arg1 が[MapForce]、**arg2** が[Force]の場合 **collation** は`default-collation`で、関数は[Map]を返します。

6.6.19.14 upper-case

全ての文字を対応する大文字に変換後 **string** の値を返します。



言語

XQuery、XSLT 2.0。

パラメーター

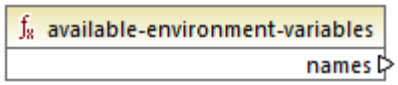
名前	型	説明
string	`xs:string`	入力文字列。

6.6.20 xpath3 | external information functions

xpath3 ライブラリの外部情報関数はXSLT 実行環境に関する情報の取得、または外部リソースからのデータの抽出を可能にします。

6.6.20.1 available-environment-variables

`environment-variable` 関数に適切な環境変数名のリストを文字列の(空である可能性のある) シーケンスで返します。

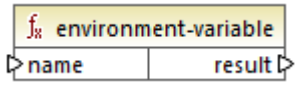


言語

XSLT 3.0。

6.6.20.2 environment-variable

存在する場合システム環境関数の値を返します。戻り型は `xs:string` です。



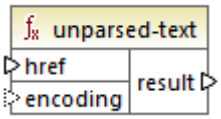
言語
XSLT 3.0。

パラメーター

名前	型	説明
name	`xs:string`	環境変数の名前

6.6.20.3 unparsed-text

外部リソース（例えばファイル）を読み取り、リソースの文字列表記を返します。



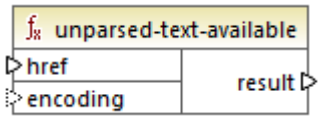
言語
XSLT 3.0。

パラメーター

名前	型	説明
href	`xs:string`	URI レファレンスのフォーム内の文字列
encoding	`xs:string`	オプションの引数。エンコードの名前を指定します。例えば"UTF-8"、""UTF-16"など。エンコードが自動的に決定されない場合 UTF-8 が想定されます。

6.6.20.4 unparsed-text-available

特定の引数を持つ`unparsed-text` への呼び出しが成功するかを決定します。戻り型は`xs:boolean` です。



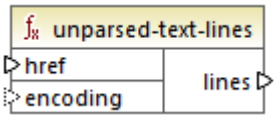
言語
XSLT 3.0。

パラメーター

名前	型	説明
href	xs:string	URI レファレンスのフォーム内の文字列
encoding	xs:string	オプションの引数。エンコードの名前を指定します。例えば "UTF-8"、""UTF-16" など。エンコードが自動的に決定されない場合 UTF-8 が想定されます。

6.6.20.5 unparsed-text-lines

外部リソース（例えば、ファイル）を読み取り、そのコンテンツを文字列のシーケンスとして返します。リソースの文字列表記がテキストのそれぞれのラインにつき1個表示されます。



言語

XSLT 3.0。

パラメーター

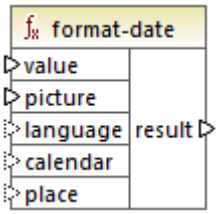
名前	型	説明
href	xs:string	URI レファレンスのフォーム内の文字列
encoding	xs:string	オプションの引数。エンコードの名前を指定します。例えば "UTF-8"、""UTF-16" など。エンコードが自動的に決定されない場合 UTF-8 が想定されます。

6.6.21 xpath3 | formatting functions

xpath3 ライブラリを使用可能な書式設定関数は日付、時刻、および整数の値を書式設定するために使用されます。

6.6.21.1 format-date

表示のために`xs:date`値を含む文字列を返します。



言語

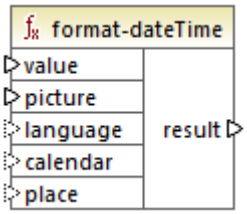
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:date`	書式設定する入力`xs:date`値です。必須のパラメーターです。
picture	`xs:string`	必須のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 9.8.4.1 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
language	`xs:string`	任意のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 9.8.4.8 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
カレンダー	`xs:string`	上記と同様です。
place	`xs:string`	上記と同様です。

6.6.21.2 format-dateTime

表示のために`xs:dateTime` 値を含む文字列を返します。



言語

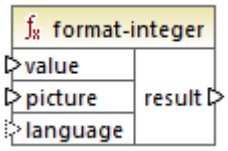
XSLT 3.0。

パラメーター

名前	型	説明
value	**xs:dateTime**	書式設定する入力 `xs:dateTime` 値です。
picture	**xs:string**	必須のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 9.8.4.1 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
language	**xs:string**	任意のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 9.8.4.8 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
カレンダー	**xs:string**	上記と同様です。
place	**xs:string**	上記と同様です。

6.6.21.3 format-integer

指定されている場合与えられた自然言語の変換を使用して与えられた文字列に従い整数を書式設定します。



言語

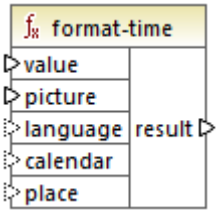
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:integer`	書式設定する入力整数値
picture	`xs:string`	必須のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 4.6.1 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
language	`xs:string`	任意のパラメーターです。 値が書式設定するために従う自然言語を設定します。指定される場合、この値は空の文字列または "拡張マークアップ言語 (XML) 1.0 W 3C 奨励 (https://www.w3.org/TR/xml) に従った`xml:lang` 属性のために許可される値になります。according to the.

6.6.21.4 format-time

表示のために xs:time 値を含む文字列を返します。



言語

XSLT 3.0。

パラメーター

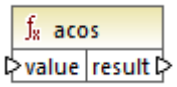
名前	型	説明
value	xs:time	書式設定する入力 xs:time 値です。
picture	xs:string	必須のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 9.8.4.1 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
language	xs:string	任意のパラメーターです。 "XPath とXquery 関数と演算子 3.1" W 3C 推奨のセクション 9.8.4.8 を参照してください (https://www.w3.org/TR/xpath-functions-31)。
カレンダー	xs:string	上記と同様です。
place	xs:string	上記と同様です。

6.6.22 xpath3 | math functions

xpath3 ライブラリの数学関数は三角および他の数学計算を行うために使用されます。

6.6.22.1 acos

0 から **pi** までの範囲で角度のアークコサインを返します。



言語

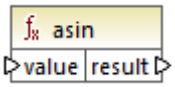
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.2 asin

-pi/2 から **pi/2** までの範囲で角度のアークサインを返します。



言語

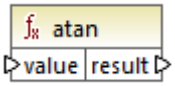
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.3 atan

-pi/2 から **pi/2** までの範囲で角度のアークタンジェントを返します。



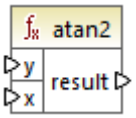
言語
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.4 atan2

座標 (x, y) と正 x-軸を持つ平面乗の原点に対しての角度をラジアンで返します。



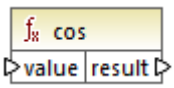
言語
XSLT 3.0。

パラメーター

名前	型	説明
y	`xs:double`	x 座標。
x	`xs:double`	y 座標。

6.6.22.5 cos

value により与えられた角度の三角コサインを返します。値のユニットはラジアンです。



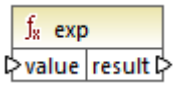
言語
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.6 exp

value に累乗したオイラー数 e が返されます。



言語

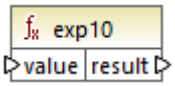
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.7 exp10

10の値乗を返します。



言語

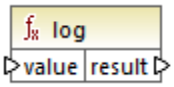
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.8 log

value の(底がe の) 自然対数を返します。



言語

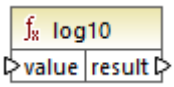
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.9 log10

value の(底が10 の) 10 十進対数を返します。



言語

XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

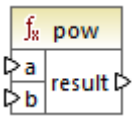
6.6.22.10 pi

数理定数 **pi** の近似値を返します。

言語
XSLT 3.0。

6.6.22.11 pow

b に累乗された **a** の値を返します。



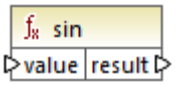
言語
XSLT 3.0。

パラメーター

名前	型	説明
a	`xs:double`	入力値 **a**。
b	`xs:double`	入力値 **b**。

6.6.22.12 sin

value により与えられた角度の三角サインを返します。値のユニットはラジアンです。



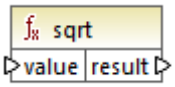
言語
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.13 sqrt

引数の負以外平方根を返します。



言語

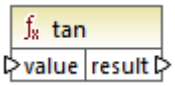
XSLT 3.0。

パラメーター

名前	型	説明
value	`xs:double`	入力値。

6.6.22.14 tan

value により与えられた角度の三角タンジェントを返します。値のユニットはラジアンです。



言語

XSLT 3.0。

パラメーター

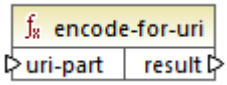
名前	型	説明
value	`xs:double`	入力値。

6.6.23 xpath3 | URI functions

xpath3 ライブラリ内のURI 関数は、URI 内での使用を目的としたエンコード、エスケープおよび変換を行います。

6.6.23.1 encode-for-uri

URI のパスセグメント内で使用される予定の文字列内の予約された文字をエンコードします。この関数の詳細については、"XPath と Xquery 関数と演算子 3.1" W 3C 推奨のセクション 6.2 を参照してください(https://www.w3.org/TR/xpath-functions-31)。



言語

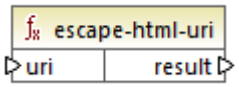
XSLT 3.0。

パラメーター

名前	型	説明
uri-part	`xs:string`	エンコードする入力 URI 値。

6.6.23.2 escape-html-uri

HTML ユーザーエージェントが URI を含むことを期待される属性値を処理する方法と同じ方法で URI をエスケープします。この関数の詳細については、"XPath と Xquery 関数と演算子 3.1" W 3C 推奨のセクション 6.4 を参照してください (https://www.w3.org/TR/xpath-functions-31)。



言語

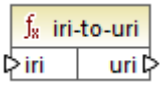
XSLT 3.0。

パラメーター

名前	型	説明
uri	`xs:string`	エスケープする入力 URI 値。

6.6.23.3 iri-to-uri

IRI (国際リソース識別子) を含む文字列を URI (統一資源識別子)に変換します。この関数の詳細については、"XPath と Xquery 関数と演算子 3.1" W 3C 推奨のセクション 6.3 を参照してください(https://www.w3.org/TR/xpath-functions-31)。



言語

XSLT 3.0。

パラメーター

名前	型	説明
iri	`xs:string`	入力 IRI 値。

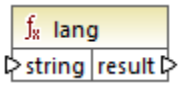
6.6.24 xslt | xpath functions (xpath 関数)

このサブグループ内の関数はマッピングアイテム(またはノード)に関する情報を抽出する XPath 1.0 関数です。関数の多くはノードを引数として取り、そのノードに関する情報を返します。The `last` と`position` 関数はマッピング上の接続により決定される現在のマッピングコンテキスト 内で作動します。

メモ その他の XPath 1.0 関数を **core** 関数 ライブラリで確認することもできます。

6.6.24.1 lang

コンテキストノードが**string** 引数に完全一致する、またはサブセットである値を持つ `xml:lang` 属性を持つ場合 **true** を返します。それ以外の場合関数は**false** を返します。



言語

XSLT 1.0。

パラメーター

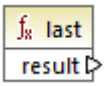
名前	型	説明
string	`xs:string`	チェックする言語コードは、例えば[en] です。

サンプル

xpath2 ライブラリの `lang` 関数のために与えられたサンプルを参照してください。

6.6.24.2 last

処理されたノードリスト内の最後のノードのポジション番号を返します。



言語

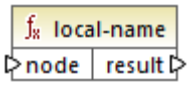
XSLT 1.0。

サンプル

xpath2 ライブラリの`last` 関数のために与えられたサンプルを参照してください。

6.6.24.3 local-name

引数として提供されたノードの名前のローカルの部分を返します。



言語

XSLT 1.0、XSLT 2.0。

パラメーター

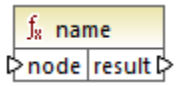
名前	型	説明
node	`node()`	入力ノード。

サンプル

xpath2 ライブラリの`local-name` 関数のために与えられたサンプルを参照してください。

6.6.24.4 name

引数として提供されたノードの名前を返します。



言語
XSLT 1.0、XSLT 2.0。

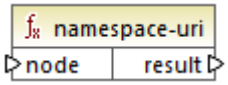
パラメーター

名前	型	説明
node	node()	入力ノード。

サンプル
xpath2 ライブラリの local-name 関数のために与えられたサンプルを参照してください。

6.6.24.5 namespace-uri

引数として提供されたノードの名前空間 URI を返します。



言語
XSLT 1.0、XSLT 2.0。

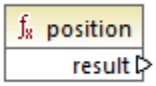
パラメーター

名前	型	説明
node	node()	入力ノード。

サンプル
xpath2 ライブラリの local-name 関数のために与えられたサンプルを参照してください。

6.6.24.6 position

現在処理されているノードセット内の現在のノードのポジションを返します。



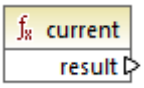
言語
XSLT 1.0。

6.6.25 xslt | xslt functions (xslt 関数)

このグループ内の関数はその他のXSLT 1.0 関数です。

6.6.25.1 current

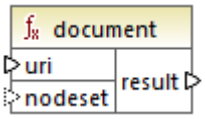
`current` 関数は引数を取らず、現在のノードを返します。



言語
XSLT 1.0。

6.6.25.2 document

外部 XML ドキュメントからノードにアクセスします。関数の結果は、出力ドキュメント内にあるノードへ返されます。



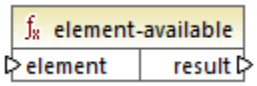
言語
XSLT 1.0。

パラメーター

名前	型	説明
uri	`xs:string`	必須。XML ドキュメントへのパスを指定します。XML ドキュメントは有効で解析可能である必要があります。
nodeset	`node()`	任意。ノード、および相対的である場合最初の引数として与えられているURI を解決するため使用されたベースURI を指定します。

6.6.25.3 element-available

element-available 関数は関数の文字列引数として入力されている要素がXSLT プロセッサーによりサポートされているかをテストします。引数文字列はQName として評価されます。これらはマッピングのために生成される基になるXSLT 内のこれらの名前空間のために宣言されているプレフィックスのため、XSLT 要素は`xsl:` プレフィックスを持つ必要があり、XML スキーマ要素は`xs:` プレフィックスを持つ必要があります。関数はブール値を返します。



言語

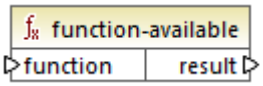
XSLT 1.0。

パラメーター

名前	型	説明
element	**xs:string**	要素名。

6.6.25.4 function-available

function-available 関数は**element-available** 関数に類似しており、関数の引数として提供されている関数名がXSLT プロセッサーによりサポートされているかテストします。入力文字列はQName として評価されます。関数はブール値を返します。



言語

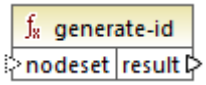
XSLT 1.0。

パラメーター

名前	型	説明
function	**xs:string**	関数名。

6.6.25.5 generate-id

generate-id 関数は任意の入力引数により識別されるノードセット内の最初のノードを識別する一意の文字列を生成します。入力パラメーターが与えられない場合、コンテキストノード上のID が生成されます。出力ドキュメントにある任意のノードに対して出力結果を接続することができます。



言語

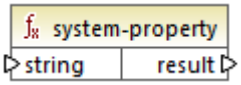
XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
nodeset	`node()`	入力ノードを提供するオプションの引数。

6.6.25.6 system-property

`system-property` 関数はXSLT プロセッサー(システム)のプロパティを返します。入力された文字列はQName として評価されるため、XSLT スタイルシート内でXSLT 名前空間に関連付けられているxsl:prefix を使用する必要があります。これらは`xsl:version`、`xsl:vendor`、と`xsl:vendor-url` です。基になるXSLT スタイルシート内のXSLT 名前空間とプレフィックスが関連付けられるためQname として入力文字列は評価され`xsl:` プレフィックスを持つ必要があります。



言語

XSLT 1.0、XSLT 2.0、XSLT 3.0。

パラメーター

名前	型	説明
string	`xs:string`	以下のいずれかであるプロパティ名を指定します: `xsl:version`、`xsl:vendor`、`xsl:vendor-url`。

6.6.25.7 unparsed-entity-uri

関数への入力文字列はDTD 内部にて宣言された(パースされていない)エンティティーにマッチする必要があります。パースされていないエンティティ(例えばイメージ)はパースされていないエンティティをロケートするURI を持ちます。関数の入力文字列はDTD 内で宣言されているパースされていないエンティティの名前と一致する必要があります。関数は出力ドキュメント内のノードに移動されるパースされていないエンティティのURI を返します。例えば、**href** ノードなど。

言語

XSLT 1.0。

パラメーター

名前	型	説明
string	`xs:string`	URI が抽出される解析されていないエンティティの名前。

7 マッピングの自動化と MapForce

MapForce を使用してデザインされたマッピングは（Linux とmacOS サーバーを含む）サーバー環境内でサーバーレベルのパフォーマンスを使用して次のAltova 変換エンジンにて実行することができます（これらのエンジンは個別にライセンスが供与される必要があります）:

- *RaptorXML Server*: マッピングの変換言語がXSLT 1.0、XSLT 2.0、XSLT 3.0 またはXQuery の場合、マッピングをこのエンジンを使用して実行することが最適です。RaptorXML Server を使用して自動化するを参照してください。
- *MapForce Server（またはMapForce Server Advanced Edition）*。変換言語がBUILT-IN* の場合このエンジンの使用が最適です。BUILT-IN 言語はMapForce 内のマッピング機能の多数をサポートしますが、MapForce Server（そして特に、MapForce Server Advanced Editionin ）はマッピングを実行するために最高のパフォーマンスを提供します。

＊BUILT-IN 変換言語は、MapForce Professional またはEnterprise Edition を必要とします。

更に、MapForce には、XSLT コードをコマンドラインインターフェイスから自動的に生成する機能が搭載されています。詳細に関しては、MapForce コマンドラインインターフェイスを参照してください。

7.1 RaptorXML Server を使用して自動化する

RaptorXML Server (今後は、略して RaptorXML) は、Altova 第3世代の超高速な XML と XBRL のためのプロセッサーです。最新の標準とパラレルコンピューティング環境のために最適化されています。クロスプラットフォームに対応可能で、エンジンは今日のマルチコアコンピューティングを有効に利用し、XML と XBRL データの高速処理を提供します。

RaptorXML には、Altova ダウンロードページ (https://www.altova.com/ja/download-trial-server.html) からダウンロードしてインストールすることのできる複数のエディションがあります:

- RaptorXML Server は、XML、XML スキーマ、XSLT、XPath、XQuery、などへのサポートを搭載した高速な XML 処理エンジンです。このエディションは、FlowForce Server インストールパッケージの一部です。
- RaptorXML+XBRL Server は、XBRL の一連の標準を処理および検証する追加機能を使用して、RaptorXML Server の全ての機能をサポートします。

制約事項:

- XML 署名はサポートされていません
- COM インターフェイスを介してグローバルリソースはサポートされていません
- ODBC と ADO データベース接続は Windows のみでサポートされています。他のオペレーティングシステムでは、JDBC を使用する必要があります。

XSLT でコード生成するには、MapForce は、生成の際に選択する出力フォルダーに保存される **DoTransform.bat** と呼ばれるバッチファイルを作成します。バッチファイルの実行は、RaptorXML Server を呼び出し、サーバー上で XSLT 変換を行います。

メモ　内蔵のエンジンを使用して、XSLT コードをプレビューすることができます。

7.2 MapForce コマンドラインインターフェイス

コマンドラインでのMapForce コマンドの一般的な構文:

```
MapForce.exe <filename> [/{target} [[<outputdir>] [/options]]]
```

レジェンド

コマンドライン 構文を示すために次の表記が使用されます:

表記	説明
かっこ無しのテキスト	アイテムは表示されるとおり入力されなければなりません。
<山かっこ内のテキスト>	値を提供する必要のあるプレースホルダです。
[角かっこ内のテキスト]	任意のアイテムです。
{かっこ内のテキスト}	必要とされるアイテムを設定します。1 つ選択します。
垂直線 (|)	相互的に排除するアイテムのためのセパレーターです。1 つ選択します。
省略記号 (…)	繰り返すことのできるアイテム。

<filename>

コードが生成されるマッピングデザイン (.mfd) ファイルです。

/{target}

コードが生成されるターゲット言語または環境を指定します。次のコード生成ターゲットがサポートされます。

ターゲット	説明
/XSLT	XSLT 1.0 コードを生成します。
/XSLT2	XSLT 2.0 コードを生成します。
/XSLT3	XSLT 3.0 コードを生成します。

<outputdir>

出力ディレクトリを指定する任意のパラメーターです。出力パスが与えられていない場合、現在の作業ディレクトリが使用されます。相対的なファイルパスは、現在の作業ディレクトリに対して相対的であることにご注意してください。

/options

/options は相互排他的ではありません。1 つ、または複数のオプションを設定することができます。

オプション	説明
/GLOBALRESOURCEFILE <filename>	マッピングが入力ファイル、出力ファイル、フォルダーパス、またはデータベースを解決するためにグローバルリソースを使用する場合、このオプションを適用することができます。詳細に関しては、Altova グローバルリソースを参照してください。 オプション /GLOBALRESOURCEFILE はグローバルリソース.xml ファイルへのパスを指定します。/GLOBALRESOURCEFILE が設定されている場合、/GLOBALRESOURCECONFIG が設定されなければなりません。
/GLOBALRESOURCECONFIG <config>	このオプションはグローバルリソース構成の名前を指定します（前のオプションも参照してください）。/GLOBALRESOURCEFILE が設定されている場合、/GLOBALRESOURCECONFIG が設定されなければなりません。
/LOG <logfilename>	指定されたパスでログファイルを生成します。<logfilename> はフルパスとして使用することができます。例えば、ディレクトリとファイル名の両方を含むことができます。しかしながら、フルパスが与えられると、ログファイルを生成するためにログファイルが存在する必要があります。ファイル名のみが指定されると、ファイルは Windows コマンドプロンプトの現在のディレクトリ内に保存されます。

リマーク

- 相対パスは MapForce を呼び出すアプリケーションの現在のディレクトリである作業ディレクトリに対して相対的です。これは、.mfd ファイル名、出力ディレクトリ、ログファイル名、およびグローバルリソースファイル名のパスに対して適用されます。
- コマンドラインではバックスラッシュと終わり引用符を最後に使用しないでください（例えば、"C:¥My directory¥"）。これら2つの文字はコマンドラインパーサーにより二重引用符マークとして解釈されます。引用符が必要な場合、("c:¥My Directory¥¥")、二重の円記号 ¥¥ を使用します。または、スペースの使用を回避してください。

サンプル

1) MapForce を開始して、マッピング <filename>.mfd を開くには、次を使用します:

```
MapForce.exe <filename>.mfd
```

2) XSLT 2.0 コードを生成し、<logfilename> という名前のログファイルを作成するには、次を使用します:

```
MapForce.exe <filename>.mfd /XSLT2 <outputdir> /LOG <logfilename>
```

3) グローバルリソースファイル <grfilename> からのグローバルリソース構成 <grconfigname> を考慮して XSLT 2.0 コードを生成するには、次を使用します:

```
Mapforce.exe <filename>.mfd /XSLT2 <outputdir> /GLOBALRESOURCEFILE
<grfilename> /GLOBALRESOURCECONFIG <grconfigname>
```

8 Altova グローバルリソース

Altova グローバルリソースはファイル、フォルダー、またはデータベースへのポータブルなレファレンスです。グローバルリソースとして保管されると、パスとデータベース接続の詳細は再利用できるようになりAltova アプリケーション全体で使用可能になります。例えば、複数のAltova デスクトップアプリケーション内で同じファイルを頻繁に開く必要がある場合、グローバルリソースとして定義すると便利な場合があります。このようにすると、「ファイルを開く」ダイアログボックスから対応するグローバルリソースを開くことができるため、ファイルパスを覚えている必要がありません。この方法はファイルパスが変更されると、パスの一部を変更するのみという利点があります。

グローバルリソースの一般的な使用法は、データベース接続を一度定義し、グローバルリソースをサポートする全てのAltova アプリケーションで再利用することです。例えば、MapForce マッピングがデザインされるマシンでデータベース接続を作成し、MapForce Server がマッピングを実行するマシン上で同じ接続を再利用することができます（これは、一部の場合、両方のマシンに同じデータベースクライアントソフトウェアがインストールされている場合があります）。

任意で、（「構成」として既知の）同じグローバルリソースの複数のバリエーションを作成することができます。これによりファイル、またはフォルダーパス（またはデータベース）を必要に応じて変更することができます。例えば、「開発」と「生産」の構成を持つデータベースリソースを作成することができます。

次のAltova デスクトップアプリケーションからグローバルリソースを作成することができます: Altova Authentic、DatabaseSpy、MobileTogether Designer、MapForce、StyleVision、およびXMLSpy。サーバー製品では、次のAltova サーバーアプリケーションでグローバルリソースを作成することができます: FlowForce Server、MapForce Server、RaptorXML Server、RaptorXML+XBRL Server。

グローバルリソース は、多種のシナリオのためにMapForce 内で使用することができます:In MapForce 内では、グローバルリソース（ファイル、フォルダー、または、データベース参照など）を多種のシナリオのために使用することができます。例:

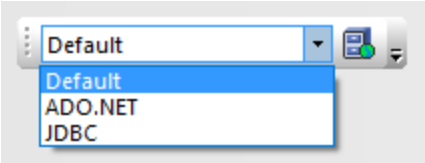
- マッピングの入力として構成することのできるファイルパスを宛てる場合。次を参照してください: 例: 変数入力ファイルを持つマッピングを実行する。
- マッピングの出力を構成することのできるパスにリダイレクトする詳細に関しては、次を参照してください: 例: 出力を変数フォルダに出力する。

- MapForce Basic Edition は、グローバルリソースとして定義されるデータベース接続をサポートしません。

8.1 グローバルリソースの作成

グローバルリソースはファイル、またはフォルダーパス、またはデータベース接続を表す再利用可能なレファレンスです。グローバルリソースは一度定義されると、複数のAltova アプリケーション全般でサポートするコンテキスト内で必要な回数再利用することができます。データベースを例に挙げると、1つ以上のAltova アプリケーションで特定のデータベースと共に頻繁に作業する場合、グローバルリソースとしてデータベース接続を追加することはよいアイデアです。このようにすると、他のAltova アプリケーションから同じデータベースに接続する必要がある都度データベース接続ウィザードのステップを実行する必要がありません。

各グローバルリソースはいわゆる「構成」を持つことができます。構成によりAltova アプリケーションにより使用、または作成されるファイル、フォルダー、およびデータベース間を簡単に切り替えることができます。これはテストのシナリオで特に役に立ちます。例えば、異なるドライバーの型を持つ同じデータベースに接続する以下の3つの個別の接続から構成されるデータベースリソースを作成することができます: (a) デフォルトの接続型であるODBC。(b) JDBC、と(c) ADO.NET。このようにして、特定のドライバーを持つデータベースに接続するには、グローバルリソースドロップダウンリストから対応する構成を選択するだけです。



グローバルリソースドロップダウンリスト

ボタンクリックのみで、構成は変数フォルダーにマッピング出力を生成することができます。例えば、フォルダーリソースを2つの構成から作成することができます: (a) ディレクトリ **C:\Testing** を指す「テスト」(b) ディレクトリ **C:\Production** を指す「生産」。グローバルリソースドロップダウンリストから必要とする構成をマッピングの実行の前に選択し **C:\Testing** または **C:\Production** フォルダーに出力を生成するマッピングを構成することができます。サンプル: 変数フォルダーに出力を生成する内で詳細について説明されています。

グローバルリソースを作成する方法:

1. 「ツール」メニューから「**グローバルリソース**」をクリックします。(または、「**グローバルリソースの管理**」ツールバーボタンをクリックします。)
2. 「**追加**」をクリックして、(ファイル、フォルダー、データベースなどの) 作成するリソースの型を選択します。
3. 「**リソースエイリアス**」テキストボックス内に詳細名を入力します (例えば、"InputFile"、"OutputFolder"、"DatabaseConnection").
4. 「Default」構成をセットアップします:
   a) ファイル、またはフォルダーの場合、デフォルトでポイントされるこのリソースへのファイル、またはフォルダーを参照します。
   b) データベース接続の場合、「**データベースの選択**」をクリックして、データベース接続ウィザードに従い、データベースに接続します。(グローバルリソースドロップダウンリストから他の構成が明示的に選択されている場合、または、サーバー実行内でコマンドラインパラメーターとして提供されていない場合以外) このデータベース接続はマッピングの実行の際にデフォルトで使用されます。
5. オプションとして、リソースに追加構成が存在する場合 (例えば、データベースの場合のドライバーの種類、または、ファイルとフォルダーの場合代替パスなどの) 「**構成の追加**」ボタンをクリックして、(例えば、"ProductionFolder" or "JDBC_Alternative" など) 詳細名を入力し、次にようにセットアップします:
   a) ファイル、またはフォルダーの場合、前のステップ内で定義済みのデフォルトの構成の代替としてこのリソースによりポイントされるファイル、またはフォルダーを参照します。
   b) データベース接続の場合、データベースに接続するためにデータベース接続ウィザードに従います。このデータベース接続はデフォルトの代替として使用されます。
   一部の場合、デフォルトの構成のコピーとして構成を作成し、編集するほうが便利な場合があります。この場合、「**現在選択されている構成のコピーとして構成を追加する**」ボタンをクリックします。
6. 必要とされる各追加構成のために前のステップを繰り返します。

8.2 グローバルリソース XML ファイル

デフォルトでは、すべてのグローバルリソースは Altova アプリケーションの作成場所に関わらず、以下のパスに保存されます: **C:\Users\<username>\Documents\Altova\GlobalResources.xml**。これにより簡単にバックアップすることができ、Altova 製品がインストールされている他のワークステーションに対してポータブルになります。**GlobalResources.xml** ファイルの名前を変更、または、複製することができ、複数のグローバルリソースファイルを作成することができます。しかしながら、Altova アプリケーション内では 1 つのグローバルリソースファイルのみしかアクティブ化することはできません。

> 他の Altova アプリケーションとは異なり、FlowForce Server はグローバルリソースファイルを必要としません。代わりにリソースは他の FlowForce 構成データと同様に管理されます（アクセスパーミッションと共に再利用することが可能です）。

アクティブなグローバルリソース ファイルをセットアップする方法:

1. 「**ツール**」メニューから、「**グローバルリソース**」をクリックします。（または、**グローバルリソース**  ツールバーボタンをクリックします。）
2. 「**参照**」をクリックして、必要とされるグローバルリソース XML ファイルを選択します。

複数のグローバルリソースファイルを使用している場合、現在アクティブなファイルに現在のコンテキストにより必要とされているすべてのグローバルリソースファイルが含まれていることを確認してください。例えば、グローバルリソースを使用するパスからデータを読み取るようにマッピングが構成されている場合、現在アクティブなグローバルリソースファイルにはその特定のグローバルリソースが含まれている必要があります。それ以外の場合、「グローバルリソースを解決する際にエラーが発生しました」などのエラーメッセージが「**メッセージ**」ウィンドウに表示されます。

8.3 例: 変数入力ファイルを持つマッピングを実行する

入力として XML ファイルを取るマッピングを頻繁に実行すると仮定します。入力 XML を変更する必要がある場合、ソース XML コンポーネントのプロパティを編集し、新規の入力ファイルを参照することができます。マッピングの入力 XML を変更する場合、ソース XML コンポーネントのプロパティを開き、新規の入力ファイルを参照します。次を参照してください: コンポーネント設定の変更。これは 1 度きりのタスクの場合、簡単に達成することができます。しかしながら、マッピングの入力 XML ファイルを一日に、または毎時に、複数回実行する場合、例えば、XML ファイルをマッピングの入力として使用し、レポートを生成するためにマッピングを実行する場合、他の XML ファイルから同じレポートを毎晩作成する場合などが挙げられます。

このような場合、グローバルリソースを使用することができます: マッピングを複数回編集する代わりに (または、複数のコピーを保管する代わりに) グローバルリソース (または「ファイルエイリアス」) として定義されているファイルから読み取りマッピングを構成することができます。このサンプルで挙げられる必要条件を満たすには、ファイルエイリアスが 2 つの構成を持つように構成することができます:

1. 「デフォルトの」 – この構成は、「morning」 XML ファイルをマッピングの入力として提供します。
2. 「EveningReports」 – この構成は、「evening」 XML ファイルをマッピングの入力として提供します。

構成を正しく設定することにより、これらの入力ファイルを使用してマッピングを実行することができます。ファイルエイリアスセットアップされると、下で示されるとおり、マッピングを実行する前に、希望する構成をドロップダウンリストから選択することができます。

ステップ 1: グローバルリソースを作成する

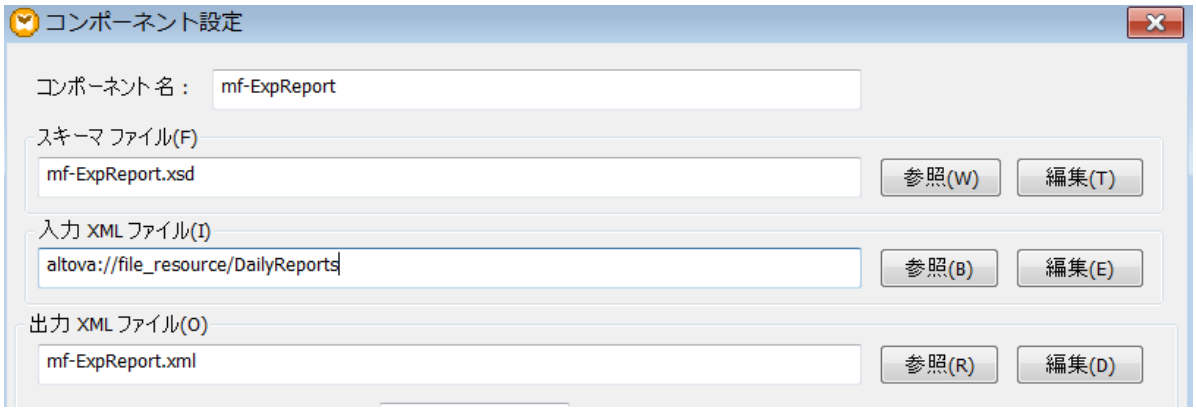
ファイルエイリアスを以下のとおり作成することができます:

1. 「ツール」 メニューから「**グローバルリソース**」 をクリックします。(または、**グローバルリソース** ツールバーボタンをクリックします)。
2. 「**追加 | ファイル**」をクリックします。
3. 「**リソースエイリアス**」 テキストボックスに名前を入力します (このサンプルでは、「DailyReports」 が適切な名前です)。
4. 「**参照**」 をクリックして、次のファイルを選択します: <**マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\mf-ExpReport.xml**.
5. 「**ファイルの追加**」 をクリックして、「EveningReports」という名前を与えます。
6. 「**参照**」 をクリックして、今度は、次のファイルを選択します: <**マイドキュメント>\Altova\MapForce2021\MapForceExamples\Tutorial\mf-ExpReport2.xml**.

ステップ 2: マッピング内のグローバルリソースを使用する

必要とされる グローバルリソースが作成されましたが、マッピングを使用することはまだできません。前に定義されたファイルエイリアス (グローバルリソース) から読み込みマッピングを変更するには、以下を行います:

1. 次のマッピングを開きます <**マイドキュメント**>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥**Tut-ExpReport.mfd**.
2. マッピング上のソースコンポーネントのヘッダーを右クリックして、コンテキストメニューから「**プロパティ**」 を選択します。
3. **入力** XML ファイルの横の「**参照**」をクリックします。
4. 「**グローバルリソースに切り替える**」 をクリックして、前に定義されたファイルエイリアス「DailyReports」 を選択します。
5. 「**開く**」をクリックします。入力 XML ファイルパスは、グローバルリソースを使用するパスを示す **altova://file_resource/DailyReports** になりました。



ステップ 3: 希望する構成を使用してマッピングを実行する

マッピングを実行するまえに、入力 XML ファイルを以下のように簡単に切り替えることができます:

- ファイル **mf-ExpReport.xml** を入力として使用するには、「**ツール**」メニューから「**アクティブな構成 | デフォルト**」をクリックします。
- ファイル **mf-ExpReport2.xml** を入力として使用するには、「**ツール**」メニューから「**アクティブな構成 | EveningReports**」をクリックします。,

または、**グローバルリソース**ドロップダウンリストから必要とされる構成を選択します。

マッピングの結果を構成を使用してプレビューするには、「**出力**」タブをクリックして、生成された出力の差分を確認してください。

8.4 例: 出力を変数フォルダに出力する

このサンプルはグローバルリソースを使用して、異なるフォルダーにマッピングの出力をリダイレクトする方法について説明しています。

マッピング出力をディレクトリ（例えば、**C:\Testing**）に生成する場合、または出力が他のディレクトリ（例えば、**C:\Production**）に生成する場合があるとします。グローバルリソースを使用すると、2つの構成を持つフォルダーエイリアスを作成することが可能になります:

1. 「デフォルトの」構成 – 出力を以下の場所に生成します: **C:\Testing**
2. 「Production」構成 – 出力を以下の場所に生成します: **C:\Production**.

この目的を達成するためにステップは以下のとおりです。

ステップ 1: グローバルリソースを作成する

フォルダーエイリアスを以下のとおり作成することができます:

1. 「**ツール**」メニューから「**グローバルリソース**」をクリックします。（または、**グローバルリソース** ツールバーボタンをクリックします）。
2. 「**追加 | フォルダー**」をクリックします。
3. 「**リソースエイリアス**」テキストボックスに名前を入力します（このサンプルでは、「OutputDirectory」は適切な名前です）。
4. 「**参照**」をクリックして、次のフォルダーを選択します: **C:\Testing**。（このフォルダが使用中のオペレーティングシステムで既存であることを確認してください）。
5. 「**ファイルの追加**」 をクリックして、新しい構成のために名前を入力します（このサンプルでは、「ProductionDirectory」）。
6. 「**参照**」をクリックして、次のフォルダーを選択します次のフォルダー: **C:\Production**（このフォルダが使用中のオペレーティングシステムで既存であることを確認してください）。

ステップ 2: マッピング内のグローバルリソースを使用する

必要とされるグローバルリソースが作成されました。しかしながら、マッピングをまだ使用することはできません。前に定義されたフォルダーエイリアス（グローバルリソース）から使用するようにマッピングを変更するには、以下を行います:

1. 次のマッピングを開きます <**マイドキュメント**>¥Altova¥MapForce2021¥MapForceExamples¥Tutorial¥**Tut-ExpReport.mfd**.
2. マッピング上のターゲットコンポーネントを右クリックし、コンテキストメニューから「**プロパティ**」を選択します。
3. **出力** XML ファイルの横の「**参照**」をクリックします。
4. 「**グローバルリソースに切り替える**」をクリックして、「**保存**」をクリックします。
5. 出力 XML ファイルを保存するようにプロンプトされると、**output.xml**（または他の出力ファイルに与える詳細なファイル名）を入力します。出力 XML ファイルパスは、グローバルリソースとして定義されるパスを示す **altova://folder_resource/OutputDirectory/output.xml** になります。

ステップ 3: 希望する構成を使用してマッピングを実行する

マッピングを実行する前に、希望するマッピング出力フォルダーを簡単に切り替えることが以下のように行うことができます:

- 「**ツール**」メニューから「**アクティブな構成 | デフォルト**」をクリックします。マッピングの結果をプレビューするために「**出力**」タブをクリックします。マッピング出力（下で説明されているとおり、一時、または永続ファイル）が **C:\Testing** ディレクトリに出力されます。
- 「**ツール**」メニューから**アクティブな構成 | ProductionDirectory** をクリックして、「**出力**」タブをクリックします。マッピング出力（下で説明されているとおり、一時、または永続ファイル）は、**C:\Production** ディレクトリ内に生成されます。

メモ MapForce を構成して、出力が永続ファイルに書き込まれるように設定されている以外は、マッピング出力は、デフォルトで、一時ファイルとして書き込まれます。

MapForce に一時ファイルの代わりに永続ファイルを生成するように構成するには、以下を行います:

1. 「**ツール**」メニューから「**オプション**」をクリックします。
2. 「**全般**」セクションで、オプション「**最終出力ファイルに直接書き込む**」を選択します。

9 MapForce のカスタマイズ

9.1 MapForce オプションの変更

MapForce 内の一般および他の優先順位を変更する方法は、以下の通りです:

- 「ツール」メニューから「オプション」をクリックします。

使用することのできるオプションは、以下に表示されているとおりです。

全般

このページで使用することのできるページ内の設定は、以下のとおりです:

**ロゴの表示	起動時に表示**	MapForce が起動されると、イメージ（スプラッシュスクリーン）を表示、または非表示にします。
背景をグラデーションで表示	マッピングペイン内の背景をグラデーションで表示する機能を有効化、または無効化します。
表示する注釈を N 列に制限する	このオプションは注釈をサポートするコンポーネントに対して適用されます（例えば、XML スキーマ、EDI）。注釈テキストに複数の行が含まれる場合、このオプションを有効化すると、コンポーネント上の最初の N 行のみが表示されます。N は指定することのできる値です。この設定は、コンポーネント内で表示される SELECT ステートメントにも適用されます。
名前のエンコード	新規コンポーネントのために デフォルトの文字の円コードを設定します。この設定は、各コンポーネントのために個別に変更することができます。次を参照してください: コンポーネント設定の変更.
実行タイムアウトの使用	「出力」ペイン内でマッピングの結果をプレビューする際に実行のタイムアウトを設定します。
一時的なファイルの出力を生成する	このオプションが設定されていると、マッピングの結果をプレビューする際に生成される出力は、一時的なファイルに書き込まれます（これはデフォルトのオプションです）. 出力ファイルパスにまだ存在しないフォルダーが含まれる場合、MapForce がこれらのファイルを作成します。
最後の出力ファイルに直接書き込む	このオプションが設定されていると、マッピングの結果をプレビューする際に生成される出力は、実際のファイルに書き込まれます 出力ファイルパスにまだ存在しないフォルダーが含まれる場合、マッピングエラーが発生します。 警告: このオプションは更なる確認をせず、既存の出力ファイルを上書きします。
N 文字のステップでテキストを表示する	大きな XML とテキストファイルを生成するマッピングをプレビューする際に、出力 ペイン内に表示されるテキストの最大のサイズを指定します。出力テキストがこの値を超えると、「更にロードする」ボタンをクリックして次のチャンクをロードする必要があります。詳細に関しては、次を参照してください: 出力のプレビュー。

編集

このページで使用することのできる設定は、以下のとおりです:

マウスのドラッグ時のコンポーネントの入れる	マウスでドラッグする際にコンポーネントまたは関数が他のコンポーネントと共に整列するかを指定します。コンポーネントの整列 を参照。
スマートコンポーネントの削除	有効化されると、**スマートコンポーネントの削除** オプションは、削除された接続を "記憶" します。コンポーネントの削除後、接続を保持する を参照。

メッセージ

「このメッセージを表示しない」オプションにより以前に無効化されたメッセージの通知を再有効化します。

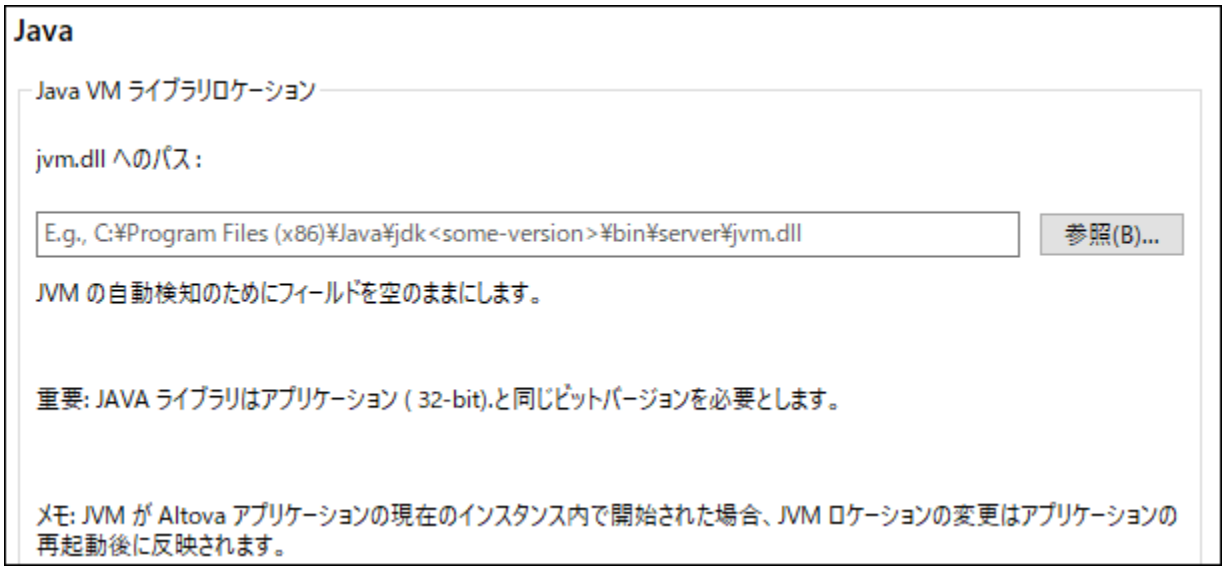
ネットワークプロキシ

ネットワークプロキシの設定を参照してください

9.1.1 Java 設定

Java タブ上では、ファイルシステム上で Java VM (仮想マシン) へのパスを任意で入力することができます。カスタム Java VM パスの追加は常に必要なわけではありません。デフォルトで、MapForce は、W indows レジストリと JAVA_HOME 環境変数を(この順序で)読み取ることにより、Java VM パスを自動的に影響しようとします。このダイアログボックスに追加されるカスタムパスは、自動的に検知される VM パスより先優先順位を与えられます。

カスタム Java VM パスを追加する必要がある場合、例えば、インストーラを持たず、(例えば、Oracle の OpenJDK などの) レジストリエントリを作成する必要のない Java 仮想マシンを使用している場合、カスタム Java VM パスを追加する必要がある場合があります。MapForce により自動的に検知された Java VM パスを何らかの理由でオーバーライドする場合、このパスを設定する必要があります。



以下の点に注意してください:

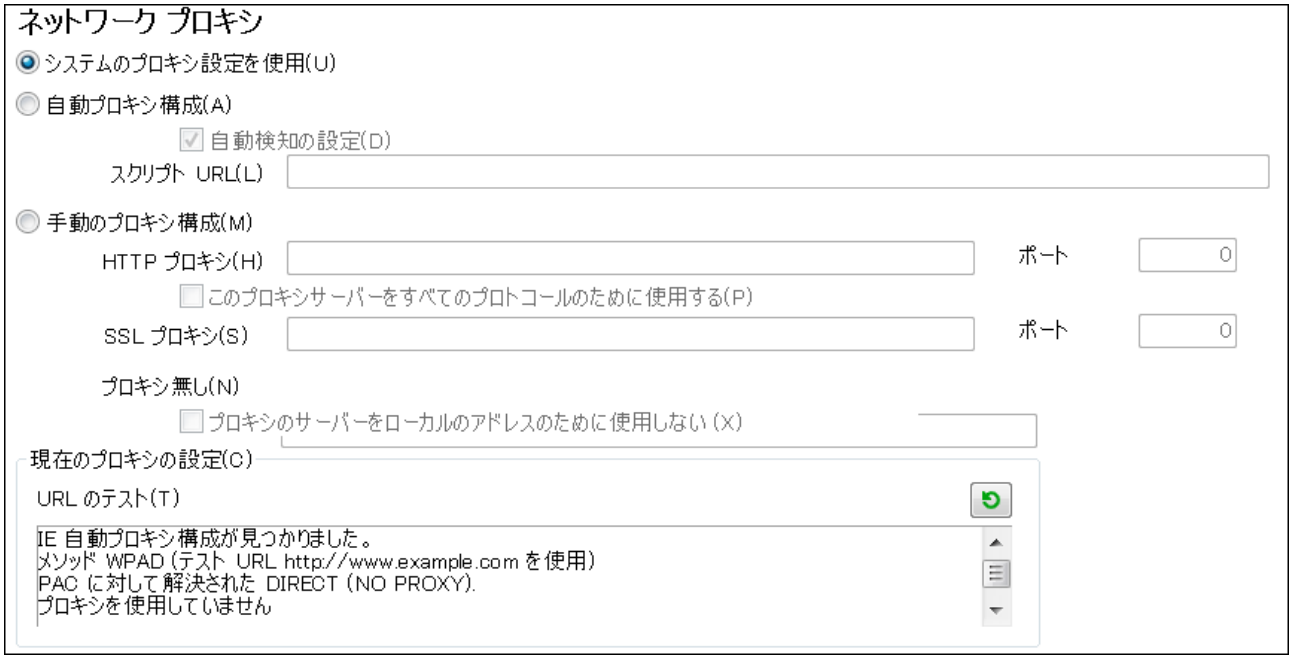
- Java VM パスは Altova デスクトップと(サーバーではない) アプリケーションにより共有されています。この結果、アプリケーション内で変更されると、他の Altova アプリケーションに自動的に適用されます。
- パスは JDK がインストールされているディレクトリに相対する **\bin\server** または **\bin\client** ディレクトリからの **jvm.dll** ファイルを指す必要があります。
- MapForce プラットフォーム(32-ビット、64-ビット) が JDK と同じである必要があります。

- Java VM パスの変更後、新規の設定を反映するためにMapForce を再起動する必要があります。

9.1.2 ネットワークプロキシ設定

ネットワーク プロキシ セクションでは、カスタムのプロキシの設定を構成することができます。(XML 検証の目的のために) MapForce のインターネットへの接続方法に影響します。デフォルトでは、 MapForce はシステムのプロキシの設定を使用します、ですから、多くの場合プロキシの設定を変更する必要はありません。必要な場合、代替のネットワークプロキシを下のオプションを使用して設定することができます。

メモ ネットワークプロキシ設定は、Altova MissionKit アプリケーション間で共有されています。結果、1 つのアプリケーション内で設定が変更されると、自動的に他の全てのアプリケーションに影響を与えます。



システムプロキシの設定の使用

システムプロキシ設定を介して構成可能な インターネット エクスプローラー(IE) 設定を使用します。`netsh.exe winhttp` を介して構成される設定が必要とされます。

自動プロキシの構成

以下のオプションを使用することができます:

- *自動検知の設定:* DHCP または DNS を使用して W PAD スクリプト (`http://wpad.LOCALDOMAIN/wpad.dat`) を検索し、プロキシセットアップのためにこのスクリプトを使用します。
- *スクリプト URL:* プロキシセットアップのために使用されるプロキシ自動構成 (`.pac`) スクリプトに対する HTTP URL を指定します。
- *再ロード:* 現在の自動プロキシ構成をリセットして再ロードします。このアクションには W indows 8 または以降が必要とされ、30 秒程の時間が必要です。

手動のプロキシの構成

ホスト名とポートを対応する製品のプロキシのために手動で指定します。サポートされるスキームはホスト名に含まれている場合があります（例：`http://hostname`）。プロキシがスキームをサポートする場合、対応するプロトコールと同じである必要はありません。

以下のオプションを使用することができます:

- *このプロキシサーバーをすべてのプロトコールのために使用する:* 全てのプロトコルのためにHTTP プロキシのホスト名とポート使用します。
- *プロキシ無し:* セミコロン（`;`）により区別されているプロキシを使用しないホスト名、ドメイン名、または、ホストのためのIP アドレスのリスト。IP アドレスは切り捨てられず、IPv6 アドレスは角かっこで囲まれる必要があります（例: `[2606:2800:220:1:248:1893:25c8:1946]`）。ドメイン名は、ドットと共に開始される必要があります（例：`.example.com`）.
- *プロキシのサーバーをローカルのアドレスのために使用しない*：チェックされている場合、*プロキシ無し*リストのために`<local>`を追加します。このオプションが選択されている場合、次の場合、プロキシは使用されません（i）`127.0.0.1`、(ii) [::1]、(iii)（`.`）ドット文字を含んでいないホスト名すべて。

メモ　プロキシサーバーが設定されており、マッピングをAltova FlowForce Server にデプロイする場合、*ローカルアドレスのためにプロキシサーバーを使用しない*を選択する必要があります。

現在のプロキシの設定

プロキシの検知の詳細なログを提供します。*URL のテスト* フィールドの右の「**更新**」ボタンを使用して更新することができます（例、URL のテスト を変更する場合、または、プロキシの設定が変更された場合）。

- *URL のテスト:* A test URL のテストを使用して、どのプロキシが特定のURL ために使用されているかを確認することができます。URL を使用して、I/O はされません。プロキシの自動構成が選択されている場合、このフィールドは空にしておく必要があります（*システムプロキシの設定の使用*、または、*自動プロキシの構成* を使用して）。

9.2 キーボードのショートカット

デフォルトで MapForce は以下のキーボードのショートカットを提供しています:

F1	ヘルプメニュー
F2	次のブックマークへ(出力ウィンドウにて)
F3	次を検索
F10	メニューバーをアクティベート
Num +	現在のアイテムノードを展開
Num -	アイテムノードを縮退
Num *	現在のアイテムノード以下を全て展開
CTRL + TAB	開かれているマッピング間の切り替え
CTRL + F6	開かれているウィンドウをサイクル
CTRL + F4	アクティブなマッピングドキュメントを閉じる
Alt + F4	MapForce を閉じる
Alt + F, F, 1	最後に閉じられたファイルを開く
Alt + F, T, 1	最後に閉じられたプロジェクトを開く
CTRL + N	新規ファイル
CTRL + O	ファイルを開く
CTRL + S	ファイルの保存
CTRL + P	ファイルの印刷
CTRL + A	すべて選択
CTRL + X	切り取り
CTRL + C	コピー
CTRL + V	貼り付け
CTRL + Z	元に戻す
CTRL + Y	やり直し
Del	コンポーネントを削除(プロンプト付き)
Shift + Del	コンポーネントを削除(プロンプトなし)
CTRL + F	検索
F3	次を検索
Shift + F3	前を検索
矢印キー	
(上 / 下)	コンポーネント内にあるアイテムを選択
Esc	編集の破棄 / ダイアログボックスを閉じる
Return	選択の確定
出力ウィンドウホットキー	
CTRL + F2	ブックマークの挿入/削除
F2	次のブックマークへ
SHIFT + F2	前のブックマークへ
CTRL + SHIFT + F2	全てのブックマークを削除
ズーム ホットキーhotkeys	
CTRL + マウスホイール前	ズームイン

CTRL + マウスホイール後	ズームアウト
CTRL + 0 (ゼロ)	ズームのリセット

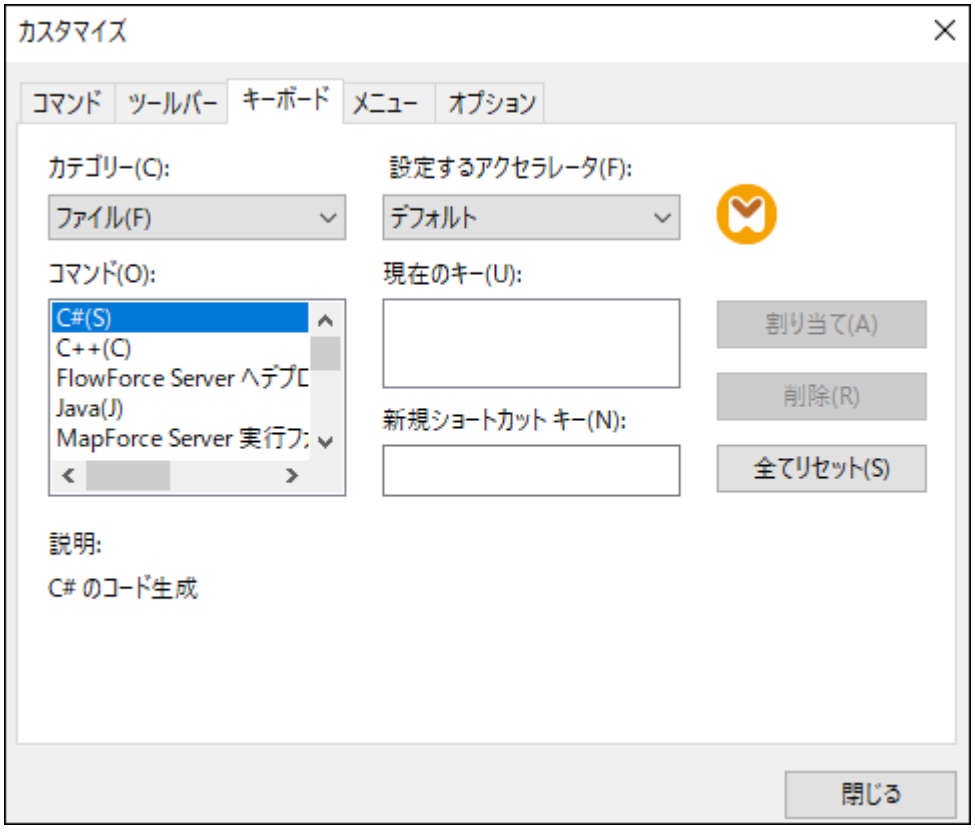
9.2.1 ショートカットのカスタム化

MapForce 内のキーボードのショートカットを以下のように定義または変更することができます:

1. 「**ツール**」メニューから「**カスタマイズ**」をクリックします。
2. 「**キーボード**」タブをクリックします。

コマンドに新規ショートカットを割り当てる

1. 「**ツール | カスタマイ**」コマンドを選択して、キーボードタブをクリックします。
2. 「**カテゴリー**」コンボボックスをクリックして、メニュー名を選択します。
3. コマンドリストボックス内で新しいショートカットを割り当てる**コマンド**を選択します。
4. 「**新規ショートカットキー**」をクリックします: テキストボックスから、コマンドを有効化するショートカットキーを押します。



ショートカットはテキストボックス内にすぐに表示されます。ショートカットが以前に割り当てられている場合、その関数がテキストボックスの下に表示されます。

5. 「**割り当**」てボタンをクリックして、ショートカットを割り当てます。
現在のキーリストボックスにショートカットが表示されます。
(新規ショートカットキーテキストボックス内のエントリをクリアするには、コントロールキーを押します **CTRL**、**ALT** または **SHIFT**)。

ショートカットを元に戻す、または削除する

1. 現在のキーリストボックス内から削除するショートカットをクリックします。
2. 「削除」ボタンをクリックします。
3. 「閉じる」ボタンクリックして、確認します。

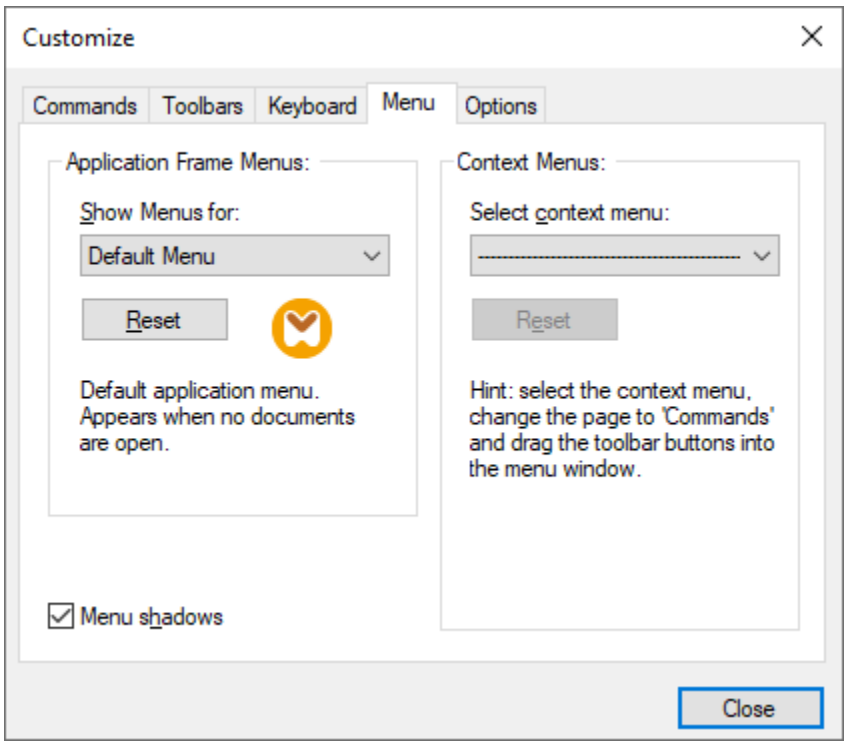
メモ　設定するアクセラレーターには現在関数が存在しません

9.3 メニューをカスタム化する方法

標準 MapForce メニューおよび例えば、コマンドを追加、変更、および削除するために、コンテキストメニューをカスタム化することができます。メニューのカスタム化をデフォルトの状態にリセットすることができます。

メニューをカスタム化する方法

- 「ツール」メニューから、「オプション」をクリックして、「メニュー」をクリックします。



メニューをカスタム化する方法

[デフォルトメニュー] バーはメインウィンドウ内でドキュメントが開かれていない場合に表示されます。[MapForce デザイン] メニューバーは1つまたは複数のマッピングが開かれていると表示されるメニューバーです。各メニューバーを個別にカスタム化することができ、カスタム化の変更はお互いに影響を与えません。

メニューバーをカスタム化するには「メニューの表示」ドロップダウンリストから選択します。[コマンド] タブをクリックして [コマンド] リストからコマンドをメニューバーまたはメニューにドラッグします。

メニューからコマンドを削除しリセットする方法

メニュー内のメニューまたはコマンドを全て削除するには以下を行います:

1. **メニューの表示** ドロップダウンリストから以下の1つを選択します:
   - **デフォルトのメニュー**(これはドキュメントが開かれていない場合に使用可能なメニューを表示します)
   - **MapForce デザイン**(これはドキュメントが開かれていない場合に使用可能なメニューを表示します)

2. カスタム化ダイアログが開かれている状態で(i) アプリケーションのメニューバーから削除するメニュー、または(ii) これらのメニューのいずれかから削除するコマンドを選択します。
3. (i) メニューからメニューバーまたはメニューコマンドからドラッグし(ii) メニューまたはメニューコマンドを右クリックし[削除]を選択します。

[メニューの表示]ドロップダウンリストから選択し[リセット]ボタンをクリックしてメニューを元の状態にリセットすることができます。

アプリケーションのコンテキストメニューをカスタマイズする方法

コンテキストメニューはアプリケーションのインターフェイス内の特定のオブジェクトを右クリックすると表示されるメニューです。これらのコンテキストメニューのそれぞれは以下を行ってカスタム化することができます:

1. [コンテキストメニューの選択]ドロップダウンリストからコンテキストメニューを選択します。コンテキストメニューがポップアップされます。
2. [コマンド]タブをクリックします。
3. [コマンド]リストボックスからコマンドをコンテキスメニューにドラッグします。
4. コンテキストメニューからコマンドを削除するために、コンテキストメニュー内でそのコマンドを右クリックして[削除]を選択します。代わりに、コマンドをコンテキストメニューからドラッグします。

[コンテキストメニューの選択]ドロップダウンリスト 内で選択し[リセット]ボタンをクリックしてコンテキストメニューを初期のインストールの状態にします。

メニューとメニューの影

全てのメニューに影を与えるために[メニューの影]チェックボックスを選択します。

9.4 カタログファイル

MapForce はOASIS XML カタログメカニズムのサブセットをサポートします（https://www.oasis-open.org/committees/entity/spec-2001-08-06.html）。カタログメカニズムにより MapForce が共通に使用されるDTD と XML スキーマ（およびスタイルシートと他のファイル）をパブリックURI から解決する代わりにローカルフォルダからを抽出することができるようになります。これは全体的な処理スピードを増やしオフライン（すなわちネットワークに接続されていない状態）での作業を可能にし、ドキュメントのポータビリティを改善します。

カタログのしくみ

カタログは一般的にパブリックDTD または通常は、ローカルファイルパスローカルURI へのスキーマレファレンスをリダイレクトするために使用されます。これを達成するために、XML フォーマットのカタログファイルはパブリックスキーマURI とローカルURI の間のマッピングを定義します。MapForce がXML ドキュメントを解析する都度、カタログファイル内部のスキーマURI（または、適用可能の場合、DTD のパブリックまたはシステム識別子）が最初に検索されます。マッピングがカタログファイル内で見つかった場合、そのレファレンスが使用されスキーマはローカルファイルから読み取られます。カタログファイル内でマッピングが見つからない場合 XML ドキュメントのURI はそのまま解決されます。

例えば、以下のXML ファイルはMapForce により処理される必要があると仮定します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Articles xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Articles.xsd">
   <Article>
      <Number>1</Number>
      <Name>T-Shirt</Name>
      <SinglePrice>25</SinglePrice>
   </Article>
</Articles>
```

(MapForce が既知の通り) **catalog.xml** ファイルがローカルディレクトリに存在し、以下のラインが含まれると想定します。この点については以下で詳しく説明されています:

```
<catalog>
  <!--...-->
  <uri name="http://www.w3.org/2001/XMLSchema-instance.xsd" uri="files/XMLSchema-
instance.xsd"/>
  <!--...-->
</catalog>
```

XML ファイルを解析する場合、MapForce はカタログファイル内でスキーマレファレンス `http://www.w3.org/2001/XMLSchema-instance.xsd` のための一致を検知します。この結果スキーマはカタログファイルに対して相対的なローカルパス `files/XMLSchema-instance.xsd` からロードされますカタログファイルファイル内でマッピングが見つからない場合、スキーマは`http://www.w3.org/2001/XMLSchema-instance` からロードされます。

ルートカタログ

MapForce が開始されると、**RootCatalog.xml** と呼ばれるファイルを"Program Files" ディレクトリからロードします。**RootCatalog.xml** はそれぞれが`nextCatalog` 要素内にあるカタログファイルのリストを含んでいます。これらのカタログファイルはルックアップされ内部にあるURI は指定されているマッピングに従いMapForce により解決されます。

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

RootCatalog.xml

上記のリスティングで、以下のカタログがルックアップのためにリストされていることに注意してください：

- **CustomCatalog.xml** は自身のマッピングを作成することができるファイルです。このファイルは以下のディレクトリ内にあります: **C:\users\<name>\Documents\Altova\MapForce2021**。Altova 構成済みのカタログファイルで既にカスタムスキーマのために対処されていない場合（次のポイントを参照してください）**CustomCatalog.xml** にマッピングを追加することができます。
- **%AltovaCommonFolder%/Schemas** ディレクトリからの複数の**catalog.xml** ファイル。各 **catalog.xml** ファイルは（SVG、DITA、DocBook、W SDL などの）特定のスキーマのディレクトリの内部にあり、それぞれは対応するスキーマのローカルで保存されたコピーを指す URI へのパブリックおよびまたはシステム識別子をマップします。
- **CoreCatalog.xml** にはスキーマをロケートするための Altova 固有のマッピングが含まれています。このファイルは MapForce [Program Files] 内にあります。

以下の点にご注意してください：

- **CustomCatalog.xml** を変更する場合 サポートされる要素を使用してください。エラーを引き起こす可能性があるため既存のマッピングを複製しないでください。
- **%AltovaCommonFolder%\Schemas\schema** フォルダー内の **catalog.xml** ファイルは古い XML スキーマ仕様を実装する DTD へのレファレンスを含んできます。XML スキーマドキュメントをこれらのスキーマに対して検証してはなりません。古い XML スキーマ仕様はこのようなドキュメントと作業するための対応するスキーマ URI を効果的に解決する機能 をMapForce に提供するためにのみ含まれていました。

上記の一部のディレクトリパスは環境関数の助けを得て表示されています。レファレンステーブルに関しては 環境関数を参照してください。

サポートされる要素

CustomCatalog.xml 内でエントリを作成するには、以下にリストされる要素のみを使用して下さい。OASIS XML カタログ仕様の他の要素はサポートされていません。

要素	属性	例
public	• `systemId` はリソースのパブリック識別子を指定します。 • `uri` URI レファレンスを指定します（例えば、ローカルディレクトリに対する相対的なパス）	`<public publicId="-//W3C//DTD XMLSCHEMA 200102//EN" uri="files/XMLSchema.dtd"/>`
system	• `systemId` はリソースのシステム識別子を	`<system`

要素	属性	例
	指定します。 • uri URI レファレンスを指定します（例えば、ローカルディレクトリに対する相対的なパス）	systemId="http://www.w3.org/2009/XMLSchema/datatypes.dtd" uri="files/datatypes.dtd"/>
uri	• name はURI レファレンスを指定します • Uri は（例えば、ローカルファイルに対する相対的なパスなど）代替 URI レファレンスを指定します	<uri name="http://www.w3.org/2009/XMLSchema/XMLSchema.xsd" uri="files/XMLSchema.xsd"/>
rewriteURI	• uriStartString は再書き込みする URI の開始部分を指定します。 • Uri 置換文字列を指定します（例えば、ローカルディレクトリに対する相対的なパス）	<rewriteURI uriStartString='http://www.altova.com/schemas/svg/' rewritePrefix='files/'/>
rewriteSystem	• systemIdStartString は再書き込みするシステムの識別子の開始部分を指定します。 • rewritePrefix 置換文字列を指定します（例えば、ローカルディレクトリに対する相対的なパス）	<rewriteSystem systemIdStartString='http://www.altova.com/schemas/svg/' rewritePrefix='files/'/>

public、**system**、および**uri** 要素は相対的な URI が解決されるベース URI を指定する xml:base 属性を取ることができます。詳細に関してはXML カタログ仕様（http://www.oasis-open.org/committees/entity/spec-2001-08-06.html）を参照してください。

環境変数

下のテーブルはW indows 上での多種のシステムロケーションへのパスを指定する **nextCatalog** 要素内でサポートされるすべての環境変数をリストしています。

%AltovaCommonFolder%	すべてのAltova プログラムで共通のファイルを保管するために使用されているディレクトリへのフルパス。オペレーティングシステムとMapForce（32-ビットまたは64-ビット）のプラットフォームにより、パスはC:\Program Files\Altova\Common2021 またはC:\Program Files (x86)\Altova\Common2021 になります。
%DesktopFolder%	デスクトップ上にファイルオブジェクトを保管するために使用されるディレクトリへのフルパス。典型的なパスはC:\Users\Username\Desktop です。
%ProgramMenuFolder%	ユーザーのプログラムグループを含む自身がファイルシステムのディレクトリであるディレクトリへのフルパス。典型的なパスはC:\Users\Username\AppData\Roaming\Microsoft\Windows\Start Menu\Programs です。
%StartMenuFolder%	ユーザーの起動メニューアイテムを含むディレクトリへのフルパス。典型的なパスはC:\Users\Username\AppData\Roaming\Microsoft\Windows\Start Menu です。
%StartUpFolder%	ユーザーの起動プログラムグループに対応するディレクトリへのフルパス。典型的なパスはC:\Users\Username\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup です。

%TemplateFolder%	ドキュメントテンプレートのための共通レポジトリとしての役割を果たすディレクトリへのフルパス。典型的なパスは C:\Users\username\AppData\Roaming\Microsoft\Windows\Templates です。
%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。典型的なパスは C:\Users\Username\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Administrative Tools です。
%AppDataFolder%	アプリケーション固有のデータのための共通レポジトリとしての役割を果たすファイルシステムディレクトリ。典型的なパスは C:\Users\username\AppData\Roaming です。
%FavoritesFolder%	現在のユーザーの[お気に入り]ディレクトリへフルパス。典型的なパスは C:\Users\Username\Favorites です。
%PersonalFolder%	現在のユーザーの[個人]ディレクトリへのフルパス。典型的なパスは C:\Users\Username\Documents です。
%SendToFolder%	宛先 のメニューアイテムを含むディレクトリへのフルパス。典型的なパスは C:\Users\username\AppData\Roaming\Microsoft\Windows\SendTo です。
%FontsFolder%	システムフォントディレクトリへのフルパス。典型的なパスは C:\Windows\Fonts です。
%ProgramFilesFolder%	プログラムファイルディレクトリへのフルパス。典型的なパスは C:\Program Files と C:\Program Files (x86) です。
%CommonFilesFolder%	共通ファイルディレクトリへのフルパス。典型的なパスは C:\Users\Public\Public Documents です。
%WindowsFolder%	(%WINDIR% 環境変数と同様) ウィンドウディレクトリへのフルパス。典型的なパスは C:\Windows です。
%SystemFolder%	システムフォルダーへのフルパス。典型的なパスは %WINDIR%\system32 です。
%CommonAppDataFolder%	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。典型的なパスは C:\ProgramData です。
%LocalAppDataFolder%	ローカル(非ローミング) アプリケーションのためのデータレポジトリとしての役割を果たすファイルシステムディレクトリへのフルパス。典型的なパスは C:\Users\username\AppData\Local です。
%MyPicturesFolder%	現在のユーザーの[ピクチャ]ディレクトリへのフルパス。典型的なパスは C:\Users\Username\Pictures です。

10 メニューレファレンス

以下のセクションでは、MapForce に用意されているメニューやメニューオプションのリストと、それらの簡単な説明を記します。

10.1 ファイル

新規作成
新規マッピングドキュメントを開きます。

開く
既に保存されたマッピング(*.mfd)ファイルを開きます。使用中のMapForce エディションで使用できない機能が含まれているファイルは、開くことができません。

上書き保存
現在アクティブなファイル名で、現在アクティブになっているマッピングを保存します。

名前をつけて保存
現在アクティブになっているマッピングを、別の名前で保存します。まだ保存されていないファイルの場合、ファイルに新たな名前をつけることができます。

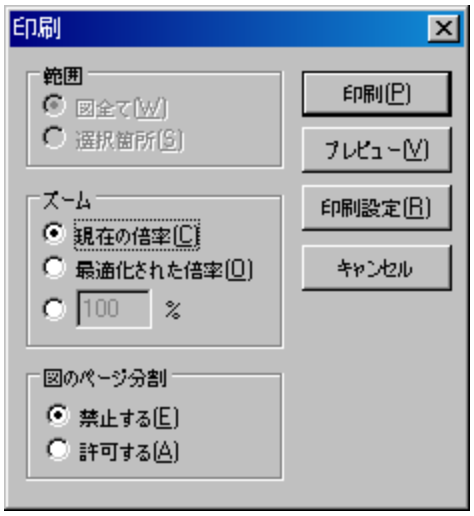
全て保存
現在開かれている全てのマッピングファイルを保存します。

再ロード
現在アクティブになっているマッピングを再ロードします。それまでに作成された変更点を破棄してもよいか尋ねられます。

閉じる
現在アクティブになっているマッピングを閉じます。ファイルを閉じる前に保存するか尋ねられます。

全て閉じる
現在開かれている全てのマッピングファイルを閉じます。保存されていないファイルを保存するか尋ねられます。

印刷
印刷ダイアログボックスが開かれ、マッピングを印刷することができます。



印刷ダイアログボックス

「**現在の倍率**」を選択すると、現在マッピングに対して指定されているズーム倍率が使用されます。「**最適化された倍率**」を選択すると、マッピングがページの大きさに収められるよう、印刷が最適化されます。コンポーネントのスクロールバーは印刷されません。1 つのコンポーネントが複数のページに切り分けて印刷してもよいか指定することもできます。

印刷プレビュー

上に記述された機能を持つ印刷ダイアログボックスが表示されます。

印刷設定

印刷設定ダイアログボックスが表示され、使用するプリンターや用紙の設定などを指定することができます。

マッピングの検証

マッピングを検証により、全てのマッピング(接続)が妥当であるかと、エラーや警告が表示されます。
詳細については[マッピングの検証]を参照ください。

マッピングの設定

マッピングの設定ダイアログボックスではドキュメントごとの設定を定義することができます。[マッピングの設定の変更] を参照してください。

選択された言語でコードを生成する

現在選択されているマッピングの言語によりコードが生成されます。現在選択されている言語は、ツールバーでハイライトされているプログラミング言語のアイコン(XSLT、XSLT2、XSLT3で確認することができます。

コード生成 | XSLT (XSLT2、XSLT3)

このコマンドにより、ソースファイルからの変換に必要な XSLT ファイルが生成されます。このオプションを選択すると、フォルダーの参照ダイアログボックスが表示され、生成された XSLT ファイルを配置する場所を指定することができます。[マッピング設定の変更] を参照してください。

最近のプロジェクト

最近開かれたプロジェクトのリストを表示します。

終了

アプリケーションを終了します。保存されていないドキュメントがある場合、それらを保存するか尋ねられます。

10.2 編集

このメニュー以下にあるコマンドの殆どは、出力タブにてマッピングの結果を参照している時、またはXSLT タブにて XSLT コードのプレビューを行なっている際に利用することができます。

元に戻す 

MapForce では使用回数に制限の無い「元に戻す」操作を行うことができ、マッピングのデザインステップを順を追って確認することができます。

やり直し 

やり直しコマンドにより、元に戻すコマンドにより戻された動作を進めることができます。これらのコマンドを使用することで、操作の履歴を確認することができます。

検索 

XSLT、XSLT2、XSLT3 または出力タブにて、指定されたテキストを検索することができます。

次を検索 F3 

同じ検索文字列が次に出現する箇所を検索することができます。

前を検索 Shift F3 

同じ検索文字列が前に出現する箇所を検索することができます。

切り取り/コピー/貼り付け/削除

標準的な W indows の編集コマンドです。マッピングウィンドウ内に表示されているコンポーネントや関数を切り取ったりコピーすることができます。

すべて選択

マッピングタブにある全てのコンポーネントを選択する、または**XSLT、XSLT2、XSLT3、** または出力タブに表示されているテキスト/コード全体を選択することができます。

10.3 挿入

XML スキーマ / ファイル 

XML スキーマまたはインスタンスをマッピングに追加します。スキーマを参照する XML ファイルを選択すると、追加情報をマッピングに追加する必要はありません。スキーマ参照のない XML ファイルを選択すると、自動的に一致する XML スキーマを生成するかを問われます。(XML スキーマの生成 を参照してください)。XML スキーマファイルを選択すると、プレビューのためのデータを提供する XML インスタンスを任意で追加するかを問われます。

入力の挿入 

マッピングウィンドウがマッピングを表示すると、このコマンドにより入力コンポーネントをマッピングに追加することができます。(マッピングにパラメーターを与える を参照してください)。マッピングウィンドウがユーザー定義関数を表示すると、このコマンドはユーザー定義関数の入力コンポーネントを追加します。(ユーザー定義関数内のパラメーター を参照してください)。

出力の挿入 

マッピングウィンドウがマッピングを表示すると、このコマンドにより出力コンポーネントをマッピングに追加することができます(マッピングから文字列の値を返す を参照してください)。マッピングウィンドウがユーザー定義関数を表示すると、このコマンドはユーザー定義関数の出力コンポーネントを追加します。ユーザー定義関数内のパラメーター を参照してください。

定数 

コネクタ に固定のデータを提供する定数を挿入します。コンポーネントを作成する際にデータはダイアログボックスに入力されます。以下のデータ型から選択することができます: 文字列、数値、その他から選択することができます。

変数 

標準的な(非インライン型の)ユーザー定義関数と等価な中間変数を挿入します。変数はインスタンスファイルを持たない構造コンポーネントで、マッピング処理を単純にするために使用されます。詳細については 中間変数 を参照ください。

並べ替え: node/row

ノードの並べ替えを許可するコンポーネントを挿入します。詳細に関しては、ノード/行の並べ替えを参照してください。

フィルター: node/row 

node/row と bool、そして on-true と on-false という 2 つの入力と出力パラメーターを備えたコンポーネントを挿入します。Boolean が true となっている場合、node/row パラメーターの値が on-true パラメーターから渡され、Boolean が false となっている場合、node/row パラメーターの値が on-false パラメーターへ渡されます。フィルターの使用方法については フィルターと条件 を参照してください。

Value-Map 

ルックアップテーブルを使って入力された値を特定の値へ変換するコンポーネントを挿入します。(例えば、月の数値を月の名前にマップするなど)他の値のセットに値のセットをマップする場合に役に立ちます。詳細については Value-Map の使用 を参照ください。

IF-Else 条件 

"If-Else 条件"のかたのコンポーネントを挿入します。詳細に関しては、フィルターと条件 を参照してください。

10.4 コンポーネント

ルート要素の変更

XML インスタンスドキュメントのルート要素を変更することができます。

XMLSpy でスキーマ定義を編集

XML スキーマコンポーネントが選択された状態でこのオプションを選択すると、XMLSpy のスキーマビューで XML スキーマファイルが開かれ、編集を行うことができます

前に入力を複製して追加

選択されたアイテムのコピー/クローンを、現在選択されているアイテムの前に挿入します。コピーされたアイテムには出力アイコンが含まれておらず、データ構造として使用することはできません。この機能を使用した例については 複数のソースから1 つのターゲットにマップ セクションを参照ください。複製されたアイテムを右クリックすると、コンテキストメニューから複製アイテムを移動することができます。必要に応じて複製されたアイテムを上/下へ移動させてください。

後に入力を複製して追加

選択されたアイテムのコピー/クローンを、現在選択されているアイテムの後に挿入します。コピーされたアイテムには出力アイテムが含まれておらず、データ構造として使用することはできません。この機能を使用した例については 複数のソースから1 つのターゲットにマップ セクションを参照ください。複製されたアイテムを右クリックすると、コンテキストメニューから複製アイテムを移動することができます。必要に応じて複製されたアイテムを上/下へ移動させてください。

複製を削除

既に作成された複製アイテムを削除します。この機能を使用した例については 複数のソースから1 つのターゲットにマップ セクションを参照ください。

ツリーを左揃えにする

全てのアイテムをコンポーネントの左端へ揃えます。

ツリーを右揃えにする

全てのアイテムをコンポーネントの右端へ揃えます。この表示方法はターゲットスキーマへのマッピングを行う際に利用することができます。

プロパティ

現在選択されているコンポーネントの設定を表示するダイアログボックスを表示します。 コンポーネント設定を変更するを参照してください。

10.5 接続

子要素の自動接続

子要素の自動接続機能を有効化または無効化します。ツールバーにあるアイコンからも選択を行うことができます。

子要素の接続設定

一致する子要素の接続設定ダイアログボックスが表示され、接続設定を行うことができます。(一致する子要素の接続を参照)。

一致する子要素の接続

このコマンドにより、ソースとターゲットスキーマ間で同じ名前を持つ複数のアイテムに対して接続を作成することができます。このダイアログボックスで定義した設定はそのまま保持され、ツールバーにある子要素の自動接続アイコン  が有効になっている状態で2つのアイテムを接続した際に適用されます。このアイコンをクリックすることで、アイコンの有効/無効状態を切り替えることができます。詳細については一致する子要素の接続のセクションを参照ください。

標準マッピング(ターゲット優先マッピング)

接続の種類を標準的なマッピングへ変更します。詳細については標準マッピング(ターゲット優先マッピング)を参照してください。

全てコピー(子要素)

親コネクタのサブツリーとして表示されている子アイテムのコネクタが存在する状況で、マッチする全ての子要素に対してコネクタを作成します。詳細については すべてコピー接続 を参照ください。

ソース優先マッピング(混合コンテンツ)

接続をソース優先 / 混合コンテンツ型のコネクタへ変更して、新たに要素を選択してマッピングすることができるようになります。詳細に関しては、ソース優先マッピング(混合コンテンツ)を参照してください。

プロパティ

現在のコネクタに対して固有(混合コンテンツ)の設定を定義するためのダイアログボックスが表示されます。利用することができないオプションはグレーアウトされます。これらの設定はテキストノードを持たない複合型のアイテムに対しても適用されます。詳細は接続設定を参照してください。

10.6 関数

ユーザー定義関数の作成

新たなユーザー定義関数を作成します。(詳細については ユーザー定義関数 を参照ください)。

選択からユーザー定義関数作成

マッピングウィンドウにて現在選択されている要素をベースに、新たなユーザー定義関数を作成します。

関数設定

現在アクティブなユーザー定義関数の設定ダイアログボックスを開き、現在の設定を変更することができます。

関数削除

既存のユーザー定義関数を開き、その関数の名前がタブに表示されている状態で、現在アクティブなユーザー定義関数を削除します。

入力の挿入

マッピングウィンドウがマッピングを表示すると、このコマンドは、マッピングに入力コンポーネントを追加します(詳細については 単純型入力 を参照ください)。マッピングウィンドウがユーザー定義関数を表示すると、このコマンドは、マッピングに入力コンポーネントをユーザー定義関数に追加します(詳細については ユーザー定義関数内のパラメーター を参照ください)。

出力の挿入

マッピングウィンドウがマッピングを表示すると、このコマンドは、マッピングに出力コンポーネントを追加します(詳細については 単純型出力 を参照ください)。マッピングウィンドウがユーザー定義関数を表示すると、このコマンドは、マッピングに出力コンポーネントをユーザー定義関数に追加します(詳細については ユーザー定義関数内のパラメーター を参照ください)。

10.7 出力

XSLT 1.0、XSLT 2.0、XSLT 3.0、XQuery、Java、C#、C++、Built-in 実行エンジン

マッピングが実行される変換言語を設定します。変換言語の選択を参照してください。

出力ファイルの検証

参照されているスキーマに対して出力 XML ファイルを検証します。

生成された出力の保存

出力タブに現在表示されているデータを保存します。

生成されたすべての出力を保存

動的マッピングにより生成された出力をファイルに保存します。詳細については 複数の入力または出力ファイルを動的に処理 を参照ください。

出力を再生成

現在のマッピングを出力ウィンドウから再生成します。

ブックマークの挿入/削除

出力ウィンドウのカーソル位置にブックマークを挿入します

次のブックマーク

出力ウィンドウ内にある次のブックマークへ移動します。

前のブックマーク

出力ウィンドウ内にある前のブックマークへ移動します。

全てのブックマークを削除

出力ウィンドウにて現在定義されている全てのブックマークを削除します。

XML テキストの整形

出力ペインに表示されている XML ドキュメントの表示を、ドキュメントの構造に従ったかたちに再構成します。各子ノードが、親ノードよりインデントされたかたちで表示されます。この機能では、タググループにて指定されたタブサイズの設定（タブまたはスペースを挿入）が使用されます。

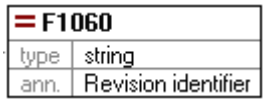
テキストビューの設定

テキストビュー設定ダイアログボックスを表示します。このダイアログボックスにより **出力** ペインと **XSLT** ペインでテキストビュー設定をカスタマイズすることができます。またこのダイアログボックスは、現在ウィンドウ内で適用されている定義済みのホットキーを表示しています。詳細に関しては、次を参照してください テキストビュー機能。

10.8 表示

注釈の表示

XML スキーマの注釈 をコンポーネントウィンドウにて表示します。
型の表示アイコンも有効になっている場合、両方の情報がグリッド形式で表示されます。



型の表示

各要素または属性に対してスキーマのデータ型を表示します。
注釈の表示アイコンも有効になっている場合、両方の情報がグリッド形式で表示されます。

関数ヘッダーにライブラリ名を表示

関数タイトルに括弧付きでライブラリ名を表示します。

ヒントの表示

マウスポインターが関数の上に配置された時に、ツールチップ内に関数の説明が表示されます。

選択されたコンポーネントの接続線の表示

すべてのマッピングの接続線または、現在選択されているコンポーネントに関連した接続線を切り替えることができます。

ソースからターゲットへの接続線表示

以下のように接続線の表示を切り替えることができます:

- 現在選択されているコンポーネントに直接接続されているコネクタ
- 現在選択されているコネクタで、ソースからターゲットコンポーネントまで

ズーム

ズームダイアログボックスが表示されます。ズームの倍率を手動で入力することができるほか、スライダーをドラッグして、ズームの倍率を変更することができます。

戻る

マッピングタブにて現在開かれているマッピングで、前に表示されていたマッピングを表示します。

進む

マッピングタブにて現在開かれているマッピングで、戻るコマンドにより表示される前のマッピングを表示します。

ステータスバー
メッセージウィンドウの下に表示されているステータスバーを表示/非表示にします。

ライブラリウィンドウ
ライブラリ関数が含まれるライブラリウィンドウを表示/非表示にします。

ライブラリの管理
ライブラリの管理ウィンドウを表示/非表示にします。

メッセージウィンドウ
検証出力ウィンドウを表示/隠します。コードの生成を行う場合、メッセージ出力ウィンドウが自動的に有効になり、検証結果が表示されます。

概要ウィンドウ
概要ウィンドウを表示/非表示にします。長方形をドラッグすることで、マッピングの表示範囲を移動することができます

10.9 ツール

グローバルリソース

グローバルリソースの管理ダイアログボックスを表示して、グローバルリソース XML ファイル内にあるグローバルリソース情報を追加、編集、または削除することができます。詳細については グローバルリソース - プロパティ を参照ください。

アクティブな構成

現在アクティブになっているグローバルリソースを、グローバルリソースの構成リストから選択/変更することができます。サブメニューから目的の構成を選択してください。

反転マッピングの作成

MapForce で現在アクティブになっているマッピングをベースに、「反転した」マッピングを作成します。反転した結果生成されるマッピングは完成したものではなく、コンポーネント間における直接接続だけが保持されるという点に注意してください。恐らく反転したマッピングは妥当なものではなく、更なる編集を行うこと無く出力タブをクリックしても、実行されません。

マッピングを反転させると、ソースコンポーネントがターゲットコンポーネントになり、ターゲットコンポーネントがソースコンポーネントになります。入力または出力 XML インスタンスファイルがコンポーネントに割り当てられている場合、これらも交換されます。

以下のデータが保持されます:

- コンポーネント間の直接接続
- チェーンマッピング内のコンポーネント間の直接接続
- 接続の種類: 標準、混合コンテンツ、全てコピー
- パススルーコンポーネント設定
- データベースコンポーネント

以下のデータが保持されません

- 関数やフィルターなどを経由した接続は、関数などとともに削除されます
- ユーザー定義関数
- Web サービスコンポーネント

ツールバーとウィンドウの復元

ツールバーやドックウィンドウなどをデフォルトの状態にリセットします。変更を反映するには、MapForce を再起動する必要があります。

カスタム化...

MapForce グラフィカルなユーザーインターフェイスをカスタム化することのできるダイアログボックスを開きます。これにはツールバーの表示、非表示、およびメニューとキーボードのショートカットのカスタム化が含まれます。

オプション

デフォルトのMapForce 設定を変更することのできるダイアログボックスを開きます（MapForce オプションの変更 を参照してください）。

10.10 ウィンドウ

重ねて表示

開かれている全てのウィンドウを重ねて表示されるように再配置します。

上下に並べて表示

開かれている全てのウィンドウを**上下に並べて表示**することで、同時に表示されるように再配置します。

左右に並べて表示

開かれている全てのウィンドウを**左右に並べて表示**することで、同時に表示されるように再配置します。

1
2
現在開かれているウィンドウが表示され、これらウィンドウ間で素早い切り替えを行うことができます。
Ctrl + tab または Ctrl + F6 キーボードショートカットを使用することでも、開かれているウィンドウを切り替えることができます。

10.11 ヘルプメニュー

- 目次
  - *説明*

    ヘルプウィンドウの左側のペインに目次を表示した、MapForce の画面上のヘルプマニュアルを開きます。目次はヘルプドキュメント全体の概要を表示しています。目次のエントリをクリックしてトピックに移動することができます。

- インデックス
  - *説明*

    ヘルプウィンドウの左側のペインにキーワード インデックスを表示したMapForce の画面上のヘルプマニュアルを開きます。目次はヘルプドキュメント全体の概要を表示しています。インデックスはキーワードをリストし、キーワードをダブルクリックすることでトピックへ移動することができます。キーワードが 1 つ以上のトピックにリンクされている場合は、トピックのリストが表示されます。

- 検索
  - *説明*

    ヘルプウィンドウの左側のペインに検索ダイアログを表示したMapForce の画面上のヘルプマニュアルを開きます。単語を検索するには、入力フィールドに検索対象を入力して、(i) 「Return」 を押す、または(ii) 「**トピックのリスト**」をクリックします。 を押します。ヘルプシステムは、ヘルプドキュメント全体で全文検索を行いヒットしたリストを返します。アイテムを表示するためにはアイテムをダブルクリックします。

- ソフトウェアのライセンス認証
  - *説明*

    Altova 製品ソフトウェアをダウンロードすると、無料評価キーまたは購入されたライセンスキーを使用して、製品にライセンスを供与、または、ライセンスの認証を行うことができます。

    - *無料 評価ライセンス*初めて製品のダウンロードとインストールを行うと、ソフトウェアライセンス認証ダイアログが表示されます。ダイアログでは無料 評価 ライセンスをリクエストすることができます。ユーザーの名前、所属会社名、そして電子メールアドレスを表示されるダイアログに入力し **リクエスト**をクリックします。ライセンスファイルが入力された電子メールアドレスに送信されます。この手順には数分を要します。ライセンスファイルを適切な場所に保存します。**リクエスト**をクリックすると、リクエストダイアログの下に入力フィールドが表示されます。このフィールドはライセンスファイルの パスを取ります。ライセンスファイルを参照 またはライセンスファイルへの パスを入力し「OK」をクリックします。(「ソフトウェアのライセンス認証ダイアログ」内で **新規のライセンスをアップロードする**」 をクリックしてライセンスファイルへの パスを入力するダイアログにアクセスすることができます。)ソフトウェアは30 日の間アンロックされます。
    - *永続的なライセンスキー*:ソフトウェアライセンス認証ダイアログには永続的なライセンスキーを購入するためのボタンが含まれています。このボタンをクリックすると、製品の永続的なライセンスキーを購入することのできる Altova オンラインショップに移動することができます。受信する電子メールはライセンスデータを含むライセンスファイルの書式で送信されています。3 つの種類の永続的なライセンスが存在します: インストール済み、同時使用ユーザー、名前を持つユーザー。*インストール済みのライセンス* は単一のコンピューター上のソフトウェアのロックを解除します。*インストール済みのライセンス* を N 台のコンピューターのために購入すると N 台までのコンピューター上でソフトウェアを使用することができます。*同時使用ユーザーライセンス* は N 人の同時使用ユーザーに N 人のユーザーに同時にソフトウェアを使用することを許可します 。(10N 台のコンピューターにソフトウェアをインストールすることができます)。 *名前を持つユーザーライセンス*は*特定のユーザー*が 5 台の異なるコンピューター上でソフトウェアを使用することを許可します。ソフトウェアのライセンスを認証するには「**新規のライセンスをアップロードする**」 をクリックして表示されるダイアログ内でライセンスファイルを参照、またはライセンスファイルの パスを入力し「OK」をクリックします。

メモ　複数のユーザーライセンスのために各ユーザーは各自の名前を入力するようにプロンプトされます。

ライセンスの電子メールとAltova 製品へのライセンス供与(有効化) の異なる方法
Altova から受信するライセンス電子メールにはライセンスファイルが添付ファイルとして含まれています。ライセンスファイルは `.altova_licenses` ファイル拡張子を有しています。

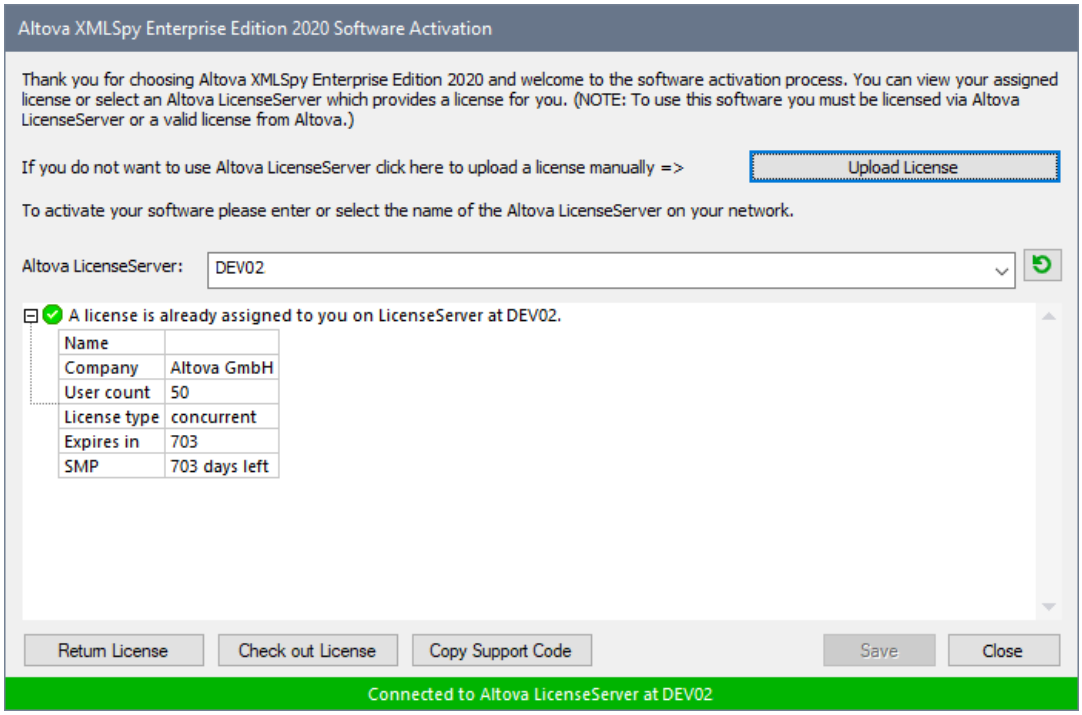
Altova 製品のライセンスを認証するには、以下のうち1つ行ってください:

- 適切な場所にライセンスファイル(`.altova_licenses`) を保存し、ライセンスファイルをダブルクリックし、表示されるダイアログに必要な情報を入力し、**「キーの適用」**をクリックして完了します。
- ライセンスファイル(`.altova_licenses`) を適切な場所に保存します。Altova 製品内では、**「ヘルプ | ライセンス登録」**メニューコマンドを選択し、**新規のライセンスをアップロード**します。ライセンスファイルへのパスを入力し**「OK」**をクリックします。
- 適切な場所にライセンスファイル(`.altova_licenses`) を保存し、Altova LicenseServer のライセンスプールにアップロードします。以下を行うことができます: (i) 製品のソフトウェアライセンス認証ダイアログから Altova 製品からライセンスを取得します。*(以下を参照)*または(ii) Altova LicenseServer から製品へのライセンスを割り当てます。*LicenseServer の使用の詳細に関しては、下記のトピックを参照してください*。

ソフトウェアライセンス認証ダイアログ(*下のスクリーンショット*) は、**「ヘルプ | ソフトウェア アクティベーション」**をクリックすることにより常にアクセスすることができます。

以下の方法によりソフトウェアをアクティブ化することができます:

- *ソフトウェアライセンス認証ダイアログでライセンスを登録する方法*。ダイアログ内で、**「新規のライセンスをアップロード」**をクリックして、ライセンスファイルを参照し選択します。**「OK」**をクリックしてライセンスファイルへのパスを確認します(複数のユーザーライセンスの場合は個人の名前です)。**「保存」**をクリックして完了します。
- *ネットワーク上のAltova LicenseServer を使用してライセンス供与する方法:* ネットワーク上の Altova LicenseServer を使用してライセンスを取得するにはソフトウェアのライセンスの認証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。使用する LicenseServer がインストールされているマシンを選択します。License Servers の自動検知は LAN 上で配信が送信されることを意味します。これらの配信がサブネットに制限されているため、License Server は自動検知のためのクライアントマシンと同じサブネット上に存在する必要があります。自動検知が作動しない場合、サーバーの名前を入力します。Altova LicenseServer はライセンスプール内で Altova 製品のためのライセンスを有している必要があります。LicenseServer プール内に存在する場合、ソフトウェアライセンス認証ダイアログ内に表示されます(*Altova XMLSpy 内のダイアログで表示されている例を参照してください*)。**「保存」**をクリックしてライセンスを取得します。



*マシン固有のライセンス*がLicenseServer からインストールされると、7日間は、LicenseServer に戻すことができません。7日過ぎると、(「**ライセンスを戻す**」をクリックして) *マシンのライセンス*をLicenseServer に戻すことができ、*このライセンス*は、*他のクライアント*によりLicenseServer から取得することができます。LicenseServer 管理者は、LicenseServer のW eb UI を使用して、取得された*ライセンス*の割り当てを解除することができます。*ライセンス*の返却は、*マシン固有のライセンスのみ*に適用され、現在使用中の*ライセンス*には適用されないことにご注意ください。

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。この機能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*するには以下をおこないます: (i) *ソフトウェアのライセンス*の認証ダイアログで「**ライセンスのチェックアウト**」をクリックします(*上のスクリーンショット参照*)。(ii) *ライセンスのチェックアウト*ダイアログ内から、*チェックアウト*の期間を選択し、「**チェックアウト**」をクリックします。*ライセンス*が*チェックアウト*されます。*ライセンスのチェックアウト*後2つの状態が発生します: (i) *ソフトウェアのライセンス*認証ダイアログは時刻および*チェックアウト*の期限を含む*チェックアウト*に関する情報を表示します。(ii) ダイアログ内の「**ライセンスのチェックアウト**」ボタンは「**チェックイン**」ボタンに変更されます。「**チェックイン**」ボタンをクリックして、*ライセンス*をチェックインすることができます。*チェックアウト*期間の期限が切れると、*ライセンス*は自動的にチェックイン状態に戻されるため、選択した*チェックアウト*の期間が*オフライン*で作業する期間をカバーするように確認してください。

メモ　*ライセンスのチェックアウト*を可能にするには、LicenseServer 上で*チェックアウト*機能が有効化されている必要があります。*チェックアウト*を試みる際この機能が有効化されていない場合、エラーメッセージが表示されます。この場合、LicenseServer 管理者に連絡してください。

サポートコードのコピー
「**サポートコードのコピー**」をクリックして、ライセンスの詳細をクリップボードにコピーしてください。これはオンラインサポートフォームを使用してサポートをリクエストする際に必要なデータです。

Altova LicenseServer を使用することにより、IT 管理者は、リアルタイムでネットワーク上の全てのライセンスの概要、および、クライアントの割り当てと、クライアントのライセンスの使用状況を確認することができます。LicenseServer を使用する利点は、ですから、多数の Altova ライセンスを管理することのできる管理機能です。Altova LicenseServer は、Altova Web サイトで無料で提供されています。Altova LicenseServer および Altova LicenseServer を使用したライセンスの供与に関する詳細は、Altova LicenseServer ドキュメントを参照してください。

▼ 注文フォーム

▣ *説明*

ソフトウェア製品のライセンス許与バージョンを注文する準備が整っている場合、(*前のセクション参照*) ソフトウェアライセンス認証ダイアログ内の「**永久ライセンスの購入**」ボタン、または「**注文フォーム**」コマンドを使用して Altova オンラインショップに移動して注文することができます。

▼ 登録

▣ *説明*

Altova 製品登録ページをブラウザーのタブに表示します。Altova ソフトウェアを登録することにより、最新の製品の情報が得られます。

▼ 更新のチェック

▣ *説明*

Altova サーバーに接続して、より新しいバージョンの製品が利用可能かどうかチェックし、その結果を表示します。

▼ サポートセンター

▣ *説明*

インターネット上にある Altova サポートセンターへのリンクとなっています。サポートセンターには FAQ やディスカッションフォーラムが含まれており、問題の解決方法を探り、Altova の技術サポートスタッフへアクセスすることができます(現在英語のみの提供となります)。

▼ WEB 上の FAQ

▣ *説明*

インターネット上にある Altova の FAQ へのリンクとなっています。FAQ データベースは Altova のサポートスタッフにより常時更新されています。

▼ コンポーネントのダウンロード

▣ *説明*

インターネット上にある Altova のコンポーネントダウンロードセンターへのリンクとなっています。このリンク先から様々なコンポーネン

トソフトウェアをダウンロードして、Altova 製品とともに使用することができます。ソフトウェアコンポーネントはXSLT やXSL-FO プロセッサーからアプリケーションサービスプラットフォームまで、幅広く提供されています。コンポーネントダウンロードセンターにてご利用になれるソフトウェアは、通常無料でご利用になれます。

- インターネット上のMapForce
  - *説明*

    インターネット上にあるAltova W eb サイトへのリンクとなっています。Altova W eb サイトでは、MapForce や関連するテクノロジーについて確認することができます。

- MapForce トレーニング
  - *説明*

    Altova W eb サイトでオンライントレーニングページを見つけることができますAltova の専門家トレーナーによるオンラインコースを選択することができます。

- MapForce について
  - *説明*

    スプラッシュ画面と製品のバージョン番号が表示されます。MapForce の64 ビットバージョンを使用している場合、これはアプリケーション名の後のサフィックス(x64) により示されています。32 ビットバージョンにはサフィックスは存在しません。

11 付録

以下の付録にはMapForce に関する技術的な情報や、ライセンスに関する重要な情報が収められています。各付録には以下のようにサブセクションが収められています:

技術的なデータ

- OS ならびにメモリの要件
- Altova XML パーサー
- Altova XSLT とXQuery エンジン
- Unicode のサポート
- インターネットの使用
- ライセンス使用状況測定

ライセンス情報

- 電子的なソフトウェアの配布
- 著作権
- 使用許諾契約書

11.1 エンジン情報

このセクションにはAltova XML バリデーター、Altova XSLT 1.0 エンジン、Altova XSLT 2.0 エンジン、そしてAltova XQuery エンジンの実装に特化した情報が含まれます。

11.1.1 XSLT および XQuery エンジンに関する情報

MapForce のXSLT およびXQuery エンジンは、W 3C 仕様に従っています、ですから、XMLSpy の以前のバージョン内のAltova エンジンより厳密です。この結果、以前のエンジンで無視されていた小さなエラーが、MapForce 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
- XQuery 1.0

11.1.1.1 XSLT 1.0

MapForce のXSLT 1.0 エンジンは、W orld W ide W eb Consortium(ワールド・ワイド・ウェブ・コンソーシアム) (W 3C) の1999 年 11 月 16 日版のXSLT 1.0 勧告 および1999 年 11 月 16 日版のXPath 1.0 勧告 に準拠します。実装に関しての以下の情報にご注意してください。

実装についての注意点

`xsl:output` の`method` 属性がHTMLに設定された場合、または、がHTML 出力 デフォルトで選択されている場合、XML またはXSLT ファイル内の特殊文字はHTML ドキュメントにHTML 文字参照として出力内に挿入されます。例えば、文字 U+00A0 (ブレーク無しのスペースのための16 進数レファレンス) がHTML コード内に文字の参照(` ` or ` `) 、または、エンティティ参照 ` ` として挿入されます

11.1.1.2 XSLT 2.0

このセクション:

- エンジン 適合性
- 下位互換性
- 名前空間
- スキーマ認識
- 実装固有の振る舞い

適合性

MapForce のXSLT 2.0 エンジンは、W orld W ide W eb Consortium（ワールド・ワイド・ウェブ・コンソーシアム）（W 3C）の2007 年 1 月 23 日版のXSLT 2.0 勧告 および2010 年 12 月 14 日版のXPath 2.0 勧告に準拠します。

下位互換性

XSLT 2.0 エンジン は下位互換性を有します。XSLT 2.0 エンジンの 下位互換性が有効になるのは、 XSLT 1.0 スタイルシートを処理するためにXSLT 2.0 エンジン が使用される際です。XSLT 1.0 エンジン と下位互換性を持つXSLT 2.0 エンジンにより作成される出力に違いがあるかもしれないことに注意してください。

名前空間

XSLT 2.0 スタイルシートは、 XSLT 2.0. プレフィックス内で使用することのできる型コンストラクタ および関数を使用するため、以下の名前空間を宣言する必要があります。 下のリストは通常使用されるリスト です。希望する場合は、代替プレフィックスを使用することもできます。

名前空間	プレフィックス	名前空間 URI
XML スキーマ型	xs:	http://www.w3.org/2001/XMLSchema
XPath 2.0 関数	fn:	http://www.w3.org/2005/xpath-functions

通常これらの名前空間は、以下のリスト で表示されるようにxsl：スタイルシート または、 xsl:transform 要素で宣言されます:

```
<xsl：スタイルシート version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    ...
</xsl：スタイルシート>
```

次の点に注意してください:

- XSLT 2.0 エンジンは、（上のテーブルでリストされている） XPath 2.0 および XQuery 1.0 関数 名前空間 をデフォルト の関数名前空間として使用します。XPath 2.0 および XSLT 2.0 関数をプレフィックス無しでスタイルシート 内で使用することができます。XPath 2.0 関数 名前空間 をスタイルシート 内でプレフィックスと共に宣言すると、 割り当てられた宣言内でプレフィックスを追加して使用することができます。
- XML スキーマ 名前空間から型コンストラクタと型を使用する場合、名前空間 宣言内で使用された、プレフィックスを使用して型コンストラクタを呼び出さなければなりません（例えば、 xs:date）。
- XPath 2.0 関数の一部は、XML スキーマ データ型と同じ名前を保有します。例えば、 XPath 関数 fn:string および fn:boolean のためには、同じロケーション名: xs:string および xs:boolean を持つ、XML スキーマ データ型 が存在します。ですから、 XPath 式 string('Hello')を使用する場合、式は、 xs:string('Hello') ではなくて fn:string('Hello') として検証します。

スキーマ認識

XSLT 2.0 エンジンは、スキーマを認識します。ですから、ユーザー定義 スキーマ型 および xsl:validate 命令を使用することができます。

実装固有の振る舞い

以下は、XSLT 2.0 エンジンが、特定のXSLT 2.0 関数の振る舞いの実装-特定のアスペクトをどのように扱うかの説明です。

xsl:result-document
追加してサポートされるエンコードは以下の通りです(Altova-固有): `x-base16tobinary` および `x-base64tobinary`.

function-available
インスコープ関数の使用をテストする関数(XSLT、XPath、および拡張関数)。

unparsed-text
`href` 属性は、以下を受け入れます(i) ベース-uri フォルダー内のファイルの相対パス、および(ii) 相対パスを持つまたは持たない `file://` プロトコール。追加してサポートされるエンコードは以下の通りです(Altova-固有): `x-binarytobase16` および `x-binarytobase64`.

unparsed-text-available
`href` 属性は、以下を受け入れます(i) ベース-uri フォルダー内のファイルの絶対パス、および(ii) 絶対パスを持つまたは持たない `file://` プロトコール。追加してサポートされるエンコードは以下の通りです(Altova-固有): `x-binarytobase16` および `x-binarytobase64`.

メモ RaptorXML の先行製品である AltovaXML で実装されていた以下のエンコード値は使用しないでください: `base16tobinary`, `base64tobinary`, `binarytobase16` and `binarytobase64`.

11.1.1.3 XQuery 1.0

このセクション:

- エンジン 適合性
- スキーマ認識
- エンコード
- 名前空間
- XML ソースと検証
- 静的および動的な型のチェック
- ライブラリモジュール
- 外部関数
- 照合順序
- 数値データの精度
- XQuery 命令サポート

適合性

MapForce のXQuery 1.0 エンジンは、W orld W ide W eb Consortium(ワールド・ワイド・ウェブ・コンソーシアム)(W 3C) の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定義済みです。

名前空間	プレフィックス	名前空間 URI
XML スキーマ型	xs:	http://www.w3.org/2001/XMLSchema
スキーマインスタンス	xsi:	http://www.w3.org/2001/XMLSchema-instance
内蔵の関数	fn:	http://www.w3.org/2005/xpath-functions
Local 関数	local:	http://www.w3.org/2005/xquery-local-functions

次の点に注意してください：

- XQuery 1.0 エンジンは、上にリストされたプレフィックスを名前空間に対応するバウンドとして認識します。
- Since the 上にリストされた内蔵の関数 名前空間は、XQuery 内のデフォルトの関数です。内蔵の関数 が呼び出される際、名前空間、fn: プレフィックス を使用する必要はありません。（例えば、 `string("Hello")` が `fn:string` 関数を呼び出す場合。）しかし、プレフィックス `fn:` は、クエリプロローグ内で名前空間を宣言することなく内蔵の関数を呼び出す時に使用することができます。（サンプル: `fn:string("Hello")`）.
- クエリプロローグ内で `default function` 名前空間 式を宣言することにより、デフォルトの関数 名前空間をすることにより変更することができます。
- XML スキーマ 名前空間空の型を使用する場合、プレフィックス `xs:` は、名前空間を明確に宣言することなく、また、これらのプレフィックスをクエリプロローグ内でバインドすることなく使用することができます。（サンプル: `xs:date` および `xs:yearMonthDuration`。）XML スキーマ 名前空間のために、他のプレフィックスを使用する場合は、クエリプロローグ内で明確に宣言されている必要があります。（サンプル: `declare 名前空間 alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.）
- `untypedAtomic`, `dayTimeDuration`, および `yearMonthDuration` データ型 が23 January 2007 のCR 共に、XPath データ型 名前空間から XML スキーマ 名前空間へ移動されていることに注意してください。ですから以下となります: `xs:yearMonthDuration`。

関数のための名前空間、型コンストラクタ、ノード テスト、が間違って割り当てられている場合、エラーが発生します。しかし、一部の関数はスキーマデータ型と同じ名前を持つことに注意してください。 例: `fn:string` および `fn:boolean`。(`xs:string` および `xs:boolean` は宣言されています) 名前空間 プレフィックス、関数または 型コンストラクタが使用されるか決定します。

XML ソースドキュメントと検証

XQuery 1.0 エンジンを使用して、実行される XQuery ドキュメント内の XML ドキュメントは、整形式である必要があります。しかし、XML スキーマに従い有効である必要はありません。XML ファイルが外部スキーマと関連付けられ、また有効な場合、ポスト スキーマ検証情報が XML データのために生成され、クエリ検証のために使用されます。

静的および動的な型のチェック

静的分析フェーズは、外部レファレンスの存在（例、モジュールのため）、呼び出された関数と変数が定義済みであるか、など構文などのクエリのアスペクトをチェックします。静的分析フェーズでエラーが検知されると、実行は停止されます。

クエリが実際に作動中にランタイム中に動的な型チェックは実行されます。型がオペレーションの必要条件と整合性を持たない場合、エラーが報告されます。例えば、式 `xs:string("1") + 1` は、エラーを返します。型 `xs:string` のオペランドを足し算のオペレーションが実行できないためです。

ライブラリモジュール

ライブラリモジュールは、再利用のため関数と変数を保管します。XQuery 1.0 エンジンは単一の外部 XQuery ファイルに保管されているモジュールをサポートします。このようなモジュールファイルはプロローグにターゲット名前空間に関連するモジュール宣言を含む必要があります。 以下はモジュールサンプルです:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

すべての関数および変数は、モジュールに関連した名前空間に属するモジュール内で宣言されています。モジュールはクエリプロローグ内の `import module` ステートメントを使用して XQuery ファイルにインポートする際に使用されます。`import module` ステートメントは、ライブラリモジュールファイル内で直接宣言された、関数と変数のみをインポートします。例:

```
import module namespace modlib = "urn:module-library" at "modulefilename.xq";
if         ($modlib:company = "Altova")
then        modlib:webaddress()
else        error("No match found.")
```

外部関数

外部関数 は、サポートされていません。例えば、`external` キーワードを仕様する式など。以下参照:

```
declare function hoo($param as xs:integer) as xs:string external;
```

照合順序

デフォルトの照合順序は、Unicode コードポイントをベースにした文字列を比較する Unicode-コードポイント照合順序です。その他にサポートされる照合順序は、ICU 照合順序 はここ にリストされるとおりです。特定の照合順序を使用する場合、サポートされる照合順序のリスト 内に与えられているとおり URI を提供します。`fn:max` と `fn:min` 関数を含む文字列の比較は、指定された照合順序により行われます。 照合順序オプションが指定されていない場合、デフォルトの Unicode-コードポイント照合順序 が使用されます。

数値データの精度

- `xs:integer` データ型には任意の精度があり、表記できる桁数に制限はありません。
- `xs:decimal` データ型には小数点の後に 20 桁の制限があります。
- `xs:float` と `xs:double` データ型には 15 桁の精度の制限があります。

XQuery 命令サポート

`Pragma` 命令 は、サポートされていません。発生した場合、無視されフォールバックの式が検証されます。

11.1.2 XSLT と XPath/XQuery 関数

このセクションでは、XPath および/または XQuery 式で使用することのできる、Altova 拡張関数と他の拡張関数をリストします。Altova 拡張関数は Altova の XSLT および XQuery エンジンで使用することができ、W3C 標準で定義された関数ライブラリで使用することのできる機能に追加して機能を提供します。

一般的な情報

以下の一般的な情報に注意してください:

- W3C 仕様により定義されているコア関数ライブラリの関数は、関数の呼び出しにプレフィックスは必要ありません。これは、XSLT および XQuery エンジンが、XPath/XQuery 関数仕様で指定されている `http://www.w3.org/2005/xpath-functions` プレフィックス無しの関数をデフォルト関数の名前空間に属するものとして読み込むためです。
- 関数において、各アイテムが引数となるようなシーケンスが期待されており、2つ以上のアイテムがシーケンスにより呼び出された場合、エラーが返されます。
- 全ての比較は Unicode コードポイントコレクションを使用することで行われます。
- QName の結果は `[prefix:]localname` という形式でシリアル化されます。

xs:decimal の精度

精度とは、数値内にある桁数のことで、仕様では少なくとも18桁が求められます。`xs:decimal` 型に結果が収められる除算の場合、端数処理を行うことなく精度は小数点以下の19桁になります。

黙示的なタイムゾーン

2つの `date`、`time`、または `dateTime` 値を比較する場合、比較する値のタイムゾーンを明らかにする必要があります。値の中にタイムゾーンが明示的に与えられていない場合、黙示的なタイムゾーンが使用されます。黙示的なタイムゾーンはシステムクロックから取得され `implicit-timezone()` 関数によりその値をチェックすることができます。

照合順序

デフォルトの照合順序は、Unicode コードポイントをベースに文字列を比較する Unicode コードポイント照合順序です。エンジンは Unicode 照合アルゴリズムを使用しています。他のサポートされる照合順序は下にリストされる ICU 照合順序 です。使用するには、サポートされる照合順序のリストの URI を提供してください（*下のテーブル*）。`max` と `min` 関数を含む、文字列の比較は、指定された照合順序に沿って行われます。照合順序オプションが指定されていない場合、デフォルトの Unicode コードポイント照合順序が使用されます。

言語	URI
`da`: デンマーク語	`da_DK`
`de`: ドイツ語	`de_AT, de_BE, de_CH, de_DE, de_LI, de_LU`
`en`: 英語	`en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW`
`es`: スペイン語	`es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE`
`fr`: フランス語	`fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG`
`it`: イタリア語	`it_CH, it_IT`
`ja`: 日本語	`ja_JP`
`nb`: ノルウェー語（ブークモール）	`nb_NO`
`nl`: オランダ語	`nl_AW, nl_BE, nl_NL`
`nn`: ノルウェー語（ニーノシュク）	`nn_NO`

pt: ポルトガル語	pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST
ru: ロシア語	ru_MD, ru_RU, ru_UA
sv: スウェーデン語	sv_FI, sv_SE

名前空間軸
名前空間軸はXPath 2.0 にて廃止されましたが、名前空間軸の使用はサポートされています。XPath 2.0 メカニズムにより名前空間情報へアクセスするには、in-scope-prefixes()、namespace-uri()、namespace-uri-for-prefix(関数を使用してください。

11.1.2.1 Altova 拡張関数

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova 拡張関数名前空間、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

W 3C のXPath/XQuery 関数仕様で定義された関数は、以下で使用することができます: (i) XSLT 子アンテキスト内のXPath 式と、(ii) XQuery 文書内のXQuery 式。このドキュメントでは、前者(XSLT 内のXPath) のコンテキストで使用することのできる関数を、**XP** シンボルと共に表示し、と称します。後者(XQuery) で使用することのできる関数はの**XQ** シンボルと共に表示され、XQuery 関数と共に作業することができます。W 3C のXSLT 仕様は、XPath/XQuery 関数の仕様ではなく、XSLT 文書内のXPath 式でも使用することのできる関数を定義します。これらの関数は、**XSLT** シンボルと共に表示され、XSLT 関数と称されます。関数を使用することのできるXPath/XQuery およびXSLT のバージョンは、関数の詳細に記載されています(*下のシンボルを参照してください*)。
XPath/XQuery およびXSLT 関数ライブラリからの関数は、プレフィックス無しでリストされています。Altova 拡張関数などの、他のライブラリからの関数はプレフィックスと共にリストされています。

XPath 関数 (XSLT 内のXPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内のXPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内のXQuery 式で使用):	**XQ1** **XQ3.1**

XSLT 関数
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

11.1.2.1.1 XSLT 関数

XSLT 拡張関数 はXSLT コンテキスト内のXPath 式にて使用することができます。(例えば、XQuery コンテキストなどの) 非-XSLT コンテキストでは使用することができません。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova 拡張関数名前空間、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

標準関数

▼ distinct-nodes [altova:]

altova:distinct-nodes(*node()****)** を **node()*** とする XSLT1 XSLT2 XSLT3
入力として1つ以上のノードを必要とし、同じセットから重複した値を持つノードを除いたノードを返します。XPath/XQuery 関数 `fn:deep-equal` を使用して比較を行うことができます。

▣ *サンプル*

- **altova:altova:distinct-nodes**(`country`) は重複した値を持つものを除く、全ての子 `country` ノード返します。

▼ evaluate [altova:]

altova:evaluate(XPathExpression *as xs:string*[**, ValueOf$p1, ... ValueOf$pN**]) XSLT1 XSLT2 XSLT3
XPath 式を必要とし、必須引数として文字列をパスします。評価された式の出力を返します。例えば **altova:evaluate('//Name[1]')** は、ドキュメント内の最初の`Name` 要素のコンテンツを返します。式 `//Name[1]` は、一重引用符を使用することにより、文字列としてパスされます。

`altova:evaluate` 関数は、オプションとして追加の引数を持つことができます。これらの引数は、`p1`, `p2`, `p3`... `pN` の名前を持つスコープ内の変数の値です。使用に関して以下の点に注意してください: (i) 変数は、`X` が整数である箇所のフォーム`pX` の名前と共に定義される必要があります。(ii) `altova:evaluate` 関数の引数は、(*上の署名参照*) 2 番目の引数からは、数値順の変数のシーケンスに対応した引数のシーケンス変数の値を与えます: `p1` to `pN`: 第 2 引数は変数 `p1` の値で、第 3 引数は、変数 `p2` の値です。(iii) 変数の値は型 `item*` である必要があります。

▣ *サンプル*

```
<xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
```

```
<xsl:value-of select="altova:evaluate($xpath, 10, 20, 'hi')" />
```
outputs "hi 20 10"

上のリストに関して、以下の点に注意してください:

- altova:evaluate 式の第 2 引数は、変数 $p1 に割り当てられた値で、第三の引数は変数 $p2 に割り当てられた値です。
- 関数の第 4 番目の引数は、引用符による囲いで表示された文字列の値です。
- xs:variable 要素の select 属性は、XPath 式を提供します。この式は xs:string の型である必要があり、一重引用符で囲まれています。

変数の使用方法を更に説明するサンプル

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, //Name[1])" />
  ```
  *最初の*Name *要素の出力値*

- ```
  <xsl:variable name="xpath" select="'$p1'" />
  <xsl:value-of select="altova:evaluate($xpath, '//Name[1]')" />
  ```
  Outputs "//Name[1]"

altova:evaluate() 拡張関数は、XSLT スタイルシート内の XPath 式が動的に評価される必要のあるシチュエーションで役に立ちます。例えば、ユーザーが並べ替えの必要条件をリクエストする場合、このシチュエーションは属性 UserReq/@sortkey に保管されます。スタイルシートでは、以下の式が使用できます: <xsl:sort select="**altova:evaluate(../UserReq/@sortkey)**" order="ascending"/>。altova:evaluate() 関数は、コンテキストノードの親の UserReq 子要素の sortkey 属性を読み込みます。sortkey 属性の値が Price の場合、Price は、altova:evaluate() 関数により返され、select 属性 <xsl:sort select="**Price**" order="ascending"/> の値になります。この sort 命令が、Order という要素のコンテキスト内で発生する場合、Order 要素は Price の子の値に従い並べ替えられます。また、@sortkey の値が、Date の場合、Order 要素は、Date の子の値に従い並べ替えられます。ですから、Order の並べ替えの条件は、ランタイムでの sortkey 属性から選択されます。これは、以下の式などでは達成することはできません: <xsl:sort select="**../UserReq/@sortkey**" order="ascending"/>。上の場合、並べ替え条件は sortkey 属性自身であり、Price または Date (または、現在の sortkey のコンテンツ) ではありません。

メモ 静的なコンテキストは、変数以外以外で、呼び出し環境の名前空間、型、機能、を含みます。ベース URI とデフォルトの名前空間は継承されます。

追加サンプル

- 静的な変数: <xsl:value-of select="$i3, $i2, $i1" />
  3 つの変数の値を出力します。

- 動的な変数を持つ動的 XPath 式:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath, 10, 20, 30)" />
  ```
  "30 20 10"*を出力します。*

- 動的な変数を持たない XPath 式:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate($xpath)" />
  ```
  出力エラー: $p3 に対して定義されている変数はありません。

▼ encode-for-rtf [altova:]

altova:encode-for-rtf(input *as xs:string*, **preserveallwhitespace** *as xs:boolean*, **preservenewlines** *as xs:boolean*) を **xs:string** とする XSLT2 XSLT3
RTF のためのコードに入力文字列を変換します。空白と新しい行は、それぞれの引数により指定される boolean の値に基づき保管されます。

[トップ]

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]

11.1.2.1.2 XPath/XQuery 関数: 日付と時刻

Altova の日付/時刻拡張関数は XPath と XQuery 式で使用することができ、XML スキーマの異なる日付および時刻データ型で保存されているデータを処理するための追加機能を提供します。このセクションの関数は、Altova の **XPath 3.0** および **XQuery 3.0** エンジンで使用することができます。これらの関数は、XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は **Altova 拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内の XPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内の XQuery 式で使用):	XQ1 XQ3.1

▼ 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
- 期間から日付、時刻、分数、および、秒数を返す
- 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:date-no-TZ
altova:dateTime-no-TZ
altova:days-in-month
altova:hours-from-dateTimeDuration-accumulated
altova:minutes-from-dateTimeDuration-accumulated
altova:seconds-from-dateTimeDuration-accumulated
altova:format-duration
altova:parse-date
altova:parse-dateTime
altova:parse-duration
altova:parse-time
altova:time-no-TZ
altova:weekday-from-date
altova:weekday-from-dateTime
altova:weeknumber-from-date
altova:weeknumber-from-dateTime

[トップ]

xs:dateTime に期間を追加する XP3.1 XQ3.1

これらの関数は**xs:dateTime** に期間を追加し、**xs:dateTime** を返します。xs:dateTime 型はCCYY-MM-DDThh:mm:ss.sss のフォーマットです。これはxs:date とxs:time フォーマットの連結で、T により区切られています。タイムゾーンサフィックス+01:00 (for example)は任意です。

add-years-to-dateTime [altova:]

altova:add-years-to-dateTime(DateTime *as xs:dateTime*, **Years** を *xs:integer*)
asxs:dateTime とする **XP3.1** **XQ3.1**
日付までの期間を年数で表示します。第 2 の引数は第 1 の引数として与えられたxs:date に追加される年数です。結果はxs:date 型です。

サンプル

- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) は2024-01-15T14:00:00 を返します。
- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -4) は2010-01-15T14:00:00 を返します。

add-months-to-dateTime [altova:]

altova:add-months-to-dateTime(DateTime *as xs:dateTime*, **Months** を *xs:integer*)
asxs:dateTime とする **XP3.1** **XQ3.1**
xs:dateTime に月数での期間を追加します(*下のサンプル参照*)。第 2 の引数は、第 1 の引数として与えられたxs:dateTime に追加される月数です。結果はxs:dateTime 型です。

サンプル

- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) 2014-11-15T14:00:00 を返します。
- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -2) 2013-11-15T14:00:00 を返します。

add-days-to-dateTime [altova:]

altova:add-days-to-dateTime(DateTime *as xs:dateTime*, **Days** *as xs:integer*) を**xs:dateTime** とする **XP3.1** **XQ3.1**
xs:dateTime に日数での期間を追加します(*下のサンプル参照*)。第 2 の引数は、第 1 の引数として与えられたxs:dateTime に追加される日数です。結果はxs:dateTime 型です。

サンプル

- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) は2014-01-25T14:00:00 を返します。
- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -8) は2014-01-25T14:00:00 を返します。

add-hours-to-dateTime [altova:]

altova:add-hours-to-dateTime(DateTime *as xs:dateTime*, **Hours** *as xs:integer*) を
xs:dateTime とする **XP3.1** **XQ3.1**
xs:dateTime に時間数での期間を追加します(*下のサンプル参照*)。第 2 の引数は、第 1 の引数として与えられたxs:dateTime に追加される時間数です。結果はxs:dateTime 型です。

サンプル

- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), 10) は2014-01-15T23:00:00 を返します。
- **altova:add-hours-to-dateTime**(xs:dateTime("2014-01-15T13:00:00"), -8) は2014-01-15T05:00:00 を返します。

▼ add-minutes-to-dateTime [altova:]

altova:add-minutes-to-dateTime(DateTime *as xs:dateTime*, **Minutes** *as xs:integer*) を **xs:dateTime** とする **XP3.1** **XQ3.1**
xs:dateTime に分数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される分数です。結果は xs:dateTime 型です。

▣ *サンプル*

- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), 45) 2014-01-15T14:55:00 を返します。
- **altova:add-minutes-to-dateTime**(xs:dateTime("2014-01-15T14:10:00"), -5) 2014-01-15T14:05:00 を返します。

▼ add-seconds-to-dateTime [altova:]

altova:add-seconds-to-dateTime(DateTime *as xs:dateTime*, **Seconds** を *xs:integer*) as**xs:dateTime** とする **XP3.1** **XQ3.1**
xs:dateTime に秒数での期間を追加します（*下のサンプル参照*）。第 2 の引数は、第 1 の引数として与えられた xs:dateTime に追加される秒数です。結果は xs:dateTime 型です。

▣ *サンプル*

- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), 20) 2014-01-15T14:00:30 を返します。
- **altova:add-seconds-to-dateTime**(xs:dateTime("2014-01-15T14:00:10"), -5) 2014-01-15T14:00:05 を返します。

[トップ]

xs:date に期間を追加する **XP3.1** **XQ3.1**

これらの関数は**xs:date** に期間を追加し、**xs:date** を返します。xs:date 型はCCYY-MM-DD フォーマットです。

▼ add-years-to-date [altova:]

altova:add-years-to-date(Date *as xs:date*, **Years** *as xs:integer*) を**xs:date** とする **XP3.1** **XQ3.1**
日付までの期間を年数で表示します。第 2 の引数は第 1 の引数として与えられた xs:date に追加される年数です。結果は xs:date 型です。

▣ *サンプル*

- **altova:add-years-to-date**(xs:date("2014-01-15"), 10) は2024-01-15 を返します。
- **altova:add-years-to-date**(xs:date("2014-01-15"), -4) は2010-01-15 を返します。

▼ add-months-to-date [altova:]

altova:add-months-to-date(Date *as xs:date*, **Months** *as xs:integer*) を**xs:date** とする **XP3.1** **XQ3.1**
日付までの期間を月数で表示します。第 2 の引数は第 1 の引数として与えられた xs:date に追加される月数です。結果は xs:date 型です。

サンプル

- **altova:add-months-to-date**(xs:date("2014-01-15"), 10) 2014-11-15 を返します。
- **altova:add-months-to-date**(xs:date("2014-01-15"), -2) 2013-11-15 を返します。

add-days-to-date [altova:]

altova:add-days-to-date(Date *as xs:date*, **Days** *as xs:integer*) を**xs:date** とする **XP3.1** **XQ3.1**
日付までの期間を日数で表示します。第 2 の引数は第 1 の引数として与えられた xs:date に追加される日数です。結果は xs:date 型です。

サンプル

- **altova:add-days-to-date**(xs:date("2014-01-15"), 10) は 2014-01-25 を返します。
- **altova:add-days-to-date**(xs:date("2014-01-15"), -8) は 2014-01-07 を返します。

[トップ]

フォーマットと期間の取得 XP3.1 XQ3.1

これらの関数は入力 **xs:duration** または **xs:string** を解析し、それぞれ an **xs:string** または **xs:duration** を返します。

format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) **as xs:string** とする **XP3.1** **XQ3.1**
第 1 の引数として提出された期間を、第 2 の引数として提出された文字列によりをフォーマットします。出力は、文字列によりフォーマットされたテキスト文字列です。

サンプル

- **altova:format-duration**(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") は "Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7" を返します。
- **altova:format-duration**(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") は "Months:03 Days:02 Hours:02 Minutes:53" を返します。

parse-duration [altova:]

altova:parse-duration(InputString *as xs:string*, **Picture** *as xs:string*) を**xs:duration** とする **XP3.1** **XQ3.1**
パターン化された文字列を最初の引数として、文字を第 2 の引数とします。入力文字列は文字をベースに解析され、xs:duration が返されます。

サンプル

- **altova:parse-duration**("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") は

"P2DT2H53M11.7S" を返します。
- **altova:parse-duration**("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") は "P3M2DT2H53M" を返します。

[トップ]

xs:time に期間を追加する XP3.1 XQ3.1

これらの関数は **xs:time** に期間を追加し、 **xs:time** を返します。 **xs:time** 型は hh:mm:ss.sss 構文フォームです。文字 Z は協定世界時 (UTC) を表します。他のタイムゾーンは、UTC との差異を +hh:mm または -hh:mm のフォーマットで表示しています。タイムゾーンの値が無い場合は、UTC ではない未知のタイムゾーンとして見なされます。

add-hours-to-time [altova:]

altova:add-hours-to-time(Time *as xs:time*, **Hours** *as xs:integer***)** を **xs:time** とする **XP3.1 XQ3.1**

日付までの期間を時間数で表示します。第 2 の引数は第 1 の引数として与えられた xs:time に追加される時間数です。結果は xs:time 型です。

サンプル

- **altova:add-hours-to-time**(xs:time("11:00:00"), 10) は 21:00:00 を返します。
- **altova:add-hours-to-time**(xs:time("11:00:00"), -7) は 04:00:00 を返します。

add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time *as xs:time*, **Minutes** *as xs:integer***)** を **xs:time** とする **XP3.1 XQ3.1**

日付までの期間を分数で表示します。第 2 の引数は第 1 の引数として与えられた xs:date に追加される分数です。結果は xs:date 型です。

サンプル

- **altova:add-minutes-to-time**(xs:time("14:10:00"), 45) 14:55:00 を返します。
- **altova:add-minutes-to-time**(xs:time("14:10:00"), -5) 14:05:00 を返します。

add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time *as xs:time*, **Minutes** *as xs:integer***)** を **xs:time** とする **XP3.1 XQ3.1**

時間までの期間を秒数で表示します。第 2 の引数は、第 1 の引数として与えられた xs:time に追加される秒数です。結果は xs:time 型です。第 2 のコンポーネントは 0 から 59.999 の範囲であることができます。

サンプル

- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20) は 14:00:20 を返します。
- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20.895) は 14:00:20.895 を返します。

[トップ]

日付/時刻データ型からタイムゾーンの部分を削除する XP3.1 XQ3.1

これらの関数は、現在の**xs:dateTime**、**xs:date**、または**xs:time** 値からそれぞれタイムゾーンを削除します。xs:dateTime と xs:dateTimeStamp の差異は、後者のタイムゾーンが必要な場合です。(前者の場合は任意です。) xs:dateTimeStamp 値のフォーマットはCCYY-MM-DDThh:mm:ss.sss±hh:mm またはCCYY-MM-DDThh:mm:ss.sssZ です。、日付と時刻が xs:dateTimeStamp としてシステムクロックから読み込まれる場合、 current-dateTime-no-TZ() 関数がタイムゾーンを削除するために使用されます。

current-date-no-TZ [altova:]

altova:current-date-no-TZ() を**xs:date** とする XP3.1 XQ3.1
この関数は引数を必要としません。current-date() (システムクロックによる現在の時刻) のタイムゾーンの部分を削除し、xs:date の値を返します。

サンプル

現在のdate が**2014-01-15+01:00** の場合:

- **altova:current-date-no-TZ**() は2014-01-15 を返します。

current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() を**xs:dateTime** とする XP3.1 XQ3.1
この関数は引数を必要としません。current-dateTime() (システムクロックによる現在の時刻) のタイムゾーンの部分を削除し、xs:dateTime の値を返します。

サンプル

現在のdateTime が**2014-01-15T14:00:00+01:00** の場合:

- **altova:current-dateTime-no-TZ**() は2014-01-15T14:00:00 を返します。

current-time-no-TZ [altova:]

altova:current-time-no-TZ() を**xs:time** とする XP3.1 XQ3.1
この関数は引数を必要としません。current-time() (システムクロックによる現在の時刻) のタイムゾーンの部分を削除し、xs:time の値を返します。

サンプル

現在のtime が**14:00:00+01:00** の場合:

- **altova:current-time-no-TZ**() は14:00:00 を返します。

date-no-TZ [altova:]

altova:date-no-TZ(InputDate *as xs:date*) を**xs:date** とする XP3.1 XQ3.1
この関数はxs:date 引数を必要とし、タイムゾーンの部分を削除し、xs:date の値を返します。日付が変更されていない点に注意してください。

サンプル

- **altova:date-no-TZ**(xs:date("2014-01-15+01:00")) は2014-01-15 を返します。

dateTime-no-TZ [altova:]

altova:dateTime-no-TZ(InputDateTime *as xs:dateTime*) を**xs:dateTime** とする XP3.1 XQ3.1

この関数は xs:dateTime 引数を必要とし、タイムゾーンの部分を削除し、xs:dateTime の値を返します。日付が変更されていない点に注意してください。

サンプル

- **altova:dateTime-no-TZ**(xs:date("2014-01-15T14:00:00+01:00")) は 2014-01-15T14:00:00 を返します。

time-no-TZ [altova:]

altova:time-no-TZ(InputTime *as xs:time*) を **xs:time** とする **XP3.1** **XQ3.1**

この関数は xs:time 引数を必要とし、タイムゾーンの部分を削除し、xs:time 値を返します。時刻は変更されていないことに注意してください。

サンプル

- **altova:time-no-TZ**(xs:time("14:00:00+01:00")) は 14:00:00 を返します。

[トップ]

期間から日数、時間数、分数、および、秒数を返す **XP3.1** **XQ3.1**

これらの関数は、期間から、月内の日数、時間数、分数、秒数をそれぞれ返します。

days-in-month [altova:]

altova:days-in-month(Year *as xs:integer*, **Month** *as xs:integer*) **asxs:integer** **XP3.1** **XQ3.1**

は 指定された月内の日数 を返します。Year と Month 引数を使用して月を指定することができます。

サンプル

- **altova:days-in-month**(2018, 10) は 31 を返します。
- **altova:days-in-month**(2018, 2) は 28 を返します。
- **altova:days-in-month**(2020, 2) は 29 を返します。

hours-from-dayTimeDuration-accumulated

altova:hours-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*) **asxs:integer** **XP3.1** **XQ3.1**

DayAndTime 引数 (which is of type xs:duration) により提供される期間内の時間の総数を返します。Day と Time コンポーネント内の時間は、整数である結果に追加されます。60 分として新規の一時間は考えられます。ネガティブな時間の値で期間の結果をナビゲートすることができます。

サンプル

- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5D") は 5 日間の時間の総計である 120 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H") は 5 日間の時間の総計プラス 2 時間である 122 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H60M") は 5 日間の時間の総計プラス 2 時間プラス 60 分である 123 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H119M") は 5 日間の時間の総計プラス 2 時間とプラス 119 分である 123 を返します。
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H120M") は 5 日間の時間の総計プラス 2 時間プラス 120 分である 124 を返します。

- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("-P5DT2H") は -122 を返します。

▼ minutes-from-dayTimeDuration-accumulated

altova:minutes-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*)
asxs:integer **XP3.1** **XQ3.1**
DayAndTime 引数 (which is of type xs:duration) により提出される期間内の分数の総数を返します Day と Time コンポーネントは、追加され、整数である結果に追加されます。ネガティブなマイナスの値で期間の結果をナビゲートします。

▣ *サンプル*

- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT60M") は を返します。60
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H") は一時間内の分数である 60 を返します。
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H40M") は 100 を返します。
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("P1D") は一日の秒数の総数である 1440 を返します。
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("-P1DT60M") は を返します。-1500

▼ seconds-from-dayTimeDuration-accumulated

altova:seconds-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*)
asxs:integer **XP3.1** **XQ3.1**
(xs:duration の型である)DayAndTime 引数 により提出された期間内の秒数の総数を返します。Day と Time コンポーネント内の秒数は、整数である結果に追加されます。ネガティブな秒の値で期間をナビゲートします。

▣ *サンプル*

- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1M") は一分内の秒数である 60 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H") は一時間内の秒数の総数である 3600 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H2M") は 3720 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("P1D") は一日の秒数の総数である 86400 を返します。
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("-P1DT1M") は -86460 を返します。

xs:dateTime または xs:dateReturn から週数を返す **XP3.1** **XQ3.1**

これらの関数は、xs:dateTime または xs:date から曜日を(整数として)を返します。曜日は(米国式フォーマットを使用して) 1 から 7 と番号づけられています。 日曜=1 と番号付けられます。欧州のフォーマットでは月曜 (=1) として番号付けられます。日曜=1 である米国フォーマットは整数 0 がフォーマットを表示するために使用できる箇所で設定することができます。

▼ weekday-from-dateTime [altova:]

altova:weekday-from-dateTime(DateTime *as xs:dateTime*) を **xs:integer** とする **XP3.1** **XQ3.1**

時刻付きの日付を単一の引数として、この日付の曜日を正数として返します。曜日は 日曜=1 から開始して番号を付けます。(月曜=1 とする) ヨーロッパのフォーマットが必要な場合、この関数の他の署名を使用します (*下の次の署名を参照*)。

サンプル

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"))` は月曜を表示する `2` を返します。

altova:weekday-from-dateTime(DateTime *as xs:dateTime*, **Format** *as xs:integer*) **asxs:integer** とする **XP3.1** **XQ3.1**

時間月の日付を最初の引数として、この日付の曜日を正数として返します。曜日は 月曜=1 から開始して番号を付けます。第 2 の引数(整数)が `0` の場合、日曜=1 から開始して、曜日は `1` から `7` と番号を付けます。第 2 の引数が `0` 以外の整数の場合、月曜=1 です。第 2 の引数がない場合、関数はこの関数の他の署名を持つと読み込まれます (*前の次の署名を参照*)。

サンプル

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 1)` は月曜を表示する `1` を返します。
- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 4)` は月曜を表示する `1` を返します。
- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 0)` は月曜を表示する `2` を返します。

▼ weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date*) を **xs:integer** とする **XP3.1** **XQ3.1**

日付を単一の引数として、この日付の曜日を正数として返します。曜日は 日曜=1 から開始して番号を付けます。(月曜=1 とする) ヨーロッパのフォーマットが必要な場合、この関数の他の署名を使用します (*下の次の署名を参照*)。

サンプル

- `altova:weekday-from-date(xs:date("2014-02-03+01:00"))` は月曜を表示する `2` を返します。

altova:weekday-from-date(Date *as xs:date*, **Format** *as xs:integer*) を **xs:integer** とする **XP3.1** **XQ3.1**

日付を最初の引数とし、この日付の曜日を正数として返します。曜日は 月曜=1 から開始して番号を付けます。第 2 (フォーマット) 引数が `0`, の場合、日曜=1 から開始し、曜日は `1` から `7` で番号付けられます。第 2 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。

[トップ]

xs:dateTime または xs:date から週数を返す **XP2** **XQ1** **XP3.1** **XQ3.1**

これらの関数は週数(整数として)を `xs:dateTime` から `xs:date` から返します。週の番号付けは、米国、欧州、イスラムのカレンダーフォーマットで使用することができます。週の始まりが異なるため、週数の番号付けは、カレンダーのフォーマットにより異なります。(米国フォーマットでは、日曜、欧州フォーマットでは月曜、イスラムフォーマットでは土曜が週の開始日です)。

▼ weeknumber-from-date [altova:]

altova:weeknumber-from-date(Date *as xs:date*, **Calendar** *as xs:integer*) を **xs:integer** とする
XP2 XQ1 XP3.1 XQ3.1
正数として提出された Date 引数の週数を返します。第 2 の引数(カレンダー)は続くカレンダーシステムを指定します。
サポートされる カレンダー の値は次のとおりです:

- 0 = US 米国のカレンダー(週の始まりは日曜日)
- 1 = ISO 標準、 欧州のカレンダー(週の始まりは月曜日)
- 2 = イスラムのカレンダー(週の始まりは土曜日)

デフォルトは 0 です。

サンプル

- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 0) は 13 を返します。
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 1) は 13 を返します。
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 2) は 13 を返します。
- **altova:weeknumber-from-date**(xs:date("2014-03-23")) は 13 を返します。

上のサンプル(2014-03-23) の date の曜日は日曜日です。米国およびイスラムのカレンダーは欧州カレンダーのこの日付よりも一週間先です。

weeknumber-from-dateTime [altova:]

altova:weeknumber-from-dateTime(DateTime *as xs:dateTime*, **Calendar** *as xs:integer*) を **xs:integer** とする XP2 XQ1 XP3.1 XQ3.1
正数として提出された DateTime 引数の週数を返します。第 2 の引数(カレンダー)は続くカレンダーシステムを指定します。
サポートされる カレンダー の値は次のとおりです:

- 0 = US 米国のカレンダー(週の始まりは日曜日)
- 1 = ISO 標準、 欧州のカレンダー(週の始まりは月曜日)
- 2 = イスラムのカレンダー(週の始まりは土曜日)

Default is 0.

サンプル

- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 0) は 13 を返します。
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 1) は 13 を返します。
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00"), 2) は 13 を返します。
- **altova:weeknumber-from-dateTime**(xs:dateTime("2014-03-23T00:00:00")) は 13 を返します。

上のサンプル(2014-03-23T00:00:00) の dateTime の曜日は日曜日です。米国およびイスラムのカレンダーは欧州カレンダーのこの日付よりも一週間先です。

[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 XP3.1 XQ3.1

関数は、xs:date, xs:time または xs:duration の構文コンポーネントを入力引数とし、引数を結合させて対応するデータ型を構築します。

build-date [altova:]

altova:build-date(Year *as xs:integer*, **Month** *as xs:integer*, **Date** *as xs:integer***)** を **xs:date** とする XP3.1 XQ3.1

第 1 、第 2 、第3 の引数は、それぞれ、年、月、日を表します。xs:date 型の値を構築するため結合されます。整数の値は、特定の日付の一部の適正な範囲内である必要があります。例えば第 2 の引数（月の部分）は 12 以上であってはなりません。

サンプル

- **altova:build-date**(2014, 2, 03) は 2014-02-03 を返します。

build-time [altova:]

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer***)** を **xs:time** とする XP3.1 XQ3.1

第 1 、第 2 、第3 引数はそれぞれ時間数（0 から 23）、分数（0 から 59）、および秒数（0 から 59）の値です。これらの値は、xs:time 型の値を作成するために結合されます。整数の値は、それぞれの部分の正しい範囲内である必要があります。例えば、秒（分）引数は、59 以上であってはなりません。値にタイムゾーンを追加するには、この関数の他の署名を使用してください（*次の署名を参照*）。

サンプル

- **altova:build-time**(23, 4, 57) は 23:04:57 を返します。

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*, **TimeZone** *as xs:string***)** を **xs:time** とする XP3.1 XQ3.1

第 1 、第 2 、第 3 引数はそれぞれ時間数（0 から 23）、分数（0 から 59）、および秒数（0 から 59）の値です。第四の引数は、値の一部としてタイムゾーンを与えます。4 つの引数が結合され、xs:time 型の値を構築します。例えば、第 2 の引数（分数）は 59 以上であってはなりません。

サンプル

- **altova:build-time**(23, 4, 57, '+1') は 23:04:57+01:00 を返します。

build-duration [altova:]

altova:build-duration(Years *as xs:integer*, **Months** *as xs:integer***)** を **xs:yearMonthDuration** とする XP3.1 XQ3.1

xs:yearMonthDuration 型の値を構築するためには 2 つの引数が必要です。最初の引数は期間の 年数 値の部分を与え、第 2 の引数は、月数 値の部分を与えます。第 2 の引数（月数）が 12 と同じまたはより大きくなると、整数は 12 により分割されます。商は、年数 の部分を表す 最初の引数に加算され、（除算の）残りは の部分は 月数の部分を表すために使用されます。期間の xs:dayTimeDuration 型を作成するには、次の署名を参照してください。

サンプル

- **altova:build-duration**(2, 10) は P2Y10M を返します。
- **altova:build-duration**(14, 27) は P16Y3M を返します。
- **altova:build-duration**(2, 24) は P4Y を返します。

altova:build-duration(Days *as xs:integer*, **Hours** *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer***)** を **xs:dayTimeDuration** とする XP3.1 XQ3.1

xs:dayTimeDuration 型の値を構築するためには4 つの引数が必要です。最初の引数は期間の日数 値の部分で、第 2 、第 3 、第 4 引数は、それぞれ期間の時間数、分数、秒数 値のです。これらの3 つの 時間 引数は、次の高い単位の値に加算され、結果は期間の全体の値を計算するために使用されます。例えば、 72 秒は、 1M+12S (1 分 と12 秒)に変換され、この値が期間全体の値を計算するために使用されます。 xs:yearMonthDuration 型の期間を構築するためには、次の署名を参照してください。

サンプル

- **altova:build-duration**(2, 10, 3, 56) は P2DT10H3M56S を返します。
- **altova:build-duration**(1, 0, 100, 0) は P1DT1H40M を返します。
- **altova:build-duration**(1, 0, 0, 3600) は P1DT1H を返します。

[トップ]

文字列入力から 日付、日付時刻 または 時刻 を構築する XP2 XQ1 XP3.1 XQ3.1

関数は、文字列を引数として、 xs:date、xs:dateTime または xs:time データ型を構築します。文字列は、データ型のコンポーネントを提出されたパターン引数をベースにして分析されます。

parse-date [altova:]

altova:parse-date(Date *as xs:string*, **DatePattern** *as xs:string*) を **xs:date** とする XP2 XQ1 XP3.1 XQ3.1

Date 入力文字列を **xs:date** の値として返します。第二の引数である **DatePattern** は、入力文字列のパターン(コンポーネントのシーケンス) を指定します。**DatePattern** は、下にリストされるコンポーネント指定子および任意の文字であるコンポーネント セパレータ と共に説明されています。下のサンプルを参照してください。

D	日付
M	月
Y	年

DatePattern のパターンは、**Date** のパターンに一致する必要があります。出力は **xs:date** 型であるため、出力は常に **YYYY-MM-DD** 構文フォーマットになります。

サンプル

- **altova:parse-date**(xs:string("06-03-2014"), "[D]-[M]-[Y]") は 2014-03-06 を返します。
- **altova:parse-date**(xs:string("06-03-2014"), "[M]-[D]-[Y]") は 2014-03-06 を返します。
- **altova:parse-date**("06/03/2014", "[M]/[D]/[Y]") は 2014-03-06 を返します。
- **altova:parse-date**("06 03 2014", "[M] [D] [Y]") は 2014-03-06 を返します。
- **altova:parse-date**("6 3 2014", "[M] [D] [Y]") は 2014-03-06 を返します。

parse-dateTime [altova:]

altova:parse-dateTime(DateTime *as xs:string*, **DateTimePattern** *as xs:string*) を **xs:dateTime** とする XP2 XQ1 XP3.1 XQ3.1

DateTime 入力文字列を **xs:dateTime** の値として返します。第二の引数である **DateTimePattern** は、入力文字列のパターン(コンポーネントのシーケンス) を指定します。**DateTimePattern** は、下にリストされるコンポーネント指定子および任意の文字であるコンポーネント セパレータ と共に説明されています。下のサンプルを参照してください。

D	日
M	月
Y	年

H	時間
m	分
s	秒

DateTimePattern のパターンは、DateTime のパターンに一致する必要があります。出力は xs:dateTime 型であるため、出力は常に YYYY-MM-DDTHH:mm:ss 構文フォーマットになります。

サンプル

- altova:parse-dateTime(xs:string("06-03-2014 13:56:24"), "[D]-[M]-[Y] [H]:[m]:[s]") は 2014-03-06T13:56:24 を返します。
- altova:parse-dateTime("time=13:56:24; date=06-03-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]") は 2014-03-06T13:56:24 を返します。

parse-time [altova:]

altova:parse-time(Time *as xs:string*, TimePattern *as xs:string*) を xs:time とする XP2 XQ1 XP3.1 XQ3.1

Time 入力文字列を xs:time の値として返します。第二の引数である TimePattern は、入力文字列のパターン（コンポーネントのシーケンス）を指定します。TimePattern は、下にリストされるコンポーネント指定子および任意の文字であるコンポーネント セパレータと共に説明されています。下のサンプルを参照してください。

H	時間
m	分
s	秒

TimePattern のパターンは、Time のパターンに一致する必要があります。出力は xs:time 型であるため、出力は常に YYYY-HH:mm:ss 構文フォーマットになります。

サンプル

- altova:parse-time(xs:string("13:56:24"), "[H]:[m]:[s]") は 13:56:24 を返します。
- altova:parse-time("13-56-24", "[H]-[m]") は 13:56:00 を返します。
- altova:parse-time("time=13h56m24s", "time=[H]h[m]m[s]s") は 13:56:24 を返します。
- altova:parse-time("time=24s56m13h", "time=[s]s[m]m[H]h") は 13:56:24 を返します。

[トップ]

年齢に関連した関数 XP3.1 XQ3.1

関数は計算された年齢(i) 入力引数の日付と現在の日付の期間 (ii) 2 つの入力引数の日付の期間 を返します。altova:age 関数は、年齢を年数で返します、altova:age-details は年齢を年数、月数、日数からなる3 つの整数のシーケンスで返します。

age [altova:]

altova:age(StartDate *as xs:date*) を xs:integer とする XP3.1 XQ3.1

引数として提出された開始日から（システムクロックから取得された）現在の日付 までの日数を数えて、あるオブジェクトの年齢の年数である正数を返します。入力引数が 1 年よりも大きい場合、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 **XP3.1** **XQ3.1**
最初の引数として提出された開始日から第 2 の引数の終了日 までの日数を数えて、あるオブジェクトの年齢の年数である正数を返します。最初の引数が1 年または第 2 の引数より先後の日付の場合、返される値は負の数です。

サンプル

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2000-01-15"), xs:date("2010-01-15")) は10 を返します。
- **altova:age**(xs:date("2000-01-15"), current-date()) は、現在の日付が2014-01-15 の場合14 を返します。
- **altova:age**(xs:date("2014-01-15"), xs:date("2010-01-15")) は-4 を返します。

age-details [altova:]

altova:age-details(InputDate *as xs:date*) を(**xs:integer**)* とする **XP3.1** **XQ3.1**
引数とシステムクロックから取得された現在の日付として提出された日付の間の年数、月数、日数の3 つの整数を返します。返されたyears+months+days の合計は、2 つの日付（入力の日付と現在の日付）の時間差です。 入力の日付は現在の日付より先早いまたはお遅い値をもつことができますが、入力の日付が早いか遅いかは返される値で示されていません。戻される値は常に正の数です。

サンプル

現在の日付が**2014-01-15** の場合:

- **altova:age-details**(xs:date("2014-01-16")) は(0 0 1)を返します。
- **altova:age-details**(xs:date("2014-01-14")) は(0 0 1) を返します。
- **altova:age-details**(xs:date("2013-01-16")) は(1 0 1) を返します。
- **altova:age-details**(current-date()) は(0 0 0)を返します。

altova:age-details(Date-1 *as xs:date*, **Date-2** *as xs:date*) を(**xs:integer**)* とする **XP3.1** **XQ3.1**
二つの引数間の年数、月数、日数の3 つの整数を返します。 返されたyears+months+days の合計は、2 つの入力の日付の時間差です。最初の引数として提出される二つの日付はどちらが速くても、また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。

[トップ]

11.1.2.1.3 XPath/XQuery 関数: 位置情報

以下の位置情報 XPath/XQuery 拡張関数は、MapForce の現在のバージョンによりサポートされています。また、次で使用することができます: (i) XSLT コンテキスト内の XPath 式、または (ii) XQuery ドキュメント内の XQuery 式。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は **Altova 拡張関数名前空間、`http://www.altova.com/xslt-extensions`** に収められており、**`altova:`** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	**XP1 XP2 XP3.1**
XSLT 関数 (XSLT 内の XPath 式で使用):	**XSLT1 XSLT2 XSLT3**
XQuery 関数 (XQuery 内の XQuery 式で使用):	**XQ1 XQ3.1**

▼ format-geolocation [altova:]

`altova:format-geolocation(Latitude` *`as xs:decimal`*`, `**`Longitude`** *`as xs:decimal`*`, `**`GeolocationOutputStringFormat`** *`as xs:integer`*`)` を **`xs:string`** とする **XP3.1 XQ3.1**
緯度と経度を最初の 2 つの引数とし、位置情報を文字列として出力します。第 3 の引数 **`GeolocationOutputStringFormat`** は、位置情報出力文字列のフォーマットです。出力文字列のフォーマットを識別するために、正数の値 `1` から `4` を使用します。(*下の「位置情報出力文字列フォーマット」を参照してください*)。緯度の値の範囲は `+90` から `-90` (北 から 南) です。経度の値の範囲は `+180` から `-180` (東 から 西) です。

メモ `image-exif-data` 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:format-geolocation`**`(33.33, -22.22, 4)` は `xs:string "33.33 -22.22"` を返します。
- **`altova:format-geolocation`**`(33.33, -22.22, 2)` は `xs:string "33.33N 22.22W"` を返します。
- **`altova:format-geolocation`**`(-33.33, 22.22, 2)` は `xs:string "33.33S 22.22E"` を返します。
- **`altova:format-geolocation`**`(33.33, -22.22, 1)` は `xs:string "33°19'48.00"S 22° 13'12.00"E"` を返します。

▣ *位置情報出力文字列フォーマット:*

与えられた緯度と経度は下にリストされる出力フォーマットによりフォーマットされます。希望するフォーマットは整数 ID (`1` から `4`) により識別されます。緯度の値の範囲は `+90` から `-90` (北 から 南) です。経度の値の範囲は `+180` から `-180` (東 から 西) です。

1
度、分、10 進の秒、方角のサフィックス付き (`N/S, E/W`) **`D°M'S.SS"N/S  D°M'S.SS"E/W`**

サンプル: `33°55'11.11"N  22°44'66.66"W`

2

10 進の度、方角のサフィックス付き (`N/S`, `E/W`)
`D.DDN/S  D.DDE/W`
サンプル: `33.33N  22.22W`

3

度、分、10 進の秒、プレフィックスサイン付き (`+/-`); (`N/E`) のためのプラスサインは任意です。
`+/-D°M'S.SS"  +/-D°M'S.SS"`
サンプル: `33°55'11.11"  -22°44'66.66"`

4

10 進の度、プレフィックスサイン付き (`+/-`); (`N/E`) のためのプラスサインは任意です。
`+/-D.DD  +/-D.DD`
サンプル: `33.33 -22.22`

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 `Geolocation`を標準 Exif メタデータ タグから生成します。`Geolocation` は、4 つの Exif タグの連結です: 単位の追加された (*下のテーブル参照*) `GPSLatitude`、`GPSLatitudeRef`、`GPSLongitude`、`GPSLongitudeRef`。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

parse-geolocation [altova:]

`altova:parse-geolocation(GeolocationInputString` *`as xs:string`*`)` を`xs:decimal+` とする **XP3.1 XQ3.1**

`GeolocationInputString` 引数を解析して、位置情報の緯度と経度 (この通りの順番) を 2 つの`xs:decimal` アイテムのシーケンスとして返します。位置情報入力文字列が提供されることのできるフォーマットは以下のリストの通りです。

メモ `image-exif-data` 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 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い(") でハイライトされており、エスケープした単位インジケーターは青い("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き (+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151°

				13'11.73"E

▼ geolocation-distance-km [altova:]

altova:geolocation-distance-km(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) を **xs:decimal** とする **XP3.1** **XQ3.1**
2 つの位置情報の間の距離をキロメートルで計算します。位置情報入力文字列を提供することのできるフォーマットは下にリストされています。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西) です。

メモ image-exif-data 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 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルとしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い (") でハイライトされており、エスケープした単位インジケーターは青い ("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き (+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD

サンプル: `33.33 -22.22`

フォーマットの組み合わせのサンプル:
```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

- *Altova Exif 属性: 位置情報*

  Altova XPath/XQuery エンジンはカスタム属性 `Geolocation`を標準 Exif メタデータ タグから生成します。`Geolocation` は、4 つの Exif タグの連結です: 単位の追加された(*下のテーブル参照*) `GPSLatitude`、`GPSLatitudeRef`、`GPSLongitude`、GPSLongitudeRef 。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) を **xs:decimal** とする **XP3.1** **XQ3.1**
2 つの位置情報の間の距離をマイルで計算します。位置情報入力文字列を提供することのできるフォーマットは下にリストされています。緯度の値の範囲は `+90` から `-90` (北 から 南) です。経度の値の範囲は `+180` から `-180` (東 から 西)です。

メモ `image-exif-data` 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` (`N` から `S`)。経度の値の範囲は `+180` から `-180` (`E` から `W`)。

  メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルにしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い(`"`) でハイライトされており、エスケープした単位インジケーターは青い(`""`) でハイライトされています。

  - 度、分、10 進の秒、方角のサフィックス付き(`N/S, E/W`)
    D°M'S.SS"N/S D°M'S.SS"W/E
    サンプル: `33°55'11.11"N 22°44'55.25"W`

  - 度、分、10 進の秒、プレフィックスサイン付き(`+/-`); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
    サンプル: `33°55'11.11" -22°44'55.25"`

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き(+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き(N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き(+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocations-bounding-rectangle [altova:]

altova:geolocations-bounding-rectangle(Geolocations を *xs:sequence* として GeolocationOutputStringFormat を *xs:integer* として) を**xs:string** として **XP3.1** **XQ3.1**
最初の引数として文字列のシーケンスを取ります。シーケンスの各文字列が位置情報です。関数はそれぞれが、最初の引数に送信されたすべての位置情報を含むために最適にサイズ調整された長方形の左上と右下の位置情報の座標である2 つの文字列のシーケンスを返します。位置情報入力文字列が表示される書式は以下にリストされています(*「位置情報入力文字列書式」を参照してください*)。緯度の値は+90 から-90 への(N から S への) の範囲です。経度の値は+180 から-180 への(E から W への) の範囲です。

関数の2 番目の引数は出力シーケンス内の2 つの位置情報文字列の書式を指定します。引数はそれぞれの値が異なる位置情報文字列書式を識別する 1 から 4 の整数の値を取ります(*下記の「位置情報出力文字列書式」を参照してください*)。

メモ image-exif-data 関数とExif メタデータの属性は入力文字列を提供するために使用することができます。

サンプル

- **altova:geolocations-bounding-rectangle**(("48.2143531 16.3707266", "51.50939 -

0.11832"), 1) はシーケンス ("51°30'33.804"N 0°7'5.952"W", "48°12'51.67116"N 16° 22'14.61576"E") を返します。

- **altova:geolocations-bounding-rectangle**(("48.2143531 16.3707266", "51.50939 -0.11832", "42.5584577 -70.8893334"), 4) はシーケンス ("51.50939 -70.8893334", "42.5584577 16.3707266") を返します

位置情報入力文字列フォーマット:

位置情報入力文字列は空白で区別された緯度と経度（この通りの順番）を含む必要があります。緯度と経度は以下のフォーマットをとることができます。組み合わせることも可能です。緯度が1つのフォーマットで、経度が他のフォーマットをとることができます。緯度の値の範囲は +90 から -90 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルにしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い (") でハイライトされており、エスケープした単位インジケーターは青い ("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  サンプル: **33°55.55'N 22°44.44'W**

- 度、分、10 進の分、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **+33°55.55' -22°44.44'**

- 10 進の度、方角のサフィックス付き (N/S, E/W)
  D.DDN/S D.DDW/E
  サンプル: **33.33N 22.22W**

- 10 進の度、プレフィックスサイン付き (+/-); (**N/S E/W**)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **33.33 -22.22**

フォーマットの組み合わせのサンプル:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

位置情報出力文字列フォーマット:

与えられた緯度と経度は下にリストされる出力フォーマットによりフォーマットされます。希望するフォーマットは整数 ID (1 から 4) により識別されます。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西) です。

1
度、分、10 進の秒、方角のサフィックス付き (N/S, E/W) **D°M'S.SS"N/S D°M'S.SS"E/W** *サンプル:* **33°55'11.11"N 22°44'66.66"W**

2
10 進の度、方角のサフィックス付き (N/S, E/W) **D.DDN/S D.DDE/W** *サンプル:* **33.33N 22.22W**

3
度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。 **+/-D°M'S.SS" +/-D°M'S.SS"** *サンプル:* **33°55'11.11" -22°44'66.66"**

4
10 進の度、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。 **+/-D.DD +/-D.DD** *サンプル:* **33.33 -22.22**

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-within-polygon [altova:]

altova:geolocation-within-polygon(Geolocation *as xs:string*, **((PolygonPoint** *as xs:string*)+)) を **xs:boolean** とする **XP3.1 XQ3.1**

PolygonPoint 引数により説明されている **Geolocation** (最初の引数) が多角形のエリア内に存在するかを決定します。もし、PolygonPoint 引数が最初と最後のポイントが同じ場合に作成される)閉じられたフィギュアを作成しない場合、フィギュアを閉じるために、最初のポイントが明示的に最後のポイントとして追加されます。全ての引数(Geolocation および PolygonPoint+) は、(*下にリストされるフォーマットの* 位置情報入力文字列により提供されます。Geolocation 引数が多角形エリア内にある場合、関数は true(); を返します。その他の場合は false() を返します。緯度の値の範囲は +90 から -90 (北 から 南) です。経度の値の範囲は +180 から -180 (東 から 西)です。

メモ image-exif-data 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` (`N` から `S`)。経度の値の範囲は `+180` から `-180` (`E` から `W`)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルにしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い(`"`) でハイライトされており、エスケープした単位インジケーターは青い(`""`) でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (`N/S`, `E/W`)
  `D°M'S.SS"N/S D°M'S.SS"W/E`
  サンプル: **`33°55'11.11"N 22°44'55.25"W`**

- 度、分、10 進の秒、プレフィックスサイン付き (`+/-`); (**`N/E`**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **`33°55'11.11" -22°44'55.25"`**

- 度、分、10 進の分、方角のサフィックス付き (`N/S`, `E/W`)
  `D°M.MM'N/S D°M.MM'W/E`
  サンプル: **`33°55.55'N 22°44.44'W`**

- 度、分、10 進の分、プレフィックスサイン付き (`+/-`); (**`N/E`**) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: **`+33°55.55' -22°44.44'`**

- 10 進の度、方角のサフィックス付き (`N/S`, `E/W`)
  `D.DDN/S D.DDW/E`
  サンプル: **`33.33N 22.22W`**

- 10 進の度、プレフィックスサイン付き (`+/-`); (**`N/S E/W`**) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: **`33.33 -22.22`**

フォーマットの組み合わせのサンプル:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 **`Geolocation`** を標準 Exif メタデータタグから生成します。**`Geolocation`** は、4 つの Exif タグの連結です: 単位の追加された (*下のテーブル参照*) `GPSLatitude`、

GPSLatitudeRef、GPSLongitude、GPSLongitudeRef。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-within-rectangle [altova:]

altova:geolocation-within-rectangle(Geolocation *as xs:string*, **RectCorner-1** *as xs:string*, **RectCorner-2** *as xs:string*) を **xs:boolean** とする **XP3.1** **XQ3.1**
長方形の対格の角を指定する第 2 及び第 3 引数(**RectCorner-1**、**RectCorner-2**)により規明されている **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 (N から S)。経度の値の範囲は +180 から -180 (E から W)。

メモ 単一およびダブル引用符が入力文字列引数を区切るために使用されていると、使用されている単一およびダブル引用が、それぞれ、分の値と秒の値、不一致をもたらします。この様な場合、分の値と秒の値を表すための使用されている引用符は、ダブルにしてエスケープされる必要があります。このセクションのサンプルでは、入力文字列を区別するために使用されている引用符は黄色い (") でハイライトされており、エスケープした単位インジケーターは青い ("") でハイライトされています。

- 度、分、10 進の秒、方角のサフィックス付き (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  サンプル: **33°55'11.11"N 22°44'55.25"W**

- 度、分、10 進の秒、プレフィックスサイン付き (+/-); (**N/E**) のためのプラスサインは任意です。+/-D°M'S.SS" +/-D°M'S.SS"
  サンプル: **33°55'11.11" -22°44'55.25"**

- 度、分、10 進の分、方角のサフィックス付き (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E

サンプル: `33°55.55'N  22°44.44'W`

- 度、分、10 進の分、プレフィックスサイン付き(+/-); (N/E) のためのプラスサインは任意です。+/-D°M.MM' +/-D°M.MM'
  サンプル: `+33°55.55'  -22°44.44'`

- 10 進の度、方角のサフィックス付き(N/S, E/W)
  `D.DDN/S  D.DDW/E`
  サンプル: `33.33N  22.22W`

- 10 進の度、プレフィックスサイン付き(+/-); (N/S E/W)) のためのプラスサインは任意です。+/-D.DD +/-D.DD
  サンプル: `33.33  -22.22`

フォーマットの組み合わせのサンプル:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif 属性: 位置情報

Altova XPath/XQuery エンジンはカスタム属性 `Geolocation`を標準 Exif メタデータ タグから生成します。`Geolocation` は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) GPSLatitude、GPSLatitudeRef、GPSLongitude、GPSLongitudeRef 。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

[トップ]

11.1.2.1.4 XPath/XQuery 関数: イメージに関連

以下のイメージに関連したXPath/XQuery 拡張関数は、MapForce の現在のバージョンによりサポートされています。また、次で使用することができます: (i) XSLT コンテキスト内のXPath 式、または(ii) XQuery ドキュメント内のXQuery 式。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova 拡張関数名前空間、`http://www.altova.com/xslt-extensions` に収められており、`altova:` プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3

XQuery 関数 (XQuery 内の XQuery 式で使用):	XQ1 XQ3.1

▼ suggested-image-file-extension [altova:]

altova:suggested-image-file-extension(Base64String *as string*) を **string?** とする **XP3.1 XQ3.1**
イメージファイルの Base64 エンコード を引数として、イメージのファイル拡張子を、イメージの Base64 エンコード内の記録として返します。返された値は、エンコード内で使用することのできるイメージ型情報を基にしたヒント です。この情報が使用できない場合、空の文字列が返されます。この関数は、Base64 イメージをファイルとして保存し、適切なファイル拡張子を動的取得する際に役に立ちます。

サンプル

- **altova:suggested-image-file-extension**(/MyImages/MobilePhone/Image20141130.01) は 'jpg' を返します。
- **altova:suggested-image-file-extension**($XML1/Staff/Person/@photo) は '' を返します。

上のサンプルでは、関数の引数として与えられたノードは Base64 エンコードイメージを含むと仮定します。最初のサンプルは jpg をファイルの型および拡張子として取得します。二番目のサンプルでは、与えられた Base64 エンコードは使用できる拡張子の情報を提供しません。

▼ mt-transform-image [altova:]

altova:mt-transform-image(Base64Image *as Base64BinaryString*, **Size** *as item()+*, **Rotation** *as xs:integer*, **Quality** *as xs:integer*) を **Base64BinaryString** とする **XP3.1 XQ3.1**
Base64-エンコード イメージ を最初の引数として、変換された Base64-エンコード イメージを返します。第 2、第 3、第 4 引数は変換された以下のイメージ パラメーターです: サイズ、回転、および クオリティ。

- サイズ引数には 3 つのサイズ変更のオプションがあります。

(X,Y)	絶対ピクセルの値。アスペクト 率は保持されません。高さと幅は自動的にイメージの長い および短い 辺に逢わされるため、高さと幅の指示は関係ありません。2 つの整数アイテムのシーケンスとして値が入力されます。かっこ が必要です。
X	X をピクセルの新しい長い 辺として、イメージの縦横比のサイズ変更が行われます。アスペクト 率は保持されます。値は整数で引用符なしで入力されます。
'X%'	元のディメンションの与えられた パーセンテージにイメージのサイズを変更します。値は文字列として引用符を付けて入力される必要があります。

- 回転は以下の値を持つことができます: 90、180、270、-90、-180、-270。これらの値は回転の度数です。正の値はイメージを時計回りに回転させます。負の値はイメージを反時計回りに回転させます。 Altova Exif 属性 **OrientationDegree** を使用して、イメージの現在の回転の度数(0, 90, 180, 270) をイメージの Exif **Orientation** タグから取得することができます。ですが、**OrientationDegree** 属性はデータの **Orientation** タグから取得されるため、Exif データ内に **Orientation** タグが存在する場合のみ使用することができます。(*下の OrientationDegree 説明を参照してください*)。
- クオリティは、0 から 100 の値で、JPEG 圧縮の IJG クオリティスケールの値を参照していますが、クオリティの パーセンテージのインジケーターではありません。サイズとクオリティのどちらかを優先すると、もう一方の優先度が下がります。フルカラーソースのイメージでは、75 が通常最高値と見なされています。もし、75 が満足のいく結果をもたらさない場合は、値を上げてください。

メモ Exif データ が元のイメージに存在する場合、変換時に削除され、変換されたイメージには Exif データは存在しません。

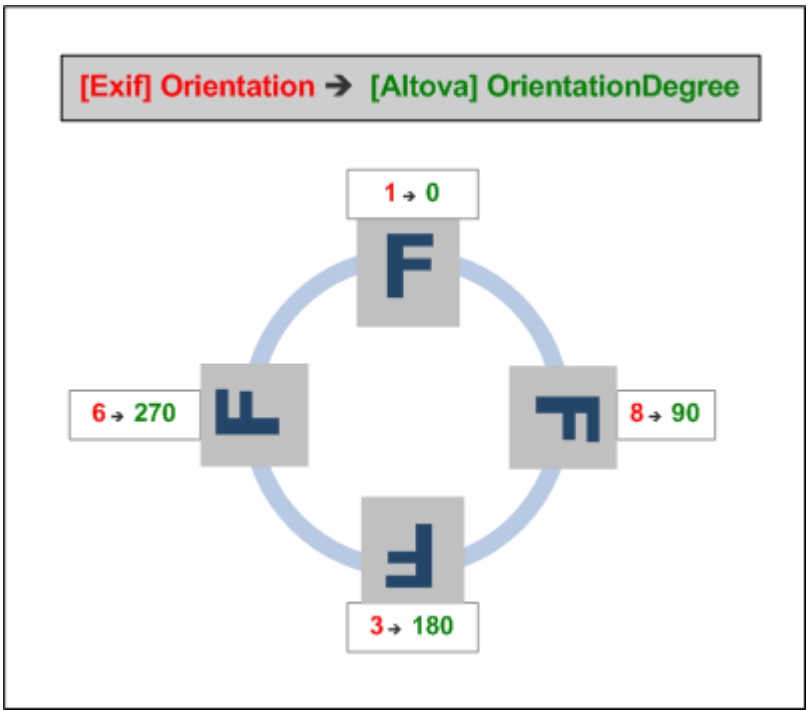
サンプル

- **mt-transform-image**(Images/Image[@id='43'], '50%', 90, 75)
  関数は、43 の@id 値を持つイメージ子孫ノード内でBase64-エンコード 文字列として保管されているイメージを入力とします。関数は変換されたイメージを返します。変換されたイメージは50% まで、サイズ変更され、時計回りに90 度回転され、75 のクオリティレベルを与えられます。
- **mt-transform-image**(Images/Image[@id='43'], 400, 90, 75)
  関数は前のサンプルと同じ結果を出しますが、長い辺は400 ピクセルの特定の値に設定されています。元のイメージのアスペクト率は保持されます。
- **mt-transform-image**(Images/Image[@id='43'], (400, 280), image-exif-data($XML1/$XML1/Images/ReferenceImage)/@OrientationDegree, 75)
  このサンプルは、前のサンプルと同じイメージを選択し、同じクオリティ値(75) を設定します。イメージのサイズは400x280 ピクセルに設定されています。回転値は、ノード内のBase64-エンコード イメージの**@OrientationDegree** 属性から得られます。

Altova Exif 属性: OrientationDegree

Altova XPath/XQuery エンジンはカスタム属性 **OrientationDegree** をExif メタデータタグ **Orientation** から生成します。

OrientationDegree は標準Exif タグ Orientation を正数の値(1、8、3 または6) からそれぞれ対応する度数の値(0, 90, 180, 270) へ下の図に示されているように変換します。2、4、5、7 のOrientation 値の変換はないことに注意してください。(これらの向きはイメージ1 を垂直方向中央軸で反転して、2 の値を持つイメージを取得します。そして、このイメージを90-度ごとに時計回りにジャンプさせそれぞれ7、4、および5 の値を取得します。)。



標準 Exif メタタグ

- ImageWidth

- ImageLength
- BitsPerSample
- Compression
- PhotometricInterpretation
- Orientation
- SamplesPerPixel
- PlanarConfiguration
- YCbCrSubSampling
- YCbCrPositioning
- XResolution
- YResolution
- ResolutionUnit
- StripOffsets
- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction
- WhitePoint
- PrimaryChromaticities
- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
- Artist
- Copyright

- ExifVersion
- FlashpixVersion
- ColorSpace
- ComponentsConfiguration
- CompressedBitsPerPixel
- PixelXDimension
- PixelYDimension
- MakerNote
- UserComment
- RelatedSoundFile
- DateTimeOriginal
- DateTimeDigitized
- SubSecTime
- SubSecTimeOriginal
- SubSecTimeDigitized
- ExposureTime
- FNumber
- ExposureProgram
- SpectralSensitivity
- ISOSpeedRatings
- OECF
- ShutterSpeedValue
- ApertureValue

- BrightnessValue
- ExposureBiasValue
- MaxApertureValue
- SubjectDistance
- MeteringMode
- LightSource
- Flash
- FocalLength
- SubjectArea
- FlashEnergy
- SpatialFrequencyResponse
- FocalPlaneXResolution
- FocalPlaneYResolution
- FocalPlaneResolutionUnit
- SubjectLocation
- ExposureIndex
- SensingMethod
- FileSource
- SceneType
- CFAPattern
- CustomRendered
- ExposureMode
- WhiteBalance
- DigitalZoomRatio
- FocalLengthIn35mmFilm
- SceneCaptureType
- GainControl
- Contrast
- Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

- GPSVersionID
- GPSLatitudeRef
- GPSLatitude
- GPSLongitudeRef
- GPSLongitude
- GPSAltitudeRef
- GPSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSStatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum

- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation
- GPSDateStamp
- GPSDifferential

image-exif-data [altova:]

altova:image-exif-data(Base64BinaryString *as string*) **をelement? とする** **XP3.1** **XQ3.1**
Base64 エンコード JPEG イメージを引数として、イメージのExif メタデータを含む**Exif** という名の要素を返します。Exif メタデータは`Exif` 要素の属性の値ペアとして作成されます。属性名は、Base64エンコード内で検出されたExif データタグです。Exif 仕様タグのリストは以下の通りです。ベンダー特有のタグがExif データ内に存在する場合、タグとその値も属性値のペアとして返されます。標準 Exif メタデータタグに追加して(*下のリスト参照*) Altova 特有の属性値のペアもまた生成されます。これらのAltova Exif 属性は以下のとおりです。

サンプル

- 属性にアクセスするには、以下の関数を使用します:
  image-exif-data(//MyImages/Image20141130.01)**/@GPSLatitude**
  image-exif-data(//MyImages/Image20141130.01)**/@Geolocation**
- 全ての属性にアクセスするには、以下の関数を使用します:
  image-exif-data(//MyImages/Image20141130.01)**/@***
- 全ての属性の名前にアクセスするには、以下の式を使用します:
  for $i in image-exif-data**(//MyImages/Image20141130.01)**/@* return name($i)
  関数により返される属性の名前を検出するために役に立ちます。

Altova Exif 属性: 位置情報

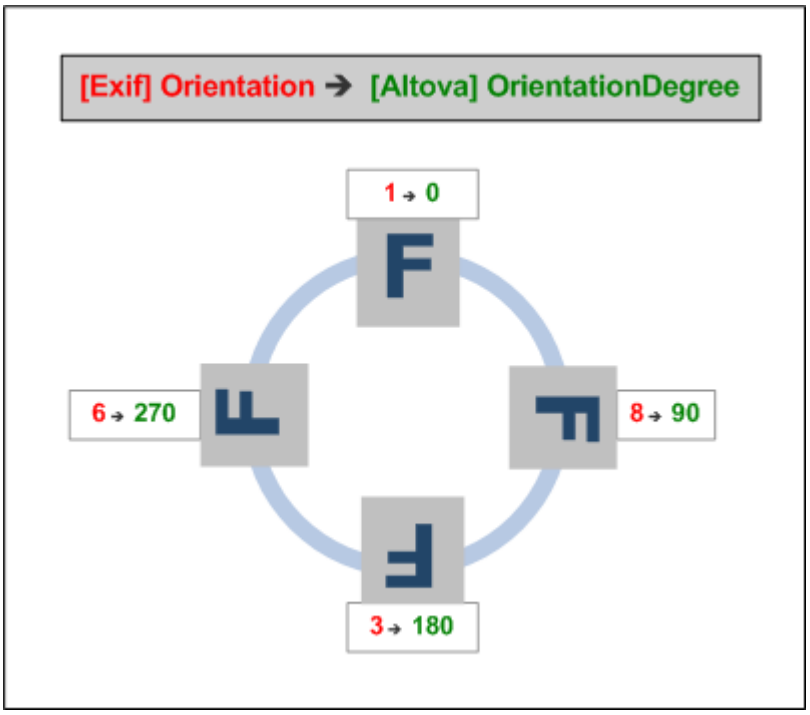
Altova XPath/XQuery エンジンはカスタム属性 **Geolocation**を標準 Exif メタデータ タグから生成します。**Geolocation** は、4 つのExif タグの連結です: 単位の追加された(*下のテーブル参照*) `GPSLatitude`、`GPSLatitudeRef`、`GPSLongitude`、`GPSLongitudeRef`。

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

Altova Exif 属性: OrientationDegree

Altova XPath/XQuery エンジンはカスタム属性 **OrientationDegree** をExif メタデータタグ **Orientation** から生成します。

OrientationDegree は標準 Exif タグ Orientation を正数の値（1、8、3 または 6）からそれぞれ対応する度数の値（0, 90, 180, 270）へ下の図に示されているように変換します。2、4、5、7 の Orientation 値の変換はないことに注意してください。（これらの向きはイメージ 1 を垂直方向中央軸で反転して、2 の値を持つイメージを取得します。そして、このイメージを 90-度ごとに時計回りにジャンプさせそれぞれ 7、4、および 5 の値を取得します。）。



標準 Exif メタデータ タグのリスト

- ImageWidth
- ImageLength
- BitsPerSample
- Compression
- PhotometricInterpretation
- Orientation
- SamplesPerPixel
- PlanarConfiguration
- YCbCrSubSampling
- YCbCrPositioning
- XResolution
- YResolution
- ResolutionUnit
- StripOffsets
- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction
- WhitePoint
- PrimaryChromaticities

- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
- Artist
- Copyright

- ExifVersion
- FlashpixVersion
- ColorSpace
- ComponentsConfiguration
- CompressedBitsPerPixel
- PixelXDimension
- PixelYDimension
- MakerNote
- UserComment
- RelatedSoundFile
- DateTimeOriginal
- DateTimeDigitized
- SubSecTime
- SubSecTimeOriginal
- SubSecTimeDigitized
- ExposureTime
- FNumber
- ExposureProgram
- SpectralSensitivity
- ISOSpeedRatings
- OECF
- ShutterSpeedValue
- ApertureValue
- BrightnessValue
- ExposureBiasValue
- MaxApertureValue
- SubjectDistance
- MeteringMode
- LightSource
- Flash
- FocalLength
- SubjectArea
- FlashEnergy
- SpatialFrequencyResponse
- FocalPlaneXResolution
- FocalPlaneYResolution
- FocalPlaneResolutionUnit
- SubjectLocation
- ExposureIndex
- SensingMethod
- FileSource
- SceneType
- CFAPattern

- CustomRendered
- ExposureMode
- WhiteBalance
- DigitalZoomRatio
- FocalLengthIn35mmFilm
- SceneCaptureType
- GainControl
- Contrast
- Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

- GPSVersionID
- GPSLatitudeRef
- GPSLatitude
- GPSLongitudeRef
- GPSLongitude
- GPSAltitudeRef
- GPSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSStatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum
- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation
- GPSDateStamp
- GPSDifferential

[トップ]

11.1.2.1.5 XPath/XQuery 関数: 数値

Altova の数値拡張関数はXPath とXQuery 内で使用することができ、データを更に処理するための追加機能を提供します。このセクションの関数はAltova の**XPath 3.0** と**XQuery 3.0** エンジンと使用することができます。XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova 拡張関数名前空間、`http://www.altova.com/xslt-extensions` に収められており、`altova:` プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることにご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

自動付番関数

▼ generate-auto-number [altova:]

altova:generate-auto-number(ID *as xs:string*, **StartsWith** *as xs:double*, **Increment** *as xs:double*, **ResetOnChange** *as xs:string***)** を `xs:integer` とする XP1 XP2 XQ1 XP3.1 XQ3.1
関数が呼び出される度に番号を生成します。関数が初めて呼び出される際に生成される最初の番号は`StartsWith` 引数により指定されます。その後の関数の呼び出しは新しい番号を生成します。この番号は前に生成された番号を、`Increment` 引数で指定された値ごとにインクリメントします。実質的には、`altova:generate-auto-number` 関数は、`ID` 引数により指定されたカウンターを作成し、このカウンターが関数が呼び出されるごとにインクリメントされます。`ResetOnChange` 引数の値が、前の関数呼び出しから変更されると、生成される番号の値が`StartsWith` 値にリセットされます。自動付番は`altova:reset-auto-number` 関数を使用してリセットすることができます。

▣ *サンプル*

- **altova:generate-auto-number**(`"ChapterNumber", 1, 1, "SomeString"`) は、関数が呼び出される度に番号を1 つ返します。`1` から始まり、関数が呼び出される度に`1` ずつインクリメントします。その後の呼び出しの4 番目の`"SomeString"`引数がである限り、インクリメントは継続されます。4 番目の引数の値が変更されると、(`ChapterNumber` と呼ばれる) カウンターは1 にリセットされます。`ChapterNumber` の値も`altova:reset-auto-number` 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。

▼ reset-auto-number [altova:]

altova:reset-auto-number(ID *as xs:string***)** XP1 XP2 XQ1 XP3.1 XQ3.1
この関数は、`ID` 引数内で名づけられた自動付番カウンターの番号をリセットします。 引数内のカウンター名を作成した関数の引数により指定された数値にリセットされます。`altova:generate-auto-number`

▣ *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。

[[トップ]]

数値関数

hex-string-to-integer [altova:]

altova:hex-string-to-integer(HexString *as xs:string*) を **xs:integer** とする **XP3.1** **XQ3.1**
10 進のシステム(10進)内の正数の16 進の同値の文字列引数を必要とし、10 進の整数を返します。

サンプル

- **altova:hex-string-to-integer**('1') は1 を返します。
- **altova:hex-string-to-integer**('9') は9 を返します。
- **altova:hex-string-to-integer**('A') は10 を返します。
- **altova:hex-string-to-integer**('B') は11 を返します。
- **altova:hex-string-to-integer**('F') は15 を返します。
- **altova:hex-string-to-integer**('G') はエラーを返します。
- **altova:hex-string-to-integer**('10') は16 を返します。
- **altova:hex-string-to-integer**('01') は1 を返します。
- **altova:hex-string-to-integer**('20') は32 を返します。
- **altova:hex-string-to-integer**('21') は33 を返します。
- **altova:hex-string-to-integer**('5A') は90 を返します。
- **altova:hex-string-to-integer**('USA') はエラーを返します。

integer-to-hex-string [altova:]

altova:integer-to-hex-string(Integer *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
正数を引数として必要とし、文字列として自身のベース16 の同値を返します。

サンプル

- **altova:integer-to-hex-string**(1) は'1' を返します。
- **altova:integer-to-hex-string**(9) は'9' を返します。
- **altova:integer-to-hex-string**(10) は'A' を返します。
- **altova:integer-to-hex-string**(11) は'B' を返します。
- **altova:integer-to-hex-string**(15) は'F' を返します。
- **altova:integer-to-hex-string**(16) は'10' を返します。
- **altova:integer-to-hex-string**(32) は'20' を返します。
- **altova:integer-to-hex-string**(33) は'21' を返します。
- **altova:integer-to-hex-string**(90) は'5A' を返します。

[[トップ]]

数値フォーマット関数

[[トップ]]

11.1.2.1.6 XPath/XQuery 関数: スキーマ

下にリストされる Altova 拡張関数はスキーマの情報を返します。以下は関数の詳細 (i) サンプル (ii) スキーマコンポーネントのリストとパースペクティブのプロパティです。 Altova の**XPath 3.0** と**XQuery 3.0** エンジンと共に使用することができ XPath/XQuery コンテキスト内で見つけることができます。

スキーマドキュメントからのスキーマ情報
関数 `altova:schema` には以下の2つの引数が存在します: 1つのゼロ引数と他の2つの引数。ゼロ引数関数はスキーマ全体を返します。この開始点からスキーマ内をナビゲートし必要とするスキーマコンポーネントをロケートします。2-引数関数は自身の QName により識別される特定のコンポーネントの型を返します。両方の場合、戻り値は関数です。返されたコンポーネント内をナビゲートするには、特定のコンポーネントのプロパティを選択する必要があります。プロパティが非動的アイテム (すなわちコンポーネントの場合) このコンポーネントのプロパティをさらに選択してナビゲートすることが可能です。選択されたプロパティが動的なアイテムの場合、アイテムの値が返され、ナビゲートを更に行うことはできません。

メモ Xquery 式ではスキーマは明示的にインポートされる必要があります。XPath 式では、スキーマは処理環境にインポートされる必要があります。例えば、XSLT には `xslt:import` 命令を使用してインポートします。

XML ノードからのスキーマ情報
関数 `altova:type` は XML ドキュメントのノードを送信しノードの型情報を PSVI から返します。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は**Altova 拡張関数名前空間**、`http://www.altova.com/xslt-extensions` に収められており、`altova:` プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内の XPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内の XQuery 式で使用):	XQ1 XQ3.1

▼ Schema (引数無し)

`altova:schema() as (function(xs:string) as item()*)?` XP3.1 XQ3.1
`schema` コンポーネント全体を返します。`schema` コンポーネントを `schema` コンポーネントのプロパティの一つを選択して更にナビゲートすることができます。

- このプロパティがコンポーネントの場合、このコンポーネントのプロパティの1つを選択して更に深く他のステップをナビゲートすることができます。このステップは更にスキーマをナビゲートするために繰り返すことができます。
- コンポーネントが動的な値の場合、動的な値が返され更にナビゲートすることはできません。

`schema` コンポーネントのプロパティ:

```
"type definitions"
"attribute declarations"
"element declarations"
"attribute group definitions"
"model group definitions"
"notation declarations"
```

```
"identity-constraint definitions"
```

(**schema** 以外の)他のすべてのコンポーネント型のプロパティが下にリストされています。

メモ Xquery 式ではスキーマは明示的にインポートされる必要があります。XPath 式では、スキーマは処理環境にインポートされる必要があります。例えば、XSLT には**xslt:import** 命令を使用してインポートします。

- *サンプル*

  - `import schema "" at "C:\Test\ExpReport.xsd"; for $typedef in altova:schema() ("type definitions") return $typedef ("name")` はスキーマ内のすべての単純型または複合型の名前を返します

  - `import schema "" at "C:\Test\ExpReport.xsd"; altova:schema() ("type definitions")[1]("name")` はスキーマ内の単純型または複合型の最初の名前を返します

コンポーネントとそのプロパティ

- Assertion

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Assertion"
test	XPath プロパティレコード	

- Attribute Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Declaration"
name	文字列	属性のローカル名
target namespace	文字列	属性の名前空間 URI
type definition	Simple Type または Complex Type	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	存在する場合、プロパティを持つ関数 (″class″: ″Value Constraint″、″variety″: ″fixed″ または″default″, ″value″: atomic value, ″lexical form″: string。″value″ プロパティは namespace-sensitive 型のために使用することはできないことに注意してください。	
inheritable	ブール値	

- Attribute Group Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Group Definition"

name	文字列	属性グループのローカル名
target namespace	文字列	属性グループの名前空間 URI
attribute uses	(Attribute Use) のシーケンス	
attribute wildcard	任意の属性のワイルドカード	

Attribute Use

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Use"
required	ブール値	属性が必要な場合はtrue 、任意の場合はfalse
value constraint	Attribute Declaration を参照	
inheritable	ブール値	

Attribute Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	文字列 ("strict"|"lax"|"skip")	

Complex Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
base type definition	Complex Type Definition	
final	文字列のシーケンス ("restriction"|"extension")	
context	空のシーケンス(not implemented)	
derivation method	文字列 ("restriction"|"extension")	
abstract	ブール値	
attribute uses	Attribute Use のシーケンス	
attribute wildcard	任意の属性のワイルドカード	
content type	プロパティを持つ関数: ("class":"Content	

	Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	文字列のシーケンス ("restriction"|"extension")	
assertions	アサーションのシーケンス	

Element Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
type definition	Simple Type または Complex Type	
type table	プロパティを持つ関数 ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	プロパティを持つ関数 ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	Attribute Declaration を参照	
nillable	ブール値	
identity-constraint definitions	Identity Constraint のシーケンス	
substitution group affiliations	Element Declaration のシーケンス	
substitution group exclusions	文字列のシーケンス ("restriction"|"extension")	
disallowed substitutions	文字列のシーケンス ("restriction"|"extension"|"substitution")	
abstract	ブール値	

Element Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not",	

	"namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	文字列 ("strict"|"lax"|"skip")	

Facet

プロパティ名	プロパティ型	プロパティ値
kind	文字列	ファセットの名前、例えば、"minLength" または"enumeration"
value	ファセットによる	ファセットの値
fixed	ブール値	
typed-value	ファセットの列挙のみ、Array(xs:anyAtomicType	列挙値を含む配列。それぞれがアトミック値のシーケンスである場合があります。(メモ 列挙ファセットに関しては、実際の型にかかわらず"value" 値プロパティは文字列のシーケンスです)

Identity Constraint

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Identity-Constraint Definition"
name	文字列	制約のローカル名
target namespace	文字列	制約の名前空間 URI
identity-constraint category	文字列 ("key"|"unique"|"keyRef")	
selector	XPath プロパティレコード	
fields	XPath プロパティレコード のシーケンス	
referenced key	(keyRef のためのみ): Identity Constraint	対応するキー制約

Model Group

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group"
compositor	文字列 ("sequence"|"choice"|"all")	
particles	Particle のシーケンス	

Model Group Definition

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group Definition"
name	文字列	モデルグループのローカル名
target namespace	文字列	モデルグループの名前空間 URI

model group	Model Group	

Notation

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Notation Declaration"
name	文字列	表記のローカル名
target namespace	文字列	表記の名前空間 URI
system identifier	anyURI	
public identifier	文字列	

Particle

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Particle"
min occurs	整数	
max occurs	整数または文字列("unbounded")	
term	Element Declaration、Element Wildcard、または ModelGroup	

Simple Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Simple Type Definition"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
final	文字列のシーケンス ("restriction"|"extension"|"list"|"union")	
context	含まれるコンポーネント	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	(実装されていない)空のシーケンス	
variety	文字列 ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(リスト型のみのための) Simple Type	
member type definitions	(ユニオン型のための) Simple Type のシーケンス	

Type Alternative

プロパティ名	プロパティ型	プロパティ値

kind	文字列	"Type Alternative"
test	XPath プロパティレコード	
type definition	Simple Type または Complex Type	

XPath Property Record

プロパティ名	プロパティ型	プロパティ値
namespace bindings	プロパティを持つ関数のシーケンス("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	XPath 式の静的ベースURI
expression	文字列	文字列としてのXPath 式

Schema (2つの引数)

altova:schema(ComponentKind *as xs:string*, **Name** *as xs:QName*) **as (function(xs:string) as item()*)? XP3.1 XQ3.1**

2番目の引数内で与えられている名前と同じ名前を持つ最初の引数内で指定されているコンポーネントの型を返します。コンポーネントのプロパティの一つを選択して更にナビゲートすることができます。

- このプロパティがコンポーネントの場合、このコンポーネントのプロパティの1つを選択して更に深く他のステップをナビゲートすることができます。このステップは更にスキーマをナビゲートするために繰り返すことができます。
- コンポーネントが動的な値の場合、動的な値が返され更にナビゲートすることはできません。

メモ Xquery 式ではスキーマは明示的にインポートされる必要があります。XPath 式では、スキーマは処理環境にインポートされる必要があります。例えば、XSLT には**xslt:import** 命令を使用してインポートします。

サンプル

- `import schema "" at "C:\Test\ExpReport.xsd";`
  `altova:schema("element declaration", xs:QName("OrgChart"))("type definition")`
  `("content type")("particles")[3]!.("term")("kind")`
  は3番目の`particles` コンポーネントの `kind` プロパティを返します。 この`particles` コンポーネントは`OrgChart` の `Qname` を持つ要素宣言の子孫です。

- `import schema "" at "C:\Test\ExpReport.xsd";`
  `let $typedef := altova:schema("type definition", xs:QName("emailType"))`
  `for $facet in $typedef ("facets")`
  `return [$facet ("kind"), $facet("value")]`
  は各 **emailType** コンポーネントの各 **facet** にそのファセットの型と値を含む列挙を返します。

コンポーネントとそのプロパティ

Assertion

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Assertion"

test	XPath プロパティレコード	

Attribute Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Declaration"
name	文字列	属性のローカル名
target namespace	文字列	属性の名前空間 URI
type definition	Simple Type または Complex Type	
scope	プロパティを持つ関数 ("class": "Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	存在する場合、プロパティを持つ関数 ("class": "Value Constraint"、"variety": "fixed" または "default", "value": atomic value, "lexical form": string。"value" プロパティは namespace-sensitive 型のために使用することはできないことにご注意してください。	
inheritable	ブール値	

Attribute Group Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Group Definition"
name	文字列	属性グループのローカル名
target namespace	文字列	属性グループの名前空間 URI
attribute uses	(Attribute Use) のシーケンス	
attribute wildcard	任意の属性のワイルドカード	

Attribute Use

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Use"
required	ブール値	属性が必要な場合は true 、任意の場合は false
value constraint	Attribute Declaration を参照	
inheritable	ブール値	

Attribute Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"

namespace constraint	プロパティを持つ関数 ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	文字列 ("strict"|"lax"|"skip")	

Complex Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
base type definition	Complex Type Definition	
final	文字列のシーケンス ("restriction"|"extension")	
context	空のシーケンス(not implemented)	
derivation method	文字列 ("restriction"|"extension")	
abstract	ブール値	
attribute uses	Attribute Use のシーケンス	
attribute wildcard	任意の属性のワイルドカード	
content type	プロパティを持つ関数: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	文字列のシーケンス ("restriction"|"extension")	
assertions	アサーションのシーケンス	

Element Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
type definition	Simple Type または Complex Type	
type table	プロパティを持つ関数 ("class":"Type Table",	

	"alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	Attribute Declaration を参照	
nillable	ブール値	
identity-constraint definitions	Identity Constraint のシーケンス	
substitution group affiliations	Element Declaration のシーケンス	
substitution group exclusions	文字列のシーケンス ("restriction"|"extension")	
disallowed substitutions	文字列のシーケンス ("restriction"|"extension"|"substitution")	
abstract	ブール値	

Element Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″class″: ″Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	文字列 (″strict″|″lax″|″skip″)	

Facet

プロパティ名	プロパティ型	プロパティ値
kind	文字列	ファセットの名前、例えば、″minLength″ または″enumeration″
value	ファセットによる	ファセットの値
fixed	ブール値	
typed-value	ファセットの列挙のみ、 Array(xs:anyAtomicType	列挙値を含む配列。それぞれがアトミック値のシーケンスである場合があります。(メモ 列挙ファセットに関しては、実際の型にかかわらず″value″ 値プロパティは文字列のシーケンスです)

Identity Constraint

プロパティ名	プロパティ型	プロパティ値

kind	文字列	"Identity-Constraint Definition"
name	文字列	制約のローカル名
target namespace	文字列	制約の名前空間 URI
identity-constraint category	文字列 ("key"|"unique"|"keyRef")	
selector	XPath プロパティレコード	
fields	XPath プロパティレコード のシーケンス	
referenced key	(keyRef のためのみ): Identity Constraint	対応するキー制約

Model Group

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group"
compositor	文字列 ("sequence"|"choice"|"all")	
particles	Particle のシーケンス	

Model Group Definition

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group Definition"
name	文字列	モデルグループのローカル名
target namespace	文字列	モデルグループの名前空間 URI
model group	Model Group	

Notation

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Notation Declaration"
name	文字列	表記のローカル名
target namespace	文字列	表記の名前空間 URI
system identifier	anyURI	
public identifier	文字列	

Particle

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Particle"
min occurs	整数	
max occurs	整数または文字列("unbounded")	
term	Element Declaration、Element Wildcard、または ModelGroup	

Simple Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Simple Type Definition"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
final	文字列のシーケンス ("restriction"|"extension"|"list"|"union")	
context	含まれるコンポーネント	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	(実装されていない)空のシーケンス	
variety	文字列 ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(リスト型のみのための) Simple Type	
member type definitions	(ユニオン型のための) Simple Type のシーケンス	

Type Alternative

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Type Alternative"
test	XPath プロパティレコード	
type definition	Simple Type または Complex Type	

XPath Property Record

プロパティ名	プロパティ型	プロパティ値
namespace bindings	プロパティを持つ関数のシーケンス("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	XPath 式の静的ベース URI
expression	文字列	文字列としての XPath 式

型

altova:type(Node *as item?***) as (function(xs:string) as item()*)? XP3.1 XQ3.1**
関数 **altova:type** は XML ドキュメントの要素または属性ノードを送信しノードの型情報を PSVI から返します。

メモ スキーマが参照可能になるために XML ドキュメントはスキーマ宣言を持つ必要があります。

サンプル

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type
  ```
  はノードの型情報を含む関数を返します。

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type ("kind")
  ```
  はノードの型のコンポーネント（単純型または複合型）を取りコンポーネントの`kind` property の値を返します。

コンポーネントとそのプロパティ

Assertion

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Assertion"
test	XPath プロパティレコード	

Attribute Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Declaration"
name	文字列	属性のローカル名
target namespace	文字列	属性の名前空間 URI
type definition	Simple Type または Complex Type	
scope	プロパティを持つ関数（"class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	存在する場合、プロパティを持つ関数（"class": "Value Constraint"、"variety": "fixed" または "default", "value": atomic value, "lexical form": string。"value" プロパティは namespace-sensitive 型のために使用することはできないことにご注意してください。	
inheritable	ブール値	

Attribute Group Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Group Definition"
name	文字列	属性グループのローカル名
target namespace	文字列	属性グループの名前空間 URI
attribute uses	（Attribute Use）のシーケンス	

attribute wildcard	任意の属性のワイルドカード	

Attribute Use

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Attribute Use"
required	ブール値	属性が必要な場合はtrue 、任意の場合はfalse
value constraint	Attribute Declaration を参照	
inheritable	ブール値	

Attribute Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″class″: ″Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	文字列 (″strict″|″lax″|″skip″)	

Complex Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
base type definition	Complex Type Definition	
final	文字列のシーケンス ("restriction"|"extension")	
context	空のシーケンス(not implemented)	
derivation method	文字列 (″restriction″|″extension″)	
abstract	ブール値	
attribute uses	Attribute Use のシーケンス	
attribute wildcard	任意の属性のワイルドカード	
content type	プロパティを持つ関数: (″class″:″Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string	

	("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	文字列のシーケンス ("restriction"|"extension")	
assertions	アサーションのシーケンス	

Element Declaration

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Complex Type"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
type definition	Simple Type または Complex Type	
type table	プロパティを持つ関数 (″class″:″Type Table″, "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	プロパティを持つ関数 (″class″:″Scope″, "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	Attribute Declaration を参照	
nillable	ブール値	
identity-constraint definitions	Identity Constraint のシーケンス	
substitution group affiliations	Element Declaration のシーケンス	
substitution group exclusions	文字列のシーケンス ("restriction"|"extension")	
disallowed substitutions	文字列のシーケンス ("restriction"|"extension"|"substitution")	
abstract	ブール値	

Element Wildcard

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Wildcard"
namespace constraint	プロパティを持つ関数 (″class″: ″Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	

process contents	文字列 ("strict"|"lax"|"skip")	

Facet

プロパティ名	プロパティ型	プロパティ値
kind	文字列	ファセットの名前、例えば、"minLength" または"enumeration"
value	ファセットによる	ファセットの値
fixed	ブール値	
typed-value	ファセットの列挙のみ、Array(xs:anyAtomicType	列挙値を含む配列。それぞれがアトミック値のシーケンスである場合があります。(メモ 列挙ファセットに関しては、実際の型にかかわらず"value" 値プロパティは文字列のシーケンスです)

Identity Constraint

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Identity-Constraint Definition"
name	文字列	制約のローカル名
target namespace	文字列	制約の名前空間 URI
identity-constraint category	文字列 ("key"|"unique"|"keyRef")	
selector	XPath プロパティレコード	
fields	XPath プロパティレコード のシーケンス	
referenced key	(keyRef のためのみ): Identity Constraint	対応するキー制約

Model Group

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group"
compositor	文字列 ("sequence"|"choice"|"all")	
particles	Particle のシーケンス	

Model Group Definition

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Model Group Definition"
name	文字列	モデルグループのローカル名
target namespace	文字列	モデルグループの名前空間 URI
model group	Model Group	

Notation

プロパティ名	プロパティ型	プロパティ値

kind	文字列	"Notation Declaration"
name	文字列	表記のローカル名
target namespace	文字列	表記の名前空間 URI
system identifier	anyURI	
public identifier	文字列	

Particle

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Particle"
min occurs	整数	
max occurs	整数または文字列("unbounded")	
term	Element Declaration、Element Wildcard、または ModelGroup	

Simple Type

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Simple Type Definition"
name	文字列	(匿名の場合は空)型のローカル名
target namespace	文字列	(匿名の場合は空)型の名前空間 URI
final	文字列のシーケンス ("restriction"|"extension"|"list"|"union")	
context	含まれるコンポーネント	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	(実装されていない)空のシーケンス	
variety	文字列 ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(リスト型のみのための) Simple Type	
member type definitions	(ユニオン型のための) Simple Type のシーケンス	

Type Alternative

プロパティ名	プロパティ型	プロパティ値
kind	文字列	"Type Alternative"
test	XPath プロパティレコード	
type definition	Simple Type または Complex Type	

XPath Property Record

プロパティ名	プロパティ型	プロパティ値
namespace bindings	プロパティを持つ関数のシーケンス("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	XPath 式の静的ベースURI
expression	文字列	文字列としてのXPath 式

11.1.2.1.7 XPath/XQuery 関数: シーケンス

Altova のシーケンス拡張関数はXPath とXQuery 式で使用することができ、XML スキーマの異なる日付および時刻データ型で保存されているデータを処理するための追加機能を提供します。このセクションの関数は、Altova の**XPath 3.0** および**XQuery 3.0** エンジンで使用することができます。これらの関数は、XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数はXPath/XQuery 式で使用することができ、XPath、XQuery、およびXSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数はAltova 拡張関数名前空間、`http://www.altova.com/xslt-extensions` に収められており、`altova:` プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内のXPath 式で使用):	XP1 XP2 XP3.1
XSLT 関数 (XSLT 内のXPath 式で使用):	XSLT1 XSLT2 XSLT3
XQuery 関数 (XQuery 内のXQuery 式で使用):	XQ1 XQ3.1

attributes [altova:]

altova:attributes(AttributeName *as xs:string*) を**attribute()*** とする XP3.1 XQ3.1
入力引数 AttributeName 内で与えられた名前と同じローカル名を持つすべての属性を返します。検索は大文字と小文字を区別し、attribute:: 軸に対して行われます。これは、コンテキストノードが親要素ノードである必要があることを意味します。

サンプル

- **altova:attributes**("MyAttribute") はMyAttribute()* を返します。

altova:attributes(AttributeName *as xs:string*, **SearchOptions** *as xs:string*)
asattribute()* XP3.1 XQ3.1
入力引数 AttributeName 内で与えられた名前と同じローカル名を持つすべての属性を返します。検索は大文字と小文字を区別し、attribute:: 軸に対して行われます。コンテキストノードが親要素ノードである必要があります。第 2 引数はオプションのフラグを含みます。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 **XP3.1** **XQ3.1**
入力引数 ElementName で与えられた名前と同じローカル名を持つすべての要素を返します。検索は大文字と小文字を区別して child:: 軸に対して実行されます。コンテキストノードは、検索される要素の親ノードである必要があります。

サンプル

- **altova:elements**("MyElement") は MyElement()* を返します。

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string*) **aselement()*** **XP3.1** **XQ3.1**
入力引数 ElementName 内で与えられた名前と同じローカル名を持つすべての属性を返します。検索は大文字と小文字を区別し、child:: 軸に対して行われます。コンテキストノードが親要素ノードである必要があります。第 2 引数はオプションのフラグを含みます。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 **XP3.1** **XQ3.1**
この関数は2 つの引数を必要とします。最初の引数は1 つ、または1 つ以上のデータ型のアイテムのシーケンスです。第 2 の引数 Condition は、(1 のアリティを持つ) 1 つの引数 を必要とし、boolean を返す XPath 関数に対する参照です。Condition で参照された関数の代わりに、Sequence の各アイテムが提出されます。(*注意:* この関数は1 つの引数のみを必要とします。) Condition 内の関数にtrue() と評価させる最初の Sequence アイテムは、altova:find-first、反復の終了の結果として返されます。

サンプル

- **altova:find-first**(5 to 10, function($a) {$a mod 2 = 0}) はxs:integer 6 を返します。Condition 引数は、$a という名のインライン関数を宣言し、定義します。XPath 3.0 インライン関数 function()を参照します。altova: Sequence 引数内の各アイテムでは、find-first がパスされ、代わりに、$a を入力値とします。入力値は関数定義($a mod 2 = 0)内の条件に対してテストされます。この条件を満たす最初の入力値が altova:find-first (この場合は6) の結果として返されます。

- **altova:find-first**((1 to 10), (function($a) {$a+3=7})) はxs:integer 4 を返します。

更なるサンプル
ファイルC:\Temp\Customers.xml が存在する場合:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) はxs:string C:\Temp\Customers.xml を返します。

ファイルC:\Temp\Customers.xml が存在せず、http://www.altova.com/index.html が存在する場合:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) はxs:string http://www.altova.com/index.html を返します。

ファイルC:\Temp\Customers.xml が存在せず、http://www.altova.com/index.html も存在しない場合:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) 結果無しを返します。

上のサンプルについての注意点

- XPath 3.0 関数 doc-available はURI として使用され、ドキュメントノードが提出されたURI で検出される場合 true を返する単一の引数を必要とします。(ですから、提出されたURI でのドキュメントはXML ドキュメントである必要があります。)
- doc-available 関数は、altova:find-first の第 2 引数である、Condition で使用することができます。これは、1 つの引数 (アリティ=1) のみを必要とするからであり、item()を入力(URI として使用される文字列) として、boolean の値を返すからです。
- doc-available 関数は、参照されているだけで、呼び出されていない点に注意してください。アタッチされている #1 サフィックスは関数が1 つのアリティであることを表示するためです。doc-available#1 の意味は以下のとおりです: アリティ=1 を持つ doc-availabe() 関数を使用し、最初のシーケンスの各アイテムの代わりに単一引数としてパスします。この結果、 2 つの文字列の各自つは、文字列をURIとして使用し、URIにドキュメントノードが存在するかテストする **doc-available()** にパスされます。1 つが各当する場合、**doc-available()** 関数はtrue() を評価し、シーケンス内のその文字列のインデックス ポジションは、 **altova:find-first** 関数の結果として返されます。doc-available() 関数に関しての注意点: 相対パスは、デフォルトで関数がロードされるXML ドキュメントの現在のベースURI に対して相対的に解決されます。

▼ find-first-combination [altova:]

altova:find-first-combination((Seq-01 *as item()**)**,* **(Seq-02** *as item()**)**,* **(Condition(Seq-01-Item, Seq-02-Item** *as xs:boolean*)) を**item()*** とする **XP3.1** **XQ3.1**
この関数は3 つの引数を必要とします:

- 最初の2 つの引数 **Seq-01** と**Seq-02**, は、1 つまたは1 つ以上のデータ型のアイテムです。
- 第 3 の引数 **Condition** は、(2 のアリティを持つ) 2 つの引数を必要とし、boolean を返す XPath 関数に対しての参照です。

Seq-01 と**Seq-02** のアイテムは指定された組み合わせ(各シーケンスからの1 つずつ)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。以下の点にご注意してください: (i) **Condition** 関数が提出された引数ペア内で繰り返され、**true()** を評価しない場合、**altova:find-first-combination** は、結果を返しません; (ii) **altova:find-first-combination** の結果が常に(データ型の)アイテムのペアである場合またはアイテムでない場合。

▼ *サンプル*

- **altova:find-first-combination**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) はxs:integers (11, 21) のシーケンスを返します。
- **altova:find-first-combination**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) はxs:integers (11, 22) のシーケンを返します。
- **altova:find-first-combination**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 34}) は xs:integers (11, 23) のシーケンを返します。

▼ find-first-pair [altova:]

altova:find-first-pair((Seq-01 *as item()**)**,* **(Seq-02** *as item()**)**,* **(Condition(Seq-01-Item, Seq-02-Item** *as xs:boolean*)) を**item()*** とする **XP3.1** **XQ3.1**
この関数は3 つの引数を必要とします:

- 最初の2 つの引数 **Seq-01** と**Seq-02**, は、1 つまたは1 つ以上のデータ型のアイテムです。
- 第 3 の引数 **Condition** は、(2 のアリティを持つ) 2 つの引数を必要とし、boolean を返す XPath 関数に対しての参照です。

Seq-01 と**Seq-02** のアイテムは指定された組み合わせで、**Condition** 内の関数の引数としてパスされました。組み合わせは以下のように指定されています。

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

Condition 関数に**true()** と評価するように指示する最初のペアは**altova:find-first-pair** の結果として返されます。以下の点にご注意してください: (i) **Condition** 関数が提出された引数ペア内で繰り返され、**true()** を評価しない場合、**altova:find-first-pair** は、結果を返しません; (ii) **altova:find-first-combination** の結果が常に(データ

型の) アイテムのペアである場合またはアイテムでない場合。

サンプル

- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` は `xs:integers (11, 21)` *のシーケンス*を返します。
- `altova:find-first-pair(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` は*結果無し*を返します。

上の2 つのサンプルに表示される通り、ペアの順序は以下の通りです: `(11, 21) (12, 22) (13, 23)...(20, 30)`。(`33` を返す指示されたペアがないため) この理由で第 2 のサンプルは *結果無し*を返します。

find-first-pair-pos [altova:]

`altova:find-first-pair-pos((Seq-01 as item()*), (Seq-02 as item()*), (Condition( Seq-01-Item, Seq-02-Item as xs:boolean))` を **`xs:integer`** とする **XP3.1** **XQ3.1**
この関数は3 つの引数を必要とします:

- 最初の2 つの引数 **`Seq-01`** と **`Seq-02`**, は、1 つまたは 1 つ以上のデータ型のアイテムです。
- 第 3 の引数 **`Condition`** は、(2 のアリティを持つ) 2 つの引数を必要とし、boolean を返す XPath 関数に対しての参照です。

`Seq-01` と **`Seq-02`** のアイテムは指定された組み合わせで、**`Condition`** 内の関数の引数としてパスされました。組み合わせは以下のように指定されています。

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

`Condition` 関数に **`true()`** を評価させる、最初に指示されたペアのインデックス ポジションは、は **`altova:find-first-pair-pos`** の結果として返されます。関数が提出された引数ペア内で繰り返され、**`true()`** を一度も評価しない場合、**`altova:find-first-pair-pos`** *結果無し*が返します。

サンプル

- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` は `1` を返します
- `altova:find-first-pair-pos(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` は*結果無し*を返します。

上の2 つのサンプルに表示される通り、ペアの順序は以下の通りです: `(11, 21) (12, 22) (13, 23)...(20, 30)`。最初のサンプルでは、最初のペアは **`Condition`** 関数に **`true()`** を評価させ、シーケンス内のインデックス ポジションには **`1`** が返されます。第 2 のサンプルでは、`33` を返すペアがないため、*結果無し*が返します。

find-first-pos [altova:]

`altova:find-first-pos((Sequence as item()*), (Condition( Sequence-Item as xs:boolean))` を **`xs:integer`** とする **XP3.1** **XQ3.1**
この関数は2 つの引数を必要とします。最初の引数は1 つ、または1 つ以上のデータ型のアイテムの*シーケンス*です。第 2 の引数 `Condition` は、(`1` のアリティを持つ) 1 つの引数 を必要とし、boolean を返す XPath 関数に対する参照です。`Condition` で参照された関数の代わりに、**`Sequence`** の各アイテムが提出されます。(*注意:* この関数は1 つの引数のみを必要とします。) **`Condition`** 内の関数に **`true()`** と評価させる最初の **`Sequence`** アイテムは、**`altova:find-first-pos`** の結果として返された **`Sequence`** 内インデックス ポジションを持ちます。

サンプル

- **altova:find-first-pos**(5 to 10, function($a) {$a mod 2 = 0}) は xs:integer 2 を返します。

Condition 引数は、**$a** という名のインライン関数を宣言し、定義します。XPath 3.0 インライン関数 **function()** を参照します。 **Sequence** 引数内の各アイテムでは、**find-first-pos** が パスされ、代わりに、**$a** を入力値とします。入力値は関数定義($a mod 2 = 0)内の条件に対してテストされます。この条件を満たす最初の入力値が **altova:find-first-pos** (シーケンス内で条件を満たす最初の値である 6 がシーケンスのインデクス位置 2 にあるためこの場合は 2、) の結果として返されます。

- **altova:find-first-pos**((2 to 10), (function($a) {$a+3=7})) は xs:integer 3 を返します。

更なるサンプル

ファイル **C:\Temp\Customers.xml** が存在する場合:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) は 1 を返します。

ファイル **C:\Temp\Customers.xml** が存在せず、**http://www.altova.com/index.html** が存在する場合:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) は 2 を返します。

ファイル **C:\Temp\Customers.xml** が存在せず、**http://www.altova.com/index.html** も存在しない場合:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) 結果無しを返します。

上のサンプルについての注意点

- XPath 3.0 関数 doc-available は URI として使用され、ドキュメントノードが提出された URI で検出される場合 true を返する単一の引数を必要とします。(ですから、提出された URI でのドキュメントは XML ドキュメントである必要があります。)
- doc-available 関数は、altova:find-first-pos の第 2 引数である、**Condition** で使用することができます。これは、1 つの引数 (アリティ=1) のみを必要とするからであり、item() を入力(URI として使用される文字列) として、boolean の値を返すからです。
- doc-available 関数は、参照されているだけで、呼び出されていない点に注意してください。アタッチされている #1 サフィックスは関数が 1 つのアリティであることを表示するためです。doc-available#1 の意味は以下のとおりです: アリティ=1 を持つ doc-availabe() 関数を使用し、最初のシーケンスの各アイテムの代わりに 単一引数として パスします。 この結果、 2 つの文字列の各自つは、文字列をURIとして使用し、URIにドキュメントノードが存在するかテストする **doc-available()** に パスされます。1 つが各当する場合、**doc-available()** 関数は true() を評価し、シーケンス内のその文字列のインデックス ポジションは、 **altova:find-first-pos** 関数の結果として返されます。
doc-available() 関数に関しての注意点: 相対パスは、デフォルトで関数がロードされる XML ドキュメントの現在のベース URI に対して相対的に解決されます。

for-each-attribute-pair [altova:]

altova:for-each-attribute-pair(Seq1 *as element()?*, **Seq2** *as element()?*, **Function** *as function()*) **asitem()*** **XP3.1** **XQ3.1**

ペアの一つの属性が最初の要素を取得し、もう一つの属性が2番目の要素から取得された箇所で、最初の2つの引数は、2つの要

素、および、属性ペアを作成するために使用された属性を識別します。要素ペアは同じ名前を持つことをベースに選択され、ペアは名前別に、アルファベット順にセットに並べ替えられます。属性のために他の要素上に対応する属性が存在しない場合、ペアのジョイントは解除されます。これは、一つのメンバーにより飲み構成されることを意味します。関数アイテム(3番目の引数 `Function`) はアイテムのシーケンスである出力であるペアのシーケンス内のペア(ジョイント、ジョイントの解除) に個別に適用されます。

サンプル

- **altova:for-each-attribute-pair**(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b}) は以下 を返します。...

```
(2, 4, 6) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />

(2, 4, 6) if
<Test-A att2="2" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />

(2, 6) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

メモ 結果 (2, 6) は次のアクションにより取得されます: (1+1, ()+2, 3+3, 4+()) 。演算子の一つが空のシーケンスの場合、アイテム 2 と 4 の場合同様、結果は空のシーケンスになります。

- **altova:for-each-attribute-pair**(/Example/Test-A, /Example/Test-B, concat#2) は以下を返します。

```
(11, 22, 33) if
<Test-A att1="1" att2="2" att3="3" />
<Test-B att1="1" att2="2" att3="3" />

(11, 2, 33, 4) if
<Test-A att4="4" att1="1" att3="3" />
<Test-B att3="3" att2="2" att1="1" />
```

for-each-combination [altova:]

altova:for-each-combination(FirstSequence *as item()**, **SecondSequence** *as item()**, **function($i,$j){$i || $j})** を **item()*** とする **XP3.1** **XQ3.1**
第 3 の引数として与えられた、インライン関数を返します。`FirstSequence` 内の各 `$i` を `SecondSequence` 内の各 `$j` と結合します。出力はこれらのアイテムのシーケンスです。

サンプル

- **altova:for-each-combination**(('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j}) は ('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3') を返します。

for-each-combination [altova:]

altova:for-each-combination(FirstSequence *as item()**, **SecondSequence** *as item()**, **Function($i,$j){$i || $j}) asitem()*** **XP3.1** **XQ3.1**
最初の2つの引数内の2つのシーケンスのアイテムは最初のシーケンスの各アイテムが2番目のシーケンスになるように結合されるように結合されます。3番目の引数として与えられる関数は結果のシーケンス内の組み合わせに適用され、アイテムのシーケンスである出力内で

出力されます（*サンプル参照*）。

サンプル

- **altova:for-each-combination**(('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j}) は ('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3') を返します。

▼ substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()*) を **item()*** とする **XP3.1** **XQ3.1**
FirstSequence が空の場合、SecondSequence を返します。FirstSequence が空でない場合、FirstSequence を返します。

サンプル

- **altova:substitute-empty**((1,2,3), (4,5,6)) は (1,2,3) を返します。
- **altova:substitute-empty**((), (4,5,6)) は (4,5,6) を返します。

11.1.2.1.8 XPath/XQuery 関数: 文字列

Altova の文字列拡張関数は XPath と XQuery 式で使用することができ、XML スキーマの異なる日付および時刻データ型で保存されているデータを処理するための追加機能を提供します。このセクションの関数は、Altova の **XPath 3.0** および **XQuery 3.0** エンジンで使用することができます。これらの関数は、XPath/XQuery コンテキストで使用することができます。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は **Altova 拡張関数名前空間**、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数（XSLT 内の XPath 式で使用）:	**XP1** **XP2** **XP3.1**
XSLT 関数（XSLT 内の XPath 式で使用）:	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数（XQuery 内の XQuery 式で使用）:	**XQ1** **XQ3.1**

▼ camel-case [altova:]

altova:camel-case(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
入力文字列を **InputString** をキャメルケースで返します。文字列は空白スペースのショートカットである）正規表現 **'\s'** を使用して分析されます。空白文字または連続する空白文字のシーケンスの後の最初の非空白スペース文字は大文字です。出力文字列の最初の文字は大文字です。

サンプル

- **altova:camel-case**("max") Max を返します。
- **altova:camel-case**("max max") Max Max を返します。
- **altova:camel-case**("file01.xml") File01.xml を返します。

- **altova:camel-case**("file01.xml file02.xml") File01.xml File02.xml を返します。
- **altova:camel-case**("file01.xml file02.xml") File01.xml File02.xml を返します。
- **altova:camel-case**("file01.xml -file02.xml") File01.xml -file02.xml を返します。

altova:camel-case(InputString *as xs:string*, **SplitChars** *as xs:string*, **IsRegex** *as xs:boolean*) を**xs:string** とする **XP3.1** **XQ3.1**
SplitChars を使用して、次の大文字をトリガーする文字を決定し、入力文字列を **InputString** キャメルケースに変換します。 **SplitChars** は、**IsRegex = true()** の場合、または、**IsRegex = false()** の場合、プレーン文字は正規表現として使用されます。出力文字列の最初の文字は大文字です。

サンプル

- **altova:camel-case**("setname getname", "set|get", true()) setName getName を返します。
- **altova:camel-case**("altova\documents\testcases", "\", false()) Altova\Documents\Testcases を返します。

char [altova:]

altova:char(Position *as xs:integer*) を**xs:string** とする **XP3.1** **XQ3.1**
xs:string.Position に対してのコンテキストアイテムの値を変換することにより得られた文字列内の Position 引数により指定されたポジションにある文字を含む文字列を返します。引数により提出されたインデックスに文字が存在しない場合、結果文字列は空です。

サンプル

コンテキスト アイテムが **1234ABCD** の場合:

- **altova:char**(2) は 2 を返します。
- **altova:char**(5) は A を返します。
- **altova:char**(9) は空の文字列を返します。
- **altova:char**(-2) は空の文字列を返します。

altova:char(InputString *as xs:string*, **Position** *as xs:integer*) を**xs:string** とする **XP3.1** **XQ3.1**
引数として提出された文字列内の Position 引数により指定されたポジションでの文字を含む文字列を返します。Position 引数により提出されたインデックスに文字が存在しない場合、結果文字列は空です。

サンプル

- **altova:char**("2014-01-15", 5) は - を返します。
- **altova:char**("USA", 1) は U を返します。
- **altova:char**("USA", 10) は空の文字列を返します。
- **altova:char**("USA", -2) は空の文字列を返します。

create-hash-from-string[altova:]

altova:create-hash-from-string(InputString *as xs:string*) **asxs:string** **XP2** **XQ1** **XP3.1** **XQ3.1**
altova:create-hash-from-string(InputString *as xs:string*, **HashAlgo** *as xs:string*) **asxs:string** **XP2** **XQ1** **XP3.1** **XQ3.1**
HashAlgo 引数により指定されているハッシュアルゴリズムを使用して InputString からハッシュ文字列を生成します。次のハッシュアルゴリズムは（大文字、または、小文字で指定されている可能性があります: **MD5**, **SHA-224**, **SHA-256**, **SHA-384**, **SHA-512**。（最初の署名を参照してください）2番目の引数が指定されていない場合、**SHA-256** ハッシュアルゴリズムが使用され

ます。

サンプル

- `altova:create-hash-from-string('abc')` は **SHA-256** ハッシュアルゴリズムを使用して生成されたハッシュ文字列 を返します。
- `altova:create-hash-from-string('abc', 'md5')` は **MD5** ハッシュアルゴリズムを使用して生成されたハッシュ文字列 を返します。
- `altova:create-hash-from-string('abc', 'MD5')` は **MD5** ハッシュアルゴリズムを使用して生成されたハッシュ文字列 を返します。

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
`xs:string` に対するコンテキストアイテムの値を変換することにより得られた最初の `X-Number` 文字を含む文字列を返します。

サンプル

コンテキストアイテムが **1234ABCD** の場合:

- `altova:first-chars(2)` は `12` を返します。
- `altova:first-chars(5)` は `1234A` を返します。
- `altova:first-chars(9)` は `1234ABCD` を返します。

altova:first-chars(InputString *as xs:string*, **X-Number** *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
`InputString` 引数として提出された文字列の最初の文字を含む文字列を返します。

サンプル

- `altova:first-chars("2014-01-15", 5)` は `2014-` を返します。
- `altova:first-chars("USA", 1)` は `U` を返します。

format-string [altova:]

altova:format-string(InputString *as xs:string*, **FormatSequence** *as item()**) **as xs:string** **XP3.1** **XQ3.1**
入力文字列（最初の引数）には、配置パラメーター（`%1`, `%2`, etc）が含まれています。各パラメーターは、（2番目の引数として提出されている）フォーマットシーケンス内の対応するポジションでロケートされる文字列アイテムと置き換えられます。フォーマットシーケンス内の最初のアイテムは、配置パラメーター `%1` を置換、二番目のアイテムは `%2` を置き換えます。関数は、書式設定された代替を持つ文字列を返します。配置パラメーターに文字列が存在しない場合は、配置パラメーターが返されます。これは、配置パラメーターのインデックスが書式シーケンス内のアイテムの数よりも大きい場合発生します。

サンプル

- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe'))` は `"Hello Jane, John, Joe"` を返します。
- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe', 'Tom'))` は `"Hello Jane, John, Joe"` を返します。
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe', 'Tom'))` は `"Hello Jane, John, Tom"` を返します。
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe'))` は `"Hello Jane, John, %4"` を返します。

last-chars [altova:]

altova:last-chars(X-Number *as xs:integer*) を **xs:string** とする **XP3.1** **XQ3.1**
xs:string に対してのコンテキストアイテムの値の変換より取得された文字列の最後の X-Number 文字を含んでいる文字列を返します。

サンプル

コンテキストアイテムが **1234ABCD** の場合:

- **altova:last-chars**(2) は CD を返します。
- **altova:last-chars**(5) は 4ABCD を返します。
- **altova:last-chars**(9) は 1234ABCD を返します。

altova:last-chars(InputString *as xs:string*, **X-Number** *as xs:integer*) **as xs:string** とする **XP3.1** **XQ3.1**
引数として提出された文字列の最後の X-Number 文字を含んでいる文字列を返します。

サンプル

- **altova:last-chars**("2014-01-15", 5) は 01-15 を返します。
- **altova:last-chars**("USA", 10) は USA を返します。

pad-string-left [altova:]

altova:pad-string-left(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
PadCharacter 引数は1文字です。文字列の左側にパッドされ、この数が、StringLength 引数の整数の値と等しくなるように StringToPad の文字の数を増やします。StringLength 引数は任意の整数の値（正数または負数）を持つことができますが、パディングは、StringLength の値が StringToPad 内の文字数より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[パド文字が長すぎます] エラーを返します。

pad-string-right [altova:]

altova:pad-string-right(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
PadCharacter 引数は1文字です。文字列の右側にパッドされ、この数が、StringLength 引数の整数の値と等しくなるように StringToPad の文字の数を増やします。StringLength 引数は任意の整数の値（正数または負数）を持つことができますが、パディングは、StringLength の値が StringToPad 内の文字数より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** **XP3.1** **XQ3.1**
最初のInputString引数により構成される文字列、Repeats 回繰り返してを生成します。

▣ *サンプル*

- **altova:repeat-string**("Altova #", 3) は"Altova #Altova #Altova #" を返します。

▼ substring-after-last [altova:]

altova:substring-after-last(MainString *as xs:string*, **CheckString** *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
CheckString がMainString 内で検出された場合、MainString 内のCheckString が発生した後のサブ文字列が返されます。MainString 内でCheckString が検出されない場合、空の文字列が返されます。CheckString が空の文字列の場合、MainString 全体が返されます。一度以上発生する場合、CheckString の最後の発生の後の サブ文字列が返されます。

▣ *サンプル*

- **altova:substring-after-last**('ABCDEFGH', 'B') は'CDEFGH' を返します。
- **altova:substring-after-last**('ABCDEFGH', 'BC') は'DEFGH' を返します。
- **altova:substring-after-last**('ABCDEFGH', 'BD') は'' を返します。
- **altova:substring-after-last**('ABCDEFGH', 'Z') は'' を返します。
- **altova:substring-after-last**('ABCDEFGH', '') は'ABCDEFGH' を返します。
- **altova:substring-after-last**('ABCD-ABCD', 'B') は'CD' を返します。
- **altova:substring-after-last**('ABCD-ABCD-ABCD', 'BCD') は'' を返します。

▼ substring-before-last [altova:]

altova:substring-before-last(MainString *as xs:string*, **CheckString** *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
CheckString がMainString 内で検出された場合、MainString 内のCheckString が発生する前のサブ文字列が返されます。MainString 内でCheckString が一度以上発生する場合、CheckString の最後の発生の前の サブ文字列が返されます。

▣ *サンプル*

- **altova:substring-before-last**('ABCDEFGH', 'B') は'A' を返します。
- **altova:substring-before-last**('ABCDEFGH', 'BC') は'A' を返します。
- **altova:substring-before-last**('ABCDEFGH', 'BD') は'' を返します。
- **altova:substring-before-last**('ABCDEFGH', 'Z') は'' を返します。
- **altova:substring-before-last**('ABCDEFGH', '') は'' を返します。
- **altova:substring-before-last**('ABCD-ABCD', 'B') は'ABCD-A' を返します。
- **altova:substring-before-last**('ABCD-ABCD-ABCD', 'ABCD') は'ABCD-ABCD-' を返します。

▼ substring-pos [altova:]

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*) を

xs:integer とする **XP3.1** **XQ3.1**
StringToCheck 内での StringToFind の最初の発生の文字位置を整数として返します。StringToCheck の最初の文字は、位置 1 にあります。StringToFind が StringToCheck 内で発生しない場合、整数 0 が返されます。第 2 またはその後の StringToCheck, 発生を確認するには、この関数の次の署名を確認してください。

サンプル

- **altova:substring-pos**('Altova', 'to') は 3 を返します。
- **altova:substring-pos**('Altova', 'tov') は 3 を返します。
- **altova:substring-pos**('Altova', 'tv') は 0 を返します。
- **altova:substring-pos**('AltovaAltova', 'to') は 3 を返します。

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*, **Integer** *as xs:integer*) を **xs:integer** とする **XP3.1** **XQ3.1**
StringToCheck 内での StringToFind の最初の発生の文字位置を返します。Integer 引数により与えられた、文字位置から StringToFind の検索が開始されます。この位置の前の文字サブ文字列は検索されません。しかし、返された整数は、全体文字列 StringToCheck の検索された文字列の位置です。この署名は、StringToCheck 内で複数回発生する、第 2 または後の発生を検索する際に役に立ちます。 StringToFind が StringToCheck 内で発生しない場合、整数 0 が返されます。

サンプル

- **altova:substring-pos**('Altova', 'to', 1) は 3 を返します。
- **altova:substring-pos**('Altova', 'to', 3) は 3 を返します。
- **altova:substring-pos**('Altova', 'to', 4) は 0 を返します。
- **altova:substring-pos**('Altova-Altova', 'to', 0) は 3 を返します。
- **altova:substring-pos**('Altova-Altova', 'to', 4) は 10 を返します。

trim-string [altova:]

altova:trim-string(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
この関数は xs:string 引数を必要とし、先頭または後続の空白を削除し、トリミングされた xs:string を返します。

サンプル

- **altova:trim-string**(" Hello World ")) は "Hello World" を返します。
- **altova:trim-string**("Hello World ")) は "Hello World" を返します。
- **altova:trim-string**(" Hello World")) は "Hello World" を返します。
- **altova:trim-string**("Hello World")) は "Hello World" を返します。
- **altova:trim-string**("Hello World")) は "Hello World" を返します。

trim-string-left [altova:]

altova:trim-string-left(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
この関数は xs:string 引数を必要とし、先頭または後続の空白を削除し、トリミングされた xs:string を返します。

サンプル

- **altova:trim-string-left**(" Hello World ")) は "Hello World " を返します。
- **altova:trim-string-left**("Hello World ")) は "Hello World " を返します。
- **altova:trim-string-left**(" Hello World")) は "Hello World" を返します。
- **altova:trim-string-left**("Hello World")) は "Hello World" を返します。
- **altova:trim-string-left**("Hello World")) は "Hello World" を返します。

▼ trim-string-right [altova:]

altova:trim-string-right(InputString *as xs:string*) を **xs:string** とする **XP3.1** **XQ3.1**
この関数は xs:string 引数を必要とし、先頭または後続の空白を削除し、トリミングされた xs:string を返します。

▣ *サンプル*

- **altova:trim-string-right**(" Hello World ")) は" Hello World" を返します。
- **altova:trim-string-right**("Hello World ")) は"Hello World" を返します。
- **altova:trim-string-right**(" Hello World")) は" Hello World" を返します。
- **altova:trim-string-right**("Hello World")) は"Hello World" を返します。
- **altova:trim-string-right**("Hello World")) は"Hello World" を返します。

11.1.2.1.9 XPath/XQuery 関数: その他

XPath/XQuery 拡張関数の以下の一般的な目的は現在の MapForce バージョンでサポートされており (i) XSLT コンテキスト内の XPath 式または (ii) Xquery ドキュメント内の XQuery 式内で使用することができます。。

関数の名前指定と言語の適用性に関するメモ

Altova 拡張関数は XPath/XQuery 式で使用することができ、XPath、XQuery、および XSLT 関数の標準ライブラリで使用可能な機能に更なる機能性を与えます。Altova 拡張関数は Altova 拡張関数名前空間、**http://www.altova.com/xslt-extensions** に収められており、**altova:** プレフィックスが、このセクションでは使用されます。製品の今後のバージョンが拡張機能への継続的サポート、または個別の関数の振る舞いは変更する可能性があることご注意してください。Altova 拡張機能へのサポートに関しては、今後のリリースのドキュメントを参照してください。

XPath 関数 (XSLT 内の XPath 式で使用):	**XP1** **XP2** **XP3.1**
XSLT 関数 (XSLT 内の XPath 式で使用):	**XSLT1** **XSLT2** **XSLT3**
XQuery 関数 (XQuery 内の XQuery 式で使用):	**XQ1** **XQ3.1**

▼ decode-string [altova:]

altova:decode-string(Input *as xs:base64Binary*) **as xs:string** **XP3.1** **XQ3.1**
altova:decode-string(Input *as xs:base64Binary*, **Encoding** *as xs:string*) **as xs:string** **XP3.1** **XQ3.1**
指定されたエンコードを使用して送信された base64Binary 入力を文字列にデコードします。エンコードが指定されていない場合 UTF-8 エンコードが使用されます。以下のエンコードがサポートされます: US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4

▣ *サンプル*

- **altova:decode-string**($XML1/MailData/Meta/b64B) は base64Binary 入力を UTF-8 エンコード済み文字列として返します。
- **altova:decode-string**($XML1/MailData/Meta/b64B, "UTF-8") は base64Binary 入力を UTF-8 エンコード済み文字列として返します。
- **altova:decode-string**($XML1/MailData/Meta/b64B, "ISO-8859-1") は base64Binary 入力を ISO-8859-1 エンコード済み文字列として返します。

encode-string [altova:]

altova:encode-string(InputString *as xs:string*) **as xs:base64Binaryinteger** XP3.1 XQ3.1
altova:encode-string(InputString *as xs:string*, **Encoding** *as xs:string*) **as xs:base64Binaryinteger** XP3.1 XQ3.1
与えらえている場合指定されているエンコードを使用して送信された文字列をエンコードします。エンコードが与えられていない場合 UTF-8 エンコードが使用されます。エンコードされた文字列はbase64Binary 文字に変換され、エンコードされたbase64Binary 値が返されます。最初にUTF-8 エンコードがサポートされ、サポートは以下のエンコードに拡張されます: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

サンプル

- **altova:encode-string**(`"Altova"`) はUTF-8 エンコード済み文字列 `"Altova"` の等価である base64Binary を返します。
- **altova:encode-string**(`"Altova", "UTF-8"`) はUTF-8 エンコード済み文字列 `"Altova"` の等価である base64Binary のを返します。

get-temp-folder [altova:]

altova:get-temp-folder() を `xs:string` とする XP2 XQ1 XP3.1 XQ3.1
この関数は引数を必要としません。この関数は現在のユーザーの一時的なフォルダーへのパスを返します。

サンプル

- **altova:get-temp-folder**() は、マシン上で、`xs:string` として `C:\Users\<UserName>\AppData\Local\Temp\` と類似したパスを返します。

generate-guid [altova:]

altova:generate-guid() asxs:string XP2 XQ1 XP3.1 XQ3.1
ユニークな文字列 GUID 文字列を生成します。

サンプル

- **altova:generate-guid**() は例えば `85F971DA-17F3-4E4E-994E-99137873ACCD` を返します。

high-res-timer [altova:]

altova:high-res-timer() asxs:double XP3.1 XQ3.1
秒数でシステムハイレゾリューションタイマーの値を返します。ハイレゾリューションタイマーがシステム内で存在すると、必要とされる場合、高度に正確な時間の計算を有効化します（例えば、アニメーションと正確なコード実行の時間の決定などが例として挙げられます）。この関数は、システムのタイマーに精度を与えます。

サンプル

- **altova:high-res-timer**() は `'1.16766146154566E6'` などを返します。

parse-html [altova:]

altova:parse-html(HTMLText *as xs:string*) **asnode()** XP3.1 XQ3.1
`HTMLText` 引数は、HTML ドキュメントのテキストを含む文字列です。関数は、文字列からHTML ツリーを作成します。提供された文字列は、HTML 要素を含む、または、含まない場合があります。いずれの場合でも、ツリーのルート要素は、**HTML** と名付けられます。提出された文字列内のHTML コードが有効な HTMLであることを確認することが奨励されます。.

サンプル

- **altova:parse-html**("<html><head/><body><h1>Header</h1></body></html>") は提供された文字列からHTML ツリーを作成します。

sleep[altova:]

altova:sleep(**Millisecs** *as xs:integer*) **asempty-sequence**() XP2 XQ1 XP3.1 XQ3.1

Millisecs により与えられるミリ秒で示される期間のための実行を延期します。

サンプル

- **altova:sleep**(1000) は1000 ミリ秒のための実行を延期します。

[[トップ]]

11.1.2.2 その他の拡張関数

Java やC# などのプログラミング言語には、XPath 2.0 / XQuery 関数、またはXSLT 2.0 関数として利用できない関数がいくつかあります。そのような関数の良い例として、Java で利用することのできるsin() やcos() といった数学関数があります。XSLT スタイルシートやXQuery のクエリにてこれらの関数が利用できるのであれば、スタイルシートやクエリの適用範囲を大幅に拡張することができ、スタイルシート作成タスクの負担が大幅に軽減されます。Altova 製品で使用されているAltova エンジン(XSLT 1.0、XSLT 2.0、XQuery 1.0)では、Java 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。 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。

11.1.2.2.1 Java 拡張関数

Java 拡張関数は、XPath またはXQuery 条件式にて使用することができるほか、Java のコンストラクターを呼び出したり、Java の(静的またはインスタンス)メソッドを呼び出すことができます。

Java クラスのフィールドは、引数を持たないメソッドとして扱われます。フィールドは静的またはインスタンスとして存在することができます。フィールドへのアクセス方法については、静的とインスタンスの両方について、以下のサブセクションにて記述されます。

このセクションは以下のサブセクションにより構成されます:

- Java：コンストラクター
- 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

以下の点に注意してください

- Altova デスクトップ製品を使用している場合、Altova アプリケーションは、Java 仮想マシンへのパスを（以下の順序で）読み取ることにより自動的に検知しようとします：(i) Windows レジストリ (ii) **JAVA_HOME** 環境変数。アプリケーションのオプションダイアログ内にカスタムパスを追加することもできます。このエントリは自動的に検知された他の Java VM パス以上の優先順位を有します。
- Altova サーバー製品が Windows マシン上で作動している場合、Java 仮想マシンへのパスは Windows レジストリから最初に読み取られます。成功しない場合、**JAVA_HOME** 環境変数が使用されます。
- Altova サーバー製品を作動する場合、および、Linux または macOS マシン上でサーバー製品を作動する場合、Java 仮想マシンへのパスが **JAVA_HOME** 環境変数内に保管されていることを確認してください。**\bin\server** または **\bin\client** ディレクトリ内の **jvm.dll** ファイルを指している必要があります。

拡張関数のフォーム

XPath/XQuery 条件式における拡張関数では、`prefix:fname()` の形式を取る必要があります。

- `prefix:` 部により拡張関数が Java 関数として認識されます。`java:` から始まる URI のスコープ内の名前空間宣言に拡張関数を関連付けることで Java 関数であるという認識が行われます。名前空間の宣言により、例えば `xmlns:myns="java:java.lang.Math"` という Java クラスが特定されます。名前空間の宣言は、（コロン無しの）`xmlns:myns="java"` という形式で、Java クラスの識別子を拡張関数にある `fname()` 部の左型に配置することでも行うことができます。
- `fname()` 部により、呼び出されている Java メソッドが識別され、メソッドの引数が提供されます（以下の例を参照ください）。`prefix:` 部にて識別された名前空間 URI が Java クラスを識別できなかった場合、Java クラスの識別はクラスの前にくる `fname()` 部にて行うことになり、ピリオドによりクラスから分離されることになります（以下にある2番目の XSLT サンプルを参照）。

メモ 呼び出されるクラスはコンピューターのクラスパス上にある必要があります。

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:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

拡張関数内にあるメソッド名（上の例では `cos()`）は、名前付き Java クラス（上の例では `java.lang.Math`）の public な静的メソッドの名前に一致する必要があります。

XQuery サンプル

以下にXSLT のサンプルに似たXQuery のサンプルを示します:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

ユーザー定義されたJava クラス

独自のJava クラスやメソッドを作成した場合、(i) JAR ファイル(またはclass ファイル)を介してこれらクラスファイルへアクセスしているか、(ii) これら(JAR またはclass)ファイルが、カレントディレクトリ(XSLT やXQuery ドキュメントが存在するディレクトリ)に配置されているかにより、これらクラスの呼び出し方法が変わってきます。これらファイルの特定方法については、ユーザー定義クラスファイル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ご注意してください。

11.1.2.2.1.1 ユーザー定義のクラスファイル

アクセスがクラスファイルを介したものである場合、4つのケースが考えられます:

- クラスファイルはパッケージである。XSLT またはXQuery ファイルがJava パッケージと同じ場所に収められている。(下のサンプルを参照)
- 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/XQuery ファイルがJava パッケージと同じ場所に収められている

以下の例では、`com.altova.extfunc` パッケージにある`Car` クラスの`getVehicleType()` メソッドが呼び出されています。`com.altova.extfunc` パッケージは`JavaProject`という名前のフォルダーに置かれており、XSLT ファイルも同じフォルダーに配置されています。

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

クラスファイルが参照され、XSLT/XQuery ファイルがクラスファイルと同じフォルダーに収められている

下のサンプルでは Car クラスの getVehicleType() メソッドが呼び出されます。以下を述べることができます: (i) Car クラスは次のフォルダー内にあります: JavaProject/com/altova/extfunc 。(ii) このフォルダーは下のサンプルの現在のフォルダー内に存在します。XSLT ファイルもフォルダー JavaProject/com/altova/extfunc 内にあります。

```
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

クラスファイルがパッケージされXSLT/XQuery ファイルがランダムな場所に収められている

以下の例では、com.altova.extfunc パッケージにある Car クラスの getVehicleColor() メソッドが呼び出されています。com.altova.extfunc パッケージは JavaProject という名前のフォルダーに置かれており、XSLT ファイルが任意の場所に配置されています。この場合、以下のような構文でパッケージの場所をクエリ文字列として URI 内にて指定する必要があります:

```
java:classname[?path=uri-of-package]
```

ここで、

java: によりユーザー定義の Java 関数が呼ばれていることを表します。
uri-of-package は Java パッケージの URI です。
classname は目的のメソッドが含まれているクラス名です。

クラスは名前空間 URI により特定され、名前空間がメソッド呼び出しのプレフィックスで使用されます。以下の例ではカレントディレクトリ以外にあるクラスファイルへのアクセスを行うことができます。

```
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:car="java:com.altova.extfunc.Car?path=file:///C:/JavaProject/" >
```

```
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

クラスファイルがパッケージではなく、XSLT/XQuery ファイルがランダムな場所に収められている

以下の例では、com.altova.extfunc パッケージにある Car クラスの getCarColor() メソッドが呼び出されています。com.altova.extfunc パッケージは JavaProject という名前のフォルダーに置かれており、XSLT ファイルが任意の場所に配置されています。以下のような構文で、クラスファイルの場所をクエリ文字列として URI 内にて指定する必要があります:

```
java:classname[?path=uri-of-classfile]
```

ここで

java: によりユーザー定義の Java 関数が呼ばれていることを表します。
uri-of-classfile は Java パッケージの URI です。
classname は目的のメソッドが含まれているクラス名です。

クラスは名前空間 URI により特定され、名前空間はメソッド呼び出しのプレフィックスで使用されます。以下の例ではカレントディレクトリ以外にあるクラスファイルへのアクセスを行うことができます。

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java:Car?path=file:///C:/JavaProject/com/altova/extfunc/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

メモ パスが外部関数により与えられている場合、ClassLoader によりパスが追加されます。

11.1.2.2.1.2 ユーザー定義の JAR ファイル

JAR ファイル経由でアクセスが行われた場合、以下の構文により JAR ファイルの URI を指定する必要があります:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

クラスの識別を行う名前空間 URI のプレフィックスを使用してメソッドが呼び出されます: classNS:method()

上の例に対する説明は以下のとおりです:

java: 関数が呼び出されていることを表します。
classname がユーザー定義されたクラスの名前になります。
? はクラス名とパスを分離するために使用されます。
path=jar: により、JAR ファイルへのパスが与えられていることを示します。
uri-of-jarfile は JAR ファイルの URI となります。
!/ は、終了を表すデリミタとなります。
classNS:method() により、メソッドの呼び出しが行われます。

その他にも、メソッド名とともにクラス名を与えることができます。構文の例を以下に示します:

```
              xmlns:ns1="java:docx.layout.pages?
path=jar:file:///c:/projects/docs/docx.jar!/"
              ns1:main()

              xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
              ns2:docx.layout.pages.main()
```

以下に JAR ファイルを使った Java 拡張関数を呼び出す XSLT サンプルを記します:

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
       <xsl:variable name="myCar" select="car:Car1.new('red')" />
       <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

メモ: 拡張関数によりパスが与えられている場合、ClassLoader にパスが追加されます。

11.1.2.2.1.3 Java：コンストラクター

拡張関数を使用することで Java コンストラクターを呼び出すことができます。new() により全てのコンストラクターを呼び出すことができます。

Java コンストラクターの呼び出し結果を、黙示的に XPath/XQuery データ型へ変換できる場合、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" />
```

11.1.2.2.1.4 Java：静的メソッドと静的フィールド

静的メソッドは、Java 名ならびにメソッドの引数により直接呼び出すことができます。`E` や `PI` といった定数の静的フィールド（引数を持たないメソッド）は、引数を指定することなくアクセスすることができます。

XSLT の例

静的メソッドならびにフィールドを呼び出す例を以下に示します:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

上の拡張関数は `prefix:fname()` という形式を使用していることに注意してください。3つの例にあるプレフィックスは全て jMath: となっており、このプレフィックスは、`java:java.lang.Math` という名前空間 URI に関連付けられています。名前空間 URI は java: で開始しなければなりません。上の例では、クラス名（`java.lang.Math`）を含むように拡張されています。拡張関数の fname() 部は（`java.lang.Math` のような）public クラスにマッチする必要があり、その後 public な静的メソッドが引数とともに続くか（例: `cos(3.14)`）、public な静的フィールドが続きます（例: PI()）。

上の例では、クラス名が名前空間 URI に含まれています。クラス名が名前空間 URI に含まれていない場合、以下の例にあるように、拡張関数の `fname()` 部にて追加する必要があります:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery の例

XQuery における似たようなサンプルを以下に示します:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

11.1.2.2.1.5 Java：インスタンスメソッドとインスタンスフィールド

メソッド呼び出しの第一引数としてパースされる Java オブジェクトが、インスタンスメソッドには与えられています。このような Java オブジェクトは通常、拡張関数を使うことで作成される（例: コンストラクターの呼び出し）か、スタイルシートパラメーター/変数により作成されます。以下に XSLT サンプルを示します:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
```

```
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                  date="{date:toString($CurrentDate)}"
                  type="{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

上の例では、ノード `enrollment/@type` の値が以下のように作成されます:

1. `java.util.Date` クラスに対するオブジェクトがコンストラクター(`date:new()` コンストラクター)とともに作成されます。
2. `jlang.Object.getClass` メソッドの引数として Java オブジェクトがパースされます。
3. `getClass` メソッドにより得られたオブジェクトが `jlang.Object.toString` メソッドの引数としてパースされます。

結果(`@type` の値)は `java.util.Date` を持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。

11.1.2.2.1.6 データ型:XPath/XQuery から Java へ

XPath/XQuery 条件式内部から Java 関数が呼び出された場合、複数ある同名の Java クラスのうち、どのクラスが呼び出されたのか決定するのに、関数へ渡される引数のデータ型が重要になります。

Java では、以下のルールが適用されます:

- 同名の Java メソッドが2つ以上あり、それぞれが違う数の引数を受け取る場合、呼び出しに使用されている引数の数に一番マッチするメソッドが選択されます。
- XPath/XQuery の文字列、数値、boolean データ型は、黙示的に対応する Java データ型へ変換されます(以下のリストを参照)。与えられた XPath/XQuery 型が2つ以上の Java 型へ変換できる場合(例: `xs:integer`)、選択されたメソッドにて宣言されている Java 型が使用されます。例えば、呼び出された Java メソッドが `fx(decimal)` で、与えられた XPath/XQuery データ型が `xs:integer` の場合、`xs:integer` が Java の `decimal` データ型へ変換されます。

以下のテーブルに、XPath/XQuery の文字列、数値、boolean 型から Java データ型への黙示的な変換リストを示します。

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (プリミティブ型), `java.lang.Boolean`
`xs:integer`	`int`, `long`, `short`, `byte`, `float`, `double`, ならびに `java.lang.Integer` のようなこれらのラッパークラス
`xs:float`	`float` (プリミティブ型), `java.lang.Float`, `double` (プリミティブ型)
`xs:double`	`double` (プリミティブ型), `java.lang.Double`

xs:decimal	float (プリミティブ型), java.lang.Float, double(プリミティブ型), java.lang.Double

上のリストにあるXML スキーマデータ型(ならびにXPath やXQuery で使用されているデータ型)のサブタイプも、対応する祖先のサブタイプとしてJava のデータ型へ変換されます。

場合によっては、与えられた情報から正しいJava メソッドを選択することができない場合もあります。例えば、以下のような場合を考えてみましょう:

- 与えられた引数が10 という値を持ったxs:untypedAtomic 型で、mymethod(float) メソッドへ渡されるのを意図している。
- しかし、そのクラスには別のデータ型を取るmymethod(double) というメソッドも存在する。
- メソッド名が同じで、与えられた型(xs:untypedAtomic) もfloat とdouble の両方に変換することができるため、xs:untypedAtomic がfloat ではなくdouble に変換される可能性もある。
- 結果として、意図したメソッドは選択されず、予期しない動作結果が招く可能性がある。この問題を回避するには、意図したメソッドを使用するユーザー定義のメソッドを別の名前で新たに作成する必要があります。

上のリストでカバーされていない型(例: xs:date)は変換されず、エラーとなります。しかし場合によっては、Java コンストラクターを使用して、目的のJava データ型を作成することが可能だということも留意してください。

11.1.2.2.1.7 データ型:Java から XPath/XQuery へ

Java メソッドにより値が返され、値のデータ型が文字列、数値、またはboolean 型の場合、対応するXPath/XQuery 型への変換が行われます。例えば、Java のjava.lang.Boolean やboolean データ型はxsd:boolean へ変換されます。

関数から返された一次元配列は、シーケンス(sequence) に展開されます。2次元以上の配列は変換されることが無いため、ラップして使用するべきでしょう。

Java オブジェクトや文字列、数値、boolean 以外のデータ型がラップされて返された場合、最初に(例えばtoString といった)Java メソッドを使用してJava オブジェクトを文字列へ変換することで、目的のXPath/XQuery 型への変換を行います。XPath/XQuery では、文字列を目的となる型のレキシカルフォーマットへ変換し、目的の型への変換を(例えばcast as 式を使用することで)行うことができます。

11.1.2.2.2 .NET 拡張関数

.NET プラットフォームにて作業を行なっている場合、.NET 言語(例えばC#)で記述された拡張関数を使用することができます。.NET 拡張関数はXPath やXQuery 条件式内部から使用することができ、.NET クラス内部にあるコンストラクターや(static またはインスタンス変数)プロパティを呼び出すことができます。

get_PropertyName() 構文を使用することにより.NET クラスのプロパティを呼び出すことができます。

このセクションは、以下のサブセクションにより構成されています:

- .NET コンストラクター
- .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新数の数値)。
`from`	ロードするアセンブリ(DLL)の場所を特定する URI。URI が相対パスの場合、XSLT や XQuery ドキュメントに対して相対的になります。このパラメーターが指定された場合、その他のパラメーターが無視されます。
`partialname`	アセンブリ名の一部。`Assembly.LoadWith.PartialName()` へ渡され、アセンブリへのロードが試みられます。`partialname` が指定された場合、その他のパラメーターが無視されます。
`loc`	例えば `en-US` というロケール。デフォルトは `neutral` です。

アセンブリが DLL からロードされる場合、`from` パラメータが使用して、`sn` パラメーターは使用しないでください。アセンブリがグローバルアセンブリキャッシュ(GAC)からロードされる場合、`sn` パラメータを使用して `from` パラメーターは使用しないでください。

最初のパラメーターの前に疑問符(?)を挿入し、パラメーター同士はセミコロンで分離する必要があります。パラメーター名へ値を受け渡すには、統合符号(=)を使用します(以下の例を参照ください)。

名前空間宣言の例

XSLT において、システムクラス `System.Environment` を特定する名前空間宣言の例を以下に示します:

```
xmlns:myns="clitype:System.Environment"
```

XSLT において、ロードするクラスを `Trade.Forward.Scrip` として特定する名前空間宣言の例を以下に示します:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

XQuery において、システムクラス `MyManagedDLL.testClass` を特定する名前空間宣言の例を以下に示します。2つのケースが考えられます:

1. アセンブリが GAC からロードされた場合:
   ```
   declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
   ```

```
ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. アセンブリがDLL からロードされた場合(完全参照と一部の参照):

```
declare namespace cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
Projects/extFunctions/MyManagedDLL.dll;

declare namespace cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

XSLT の例

システムクラス System.Math 内の関数を呼び出すための完全な XSLT の例を以下に示します:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes" />
   <xsl:template match="/">
      <math xmlns:math="clitype:System.Math">
         <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
         <pi><xsl:value-of select="math:PI()"/></pi>
         <e><xsl:value-of select="math:E()"/></e>
         <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
      </math>
   </xsl:template>
</xsl:stylesheet>
```

math 要素にある名前空間宣言により、math: プレフィックスと clitype:System.Math URI が関連付けられます。URI の最初にある clitype: により、それ以降の記述がシステムクラスまたはロードされたクラスを特定するものであることが示されます。XPath 条件式にある math: プレフィックスにより、拡張関数が URI (そしてクラス) System.Math に関連付けられます。拡張関数により、System.Math クラス内のメソッドが特定され、必要な場所に引数が与えられます。

XQuery の例

上の XSLT に対する例と同様の XQuery 例を以下に示します:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

上の XSLT と同様に、名前空間宣言により .NET クラス(この場合はシステムクラス) が特定されます。XQuery 式により呼び出されるメソッドが特定され、引数が与えられます。

11.1.2.2.2.1 .NET コンストラクター

拡張関数を使用することで、.NET コンストラクターを呼び出すことができます。new() により全てのコンストラクターを呼び出すことができます。クラス内に2つ以上のコンストラクターがある場合、与えられた引数の数が最もマッチするコンストラクターが選択されます。与えられた引数に対してマッチするコンストラクターが見つからない場合、"No constructor found'" エラーが返されます。

XPath/XQuery データ型を返すコンストラクター

.NET コンストラクター呼び出しの結果が XPath/XQuery データ型へ黙示的に変換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` 型を持つ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" />
```

11.1.2.2.2.2 .NET: 静的メソッドと静的フィールド

メソッド名と引数を与えることで、静的メソッドを直接呼び出すことができます。呼び出しに使用される名前は、クラス内にある public static メソッドと完全に一致する必要があります。関数の呼び出しに使用されたメソッド名と引数の数にマッチするものがクラス内に複数ある場合、与えられた引数が評価され、最もマッチするものが選択されます。マッチする結果が得られない場合、エラーが返されます。

メモ: .NET クラス内にあるフィールドは、引数を持たないメソッドとしてみなされます。プロパティは `get_PropertyName()` 構文により呼び出されます。

例

1つの引数とともにメソッド (`System.Math.Sin(arg)`) を呼び出す XSLT サンプルを以下に示します:

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

(引数なしのメソッドとしてみなされる) フィールド (`System.Double.MaxValue()`) を呼び出す XSLT サンプルを以下に示します:

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

(`get_PropertyName()` 構文を使って) プロパティ (`System.String()`) を呼び出す XSLT サンプルを以下に示します:

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

1つの引数とともにメソッド (`System.Math.Sin(arg)`) を呼び出す XQuery サンプルを以下に示します:

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

11.1.2.2.2.3 .NET: インスタンスメソッドとインスタンスフィールド

インスタンスメソッドには、メソッド呼び出しの第一引数として .NET オブジェクトが渡されます。通常この.NET オブジェクトは、拡張関数(例えばコンストラクター呼び出し)またはスタイルシートパラメーター/変数により作成されます。以下にXSLT の例を示します:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

上の例では、System.DateTime コンストラクター(`new(2008, 4, 29)`) が、System.DateTime 型の.NET オブジェクトの作成に使用されます。このオブジェクトは、最初に`releasedate` 変数の値として、次に`System.DateTime.ToString()` メソッドの引数として作成されます。`System.DateTime.ToString()` インスタンスメソッドは、`System.DateTime` コンストラクターの (new(2008, 4, 29)) における引数として2度呼び出されます。これらインスタンスにおいて `releasedate` 変数が.NET オブジェクトを取得するのに使用されます。

インスタンスメソッドとインスタンスフィールド

インスタンスメソッドとインスタンスフィールドの違いは理論的なものです。インスタンスメソッドでは.NET オブジェクトが直接引数に渡され、インスタンスフィールドではパラメーターや変数が(.NET オブジェクトそのものを含めることはできるものの)代わりに渡されます。例えば上の例では、`releasedate` 変数に.NET オブジェクトが含まれており、この変数が2番目の date 要素コンストラクターにて `ToString()` の引数として渡されます。そのため、最初の date 要素にある `ToString()` インスタンスがインスタンスメソッドであるのに対し、2番目はインスタンスフィールドとしてみなされます。両方のインスタンスで求められる結果は等価です。

11.1.2.2.2.4 データ型：XPath/XQuery から .NET へ

.NET 拡張関数がXPath/XQuery 条件式内部で使用された場合、複数ある.NET メソッドのうち、どれが呼び出されたの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1.0ある`xs:untypedAtomic` 値で、`mymethod(float)` メソッドへ渡されるのを意図している。
- しかし、そのクラスには別のデータ型をとる`mymethod(double)` というメソッドも存在する。
- メソッド名が同じで、与えられた型（`xs:untypedAtomic`）も`float` と`double` の両方に変換することができるため、`xs:untypedAtomic` が`float` ではなく`double` に変換される可能性もある。
- 結果として、意図したメソッドは選択されず、予期しない動作結果が招く可能性がある。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。

11.1.2.2.2.5 データ型：.NET から XPath/XQuery へ

.NET メソッドにより値が返される際に値のデータ型が文字列、数値、またはboolean 型の場合、対応するXPath/XQuery 型への変換が行われます。例えば、.NET の`decimal`データ型は`xsd:decimal` へ変換されます。

.NET オブジェクトや文字列、数値、boolean 以外のデータ型が返された場合、最初に(例えば System.DateTime.ToString() といった).NET メソッドを使用して.NET オブジェクトを文字列へ変換します。XPath/XQuery では、文字列を目的となる型のレキシカルフォーマットへ変換し、目的の型への変換を(例えば cast as 式を使用することで)行うことができます。

11.1.2.2.3 XSLT に対する MSXSL スクリプト

<msxsl:script> 要素にはユーザー定義の関数や変数が含まれており、XSLT スタイルシート内の XPath 条件式内部から呼び出しを行うことができます。<msxsl:script> はトップレベル要素で、<xsl:stylesheet> または <xsl:transform> の子要素である必要があります。

<msxsl:script> 要素は urn:schemas-microsoft-com:xslt 名前空間内に存在する必要があります(以下を参照ください)。

スクリプト言語と名前空間

ブロック内で使用されるスクリプト言語は <msxsl:script> 要素の language 属性にて指定され、XPath 条件式における関数の呼び出しに対して使用される名前空間は implements-prefix 属性により特定されます(以下を参照)。

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

<msxsl:script> 要素は Windows Scripting Runtime を使ったやりとりを行うため、お使いのコンピューターにインストールされた言語だけしか <msxsl:script> 要素では使用することができません。MSXSL スクリプトを使用するには .NET Framework 2.0 以上のプラットフォームをインストールする必要があります。結果として <msxsl:script> 言語から .NET スクリプト言語を使用することができます。

HTML の <script> 要素における language 属性と同じ値が language 属性では受理されます。language 属性が指定されていない場合、Microsoft JScript がデフォルトとして想定されます。

implements-prefix 属性には名前空間スコープ内で宣言されたプレフィックスが与えられます。通常この名前空間は関数ライブラリのために予約されたユーザーの名前空間となります。<msxsl:script>要素内で定義された全ての関数ならびに変数は、implements-prefix 属性にて指定されたプレフィックスで特定される名前空間に収められます。XPath 条件式内部から関数が呼ばれる場合、完全修飾関数名が同じ名前空間内に関数として定義されていなければなりません。

サンプル

<msxsl:script> 要素内で定義された関数を使用する XSLT スタイルシートの例を以下に示します:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input:A currency value: the wholesale price
    ' Returns:The retail price: the input value plus 20% margin,
```

```
    ' rounded to the nearest cent
    dim a as integer  = 13
    Function AddMargin(WholesalePrice) as integer
      AddMargin = WholesalePrice * 1.2 + a
    End Function
  ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

データ型

スクリプトブロックとのやりとりに使用される パラメーターの値はXPath データ型に限定されます。スクリプトブロック内にある関数にてやりとりされるデータや変数に、この制限はありません。

アセンブリ

msxsl:assembly 要素を使用することで、アセンブリをスクリプト内部へインポートすることができます。アセンブリは名前やURI により特定されます。アセンブリのインポートは、コンパイル時に行われます。以下に msxsl:assembly 要素の簡単な使用例を示します:

```
<msxsl:script>
       <msxsl:assembly name="myAssembly.assemblyName" />
       <msxsl:assembly href="pathToAssembly" />

       ...

</msxsl:script>
```

アセンブリ名は、以下のような完全な名前でも::

```
"system.Math, Version=3.1.4500.1 Culture=neutral PublicKeyToken=a46b3f648229c514"
```

`"myAssembly.Draw"` のような短い名前でも指定できます。

名前空間

msxsl:using 要素により名前空間の宣言を行うことができます。これにより、スクリプト内において名前空間無しでアセンブリクラスを使用することができ、タイピングの手間を軽減することができます。以下に `msxsl:using` 要素の簡単な使用例を示します:

```
<msxsl:script>
       <msxsl:using namespace="myAssemblyNS.NamespaceName" />

       ...
```

```
</msxsl:script>
```

`namespace` 属性の値は名前空間の名前となります。

11.2 技術データ

このセクションは、ソフトウェアの技術面に関する役に立つ背景情報を含んでいます。以下のように整理されています:
OS とメモリ要件Altova XML バリデーターAltova XSLT とXQuery エンジンUnicode のサポートインターネットの使用

11.2.1 OS とメモリ要件

オペレーティングシステム

Altova ソフトウェア アプリケーションは、以下のプラットフォームでご使用いただけます:
プラットフォーム更新済みのW indows 7 SP1、W indows 8、W indows 10プラットフォーム更新済みのW indows Server 2008 R2 SP1 または以降 メモリ
ソフトウェアがC++ で書かれているため、Java Runtime Environment をダウンロードする必要はなく、Java ベースのアプリケーションに比べ、通常少ないメモリを必要とします。しかしながら、各ドキュメントは完全に解析するため、また、ビューと編集の速度を向上するためにメモリにダウンロードされます。メモリの要件は、ドキュメントのサイズを増やします。

メモリ要件は、制限のない「元に戻す」履歴により影響を受けます。大きなドキュメントの大きなセクションの切り取り、貼り付け操作を繰り返し行うと、使用できるメモリがすぐに消費されます。

11.2.2 Altova XML バリデーター

XML ドキュメントを開くと、アプリケーションは、内蔵のXML バリデーターを使用して、指定されている場合、スキーマに対して整形式をチェックし、ツリーとインフォセットを作成します。XML バリデーターは、ドキュメントを編集する際にインテリジェントな編集ヘルプを提供し、発生する検証エラーを表示するために使用されます。

内蔵のXML バリデーターは、W 3C のXML スキーマ1.0 と1.1 仕様の最終勧告を実装しています。. New developments recommended by the W 3C XML スキーマ作業グループにより勧告される新しい項目は、XML バリデーターに継続的に組み込まれるため、Altova 製品は最高水準の開発環境を届けることができます。

11.2.3 Altova XSLT と XQuery エンジン

Altova 製品 は、Altova XSLT 1.0、2.0、および3.0 エンジンとAltova XQuery 1.0 と3.1 エンジンを使用しています。各エンジンのためのドキュメントと実装に固有の振る舞いに関しては、製品で使用されるエンジンの各ドキュメントの付属書(エンジン情報) で確認することができます。

メモ Altova MapForce は、XSLT 1.0、2.0 およびXQuery 1.0 エンジンを使用したコードを生成します。

11.2.4 Unicode のサポート

Altova XML 製品は、Unicode を完全にサポートします。XML ドキュメントを編集するには、ドキュメント内で使用されているUnicode 文字をサポートするフォントが必要です。

フォントの多くは、Unicode 範囲全体の特定のサブセットを含む場合があり、このため、通常は対応する表記システムをターゲットとします。テキストの一部が、文字化けして表示された場合、理由としては、選択されたフォントが必要とする字形を含まない場合があげられます。です

から、特に、異なる言語、または、異なる言語システムのXMLドキュメントを編集する場合、範囲全体をカバーするフォントを使用することが役にたちます。典型的なUnicode フォントは、W indows PC のArial Unicode MS で確認することができます。

アプリケーションフォルダーの/Examples フォルダー内で、異なる言語システムで表記された次の文章を含むUnicodeUTF-8.html というXHTML ファイルを確認してください:
When the world wants to talk, it speaks Unicode Wenn die Welt miteinander spricht, spricht sie Unicode
世界的に話すなら、Unicode です。) XHTML ファイルを開くと、Unicode の可能性を確認することができ、使用中のPC の使用することのできるフォントによりサポートされている表記システムが表示されます。

11.2.5 インターネットの使用

Altova アプリケーションは、次の状況でインターネット 接続を開始します:

- 登録ダイアログ(「**ヘルプ | ソフトウェアのライセンス認証**」)内の「評価キーコードをリクエスト」をクリックした場合、登録ダイアログボックス内の3つのフィールドが通常のhttp (ポート 80)接続を使用し、サーバーに転送され、無料の評価キーが顧客に通常のSMTP 電子メールを使用して送り返されます。
- Altova 製品の一部では、インターネットからファイルを開くことができます(「**ファイル | 開く | URL に切り替える**」)。この場合、ドキュメントは、次のプロトコルメソッドと接続の1つを使用して取得されます: HTTP (通常、ポート 80)、FTP (通常、ポート 20/21)、HTTPS (通常、ポート 443)。HTTP サーバーをポート 8080 で作動することもできます(URL ダイアログ内で、サーバー名とコロンの後にポートを指定します)。
- XML スキーマ、または、DTD を参照するXMLドキュメント、と、URL により指定されているドキュメントを開くと、参照されているスキーマドキュメントは、HTTP 接続(ポート 80) またはURL により指定されている他のプロトコール(上のポイント 2 参照)により抽出されます。XML ファイルが検証されている場合、スキーマドキュメントも抽出されます(オプションダイアログのファイルタブ内の(「**ツール | オプション**」))。アプリケーションに命令している場合、ドキュメントが開かれると検証が自動的に行われる場合もあります。
- W SDL とSOAP を使用するAltova アプリケーションでは、W eb サービスを使用する接続は、W SDLドキュメントにより定義されています。
- XMLSpy 内で、「**電子メールで送信**」コマンドを使用する場合、(「**ファイル | 電子メールで送信**」) 現在選択されている範囲、または、ファイルは、ユーザーのマシンにインストールされているMAPI コンプライアント電子メールプログラムにより送信されます。
- ソフトウェアの荒いセンス認証 とLiveUpdate の一部として、Altova ソフトウェア使用許諾書内で更に詳しい説明を確認することができます。

11.3 ライセンス情報

このセクションには以下の内容が含まれています:

- ソフトウェアの配布に関する情報
- ソフトウェアのアクティベーションとライセンスの計測
- ソフトウェアの使用に関する使用許諾契約書

本製品を使用する前に、上記の情報をよくお読みください。ソフトウェアのインストール時に上記のすべての条件に同意したとみなされ、お客様は上記の条件に拘束されることを同意したとみなされます。

Altova ライセンスの内容を確認するには、Altova Web サイト の Altova 法的な情報のページ に移動してください。

11.3.1 電子的なソフトウェアの配布

この製品は電子的なソフトウェアの配布により利用することが可能で、この配布方法により、以下のユニークなメリットがあります:

- 購入を決定する前に、無料でソフトウェアを試用することができます。*(Note: Altova MobileTogether Designer に対してライセンスを無料で割り当てることができます)。*
- Once ソフトウェアの購入を決定した際には、Altova Web サイト にて注文を行います。すぐにライセンス登録された製品の使用を開始することができます。
- オンラインにて注文を行うと、常に最新のソフトウェアをご利用いただけます。
- 製品パッケージには包括的なヘルプシステムが画面上に表示されます。最新バージョンのユーザーマニュアルは https://www.altova.com/ja/ 上にあり、(i) HTML フォーマットによる閲覧、ならびに(ii) PDF フォーマットのダウンロードと印刷に対応しております。

30 日間の評価期間

この製品をダウンロードした後は、最大で30 日の間無料で製品の評価を行うことができます。20 日間を超えた頃から、製品がライセンス登録されていないことがソフトウェアにより表示されます。このメッセージはアプリケーションが起動されるたびに表示され、30 日間を超えてプログラムを使用するには、キーコードを含むライセンスファイルから提供される製品のライセンスを購入します。ライセンスファイルを製品のソフトウェアアクティベーションダイアログにアップロードして、製品をアンロックします。

https://shop.altova.com/ でライセンスを購入することができます

組織内でソフトウェアの評価を行う

評価版のソフトウェアを組織内のネットワークにて配布したい場合、またはインターネットに接続されていないコンピューターにてソフトウェアを使用する場合、どのような状態でも改変されていないことを条件に、セットアッププログラムおよびの配布を行うことが可能です。ソフトウェアインストーラーへアクセスした人は、例外なく30 日間の評価ライセンスキーコードをリクエストして、試用期間が経過した後は、製品を使い続けるためにライセンスの購入を行う必要があります。

11.3.2 ソフトウェアのアクティベーションとライセンスの計測

Altova のソフトウェアアクティベーションの一部として、ソフトウェアにより内部ネットワークまたはインターネットへの接続を行い、インストール時、登録時、Altova により使用されるライセンスサーバーの更新やライセンスの正当性を検証することで、ソフトウェアの不正な使用を防ぎ、顧客サービスを向上するため、ライセンスに関する情報を送信することがあります。アクティベーションにより、オペレーティングシステムや IP アドレス、

日付/時刻、ソフトウェアのバージョン、コンピュータの名前などのライセンスに関する情報が、お使いのコンピューターとAltova ライセンスサーバー間にてやり取りされます。

お使いのAltova 製品にはライセンス計測モジュールが内蔵されており、エンドユーザー使用許諾契約書の意図しない違反を防ぎます。お使いの製品はシングルユーザーまたはマルチユーザーとしてインストールされており、ライセンス計測モジュールにより、ライセンスされている数を超えたユーザーが同時に製品を使用することが無いことが保証されます。

このライセンス計測技術により、ローカルエリア接続（LAN）において、別々のコンピューター間で動作しているアプリケーションインスタンス間の通信が行われます。

シングルライセンス

ライセンス計測プロセスの一部としてアプリケーションが起動すると、ソフトウェアにより短いデータグラムがブロードキャストにより送信され、同一のネットワークセグメントにある他のコンピューターにてプログラムが動作していないかのチェックが行われます。応答が無い場合は、アプリケーションの他インスタンスから送信される信号に応えるため、ポートが開かれます。

マルチユーザーライセンス

同一のLAN 内にて2つ以上のアプリケーションインスタンスが使用された場合、スタートアップ時に、これらインスタンス間において通信が行われます。これらのインスタンス間にてキーコードのやりとりが行われ、購入された数のライセンスを超えてインスタンスが起動しないように保証することができます。このようなライセンス計測システムはUNIX やデータベース開発ツールにて広く使用されているもので、Altova ユーザーはリーズナブルな価格にて同時使用マルチユーザーライセンスを購入することができます。

弊社はアプリケーションのデザインも行っており、少数の小さなネットワークパケットを送信することで、ネットワークに対する負荷を最小限に抑えております。Altova により使用される2799番 TCP/IP ポートはIANA により公式登録されており（詳細は（ IANA W eb サイト (http://www.iana.org/) を参照ください）、弊社のライセンス計測モジュールも既にテストされたものです。

ファイヤーウォールを使用している場合、2799番ポートにてAltova 製品が動作しているコンピューター同士が通信しているのに気づかれるかも知れません。その他の手段によりライセンス使用許諾書の内容が守られることを保証できる限り、組織間の異なるグループにおいてこのようなトラフィックをブロックすることは勿論可能です。

証明書に関するメモ

Altova アプリケーションはHTTPS を介してAltova ライセンスサーバー（`link.altova.com`）に通信します。この通信のためにAltova は登録済みのSSL 証明書を使用します。（例えば、社内IT 部署または外部エージェンシーにより）この証明書が置き換えられている場合、使用中のAltova アプリケーションは接続が安全でないことを警告します。Altova アプリケーションを開始するために代替の証明書を使用することができますが、自己責任で行ってください。*安全ではない接続* の警告メッセージが表示されると、証明書の発行元を確認して（ Altova 証明書の代替証明書の使用の継続または停止を決定することができる）社内IT チームと相談してください。

（例えば、クライアントマシンへの、または、クライアントマシンへの通信を監視するために）自身の証明書の使用が必要な場合 Altova の無料管理ソフトウェアである Altova LicenseServer を使用中のネットワークにインストールすることが奨励されます。このセットアップでは、Altova LicenseServer はAltova との通信のためにAltova 証明書の使用を許可しつつクライアントマシンが所属機関の証明書の使用を継続することができます。

11.3.3 エンドユーザー使用許諾契約書

- Altova エンドユーザー使用許諾契約書: http://www.altova.com/ja/legal/eula
- Altova プライバシーポリシー: http://www.altova.com/ja/privacy

12 用語

このセクションにはMapForce およびMapForce に関連する製品に固有のキーワードのリストが含まれています。

コンポーネント

MapForce 内では、「コンポーネント」という用語はデータの構造(スキーマ)またはデータがどのように変換(関数)されるかを視覚的に表します。コンポーネントはすべてのマッピングを構築する中心的な役割を果たします。マッピングエリアでは、コンポーネントは正方形の枠で表示されます。以下は、MapForce コンポーネントの例です:

- 定数
- フィルター
- 条件
- 関数コンポーネント
- EDI ドキュメント (UN/EDIFACT、ANSI X12、HL7)
- Excel 2007+ ファイル
- 単純型 入力コンポーネント
- 単純型 出力コンポーネント
- スキーマおよびDTD

接続

接続とは2つのコネクタ間に描くことができる線です。接続を描くことにより、MapForce にデータを特定の方法で変換するように命令します。(例: XML ドキュメントからデータを読み込み、他のXML ドキュメントに書き込みます)。

コネクタ

コンポーネント の左右に表示される小さな三角形です。コンポーネントの左に表示されるコネクタはそのコンポーネントのエントリポイントを表します。コンポーネントの右に表示されるコネクタはそのコンポーネントからのデータの終了ポイントを表します。

資格情報

() 認証データを異なるマッピング実行環境で安全なポータブル化する方法を資格情報オブジェクトは提供します。資格情報は基本的なHTTP 認証を必要とするマッピング内で役に立ちます。MapForce とFlowForce Server 内で資格情報を定義することができます。MapForce 内で資格情報が定義されている場合、マッピングデプロイされるようにFlowForce Server にデプロイすることができます。

固定長 ファイル(FLF)

データが習慣的に固定値を持つフィールドに分割される共通のテキストフォーマット。(例: 行の最初の5 文字がトランザクションID を表し、次の20 文字がトランザクションの詳細を表すなど)。

FlexText

FlexText とはMapForce によりサポートされる他のフォーマットに変換するには複雑な非標準またはレガシーテキストファイルからのデータを変換するMapForce Enterprise Edition 内のモジュールです。

グローバルリソース

Altova グローバルリソースはファイル、フォルダー、またはデータベースへのポータブルなレファレンスです。グローバルリソースとして保管されると、パスとデータベース接続の詳細は再利用できるようになりAltova アプリケーション全体で使用可能になります。例えば、複数のAltova デスクトップアプリケーション内で同じファイルを頻繁に開く必要がある場合、グローバルリソースとして定義すると便利な場合があります。このようにすると、「ファイルを開く」ダイアログボックスから対応するグローバルリソースを開くことができるため、ファイルパスを覚えている必要がありません。こ

の方法はファイル パスが変更されると、パスの一部を変更するのみという利点があります。

グローバルリソースの一般的な使用法は、データベース接続を一度定義し、グローバルリソースをサポートする全てのAltova アプリケーションで再利用することです。例えば、MapForce マッピングがデザインされるマシンでデータベース接続を作成し、MapForce Server がマッピングを実行するマシン上で同じ接続を再利用することができます（これは、一部の場合、両方のマシンに同じデータベースクライアントソフトウェアがインストールされている場合があります）。

任意で、（「構成」として既知の）同じグローバルリソースの複数のバリエーションを作成することができます。これによりファイル、またはフォルダーパス（またはデータベース）を必要に応じて変更することができます。例えば、「開発」と「生産」の構成を持つデータベースリソースを作成することができます。

次のAltova デスクトップアプリケーションからグローバルリソースを作成することができます: Altova Authentic、DatabaseSpy、MobileTogether Designer、MapForce、StyleVision、およびXMLSpy。サーバー製品では、次のAltova サーバーアプリケーションでグローバルリソースを作成することができます: FlowForce Server、MapForce Server、RaptorXML Server、RaptorXML+XBRL Server。

入力コンポーネント

入力コンポーネントは、単純型の値をマッピングにパスすることができるMapForce コンポーネントです。入力コンポーネントは通常ファイル名またはその他の文字列の値をランタイムにマッピングにパスするために使用されます。入力コンポーネントとソースコンポーネントを混同しないようにご注意してください。

ジョインコンポーネント

ジョインコンポーネントは、カスタムに定義付けられた条件をベースにして2つの構造をジョインすることを有効化するMapForce コンポーネントです。条件を満たすアイテムのアソシエーション（ジョインされたセット）を返します。ジョインは、（ID などの）共通のフィールドを共有する2つの構造からのデータを結合する際にとても役に立ちます。

MapForce

MapForce は、プログラムコードを作成することなく視覚的なドラッグアンドドロップを使用するグラフィカルなユーザーインターフェイスを用いデータを異なるデータ間、スキーマ間で変換するW indows ベースの多目的のIDE（統合された開発環境 統合開発環境）です。また、事実上、MapForce は時際のデータ変換（またデータマッピング）を行うプログラムコードを生成します。プログラムコードを生成しない場合、（MapForce Professional またはEnterprise Editionsで使用することのできる）MapForce 内蔵の変換言語 を実行します。

マッピング

MapForce マッピングデザイン（または、略して「マッピング」）は、データがどのように1つのフォーマットから他のフォーマットに変換さえるかの視覚的な表現です。マッピングは、データ変換を行うためのMapForce マッピングエリア内で1つのスキーマから他のスキーマに追加するコンポーネントから構成されています。（例: XMLドキュメントを1つのスキーマから他のスキーマに変換）。有効なマッピングは、1つまたは複数のターゲットコンポーネントに接続されている、1つまたは複数の複数のソースコンポーネントから構成されています。マッピングを実行して、結果を直接 MapForce 内で確認することができます。コードを生成し、外部で実行することも可能です。MapForce 実行可能ファイルにマッピングをコンパイルし、MapForce Server またはFlowForce Server を使用してマッピングの実行を自動化することもできます。MapForce は、マッピングを.mfd の拡張子を持つファイルとして保存します。

MFF

MapForce 関数ファイルのファイル名拡張子

MFD

MapForce デザインドキュメント（マッピング）のファイル名拡張子

出力コンポーネント

出力コンポーネント（または「単純型出力」）とは、マッピングから文字列の値を返すことを有効化するMapForce コンポーネントです。出力 コンポーネントは、1つのターゲットコンポーネント型のみをあらわしますが、後者と混同しないように注意してください。

parent-context

min、max、avg、count などのMapForce 一部のコア集計関数 **parent-context** 任意の引数です。複数の階層的シーケンスを持つソースコンポーネントでは、親コンテキストは、ノードのセットがどの関数上で操作されるかを決定します。

ソースコンポーネント

ソースコンポーネントとは、MapForce がデータを読み込む元のコンポーネントです。マッピングを実行すると、ソースコンポーネントのコネクタにより与えられたデータを読み込み、必要とされる型に変換し、ターゲットコンポーネントのコネクタに送信します。

ターゲットコンポーネント

ターゲットコンポーネントとMapForce はがデータを書き込むコンポーネントです。マッピング を実行すると、ターゲットコンポーネントは、外部プログラム内で更に処理するためにMapForce にファイル（または複数のファイル）の生成、または結果を文字列の値として出力するように命令します。ターゲットコンポーネントは、ソース コンポーネントの逆です。

インデックス

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z