プロダクション内での XML 仮想ドキュメントのルーティング

Version 2024.1
2024-06-06

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

プロダクション内での XML 仮想ドキュメントのルーティング
InterSystems IRIS Data Platform　Version 2024.1　2024-06-06

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

XML 仮想ドキュメントの概要

InterSystems IRIS® データ・プラットフォームは、XML ドキュメントを仮想ドキュメントとしてサポートしています。仮想ドキュメントはメッセージの一種で、InterSystems IRIS では部分的にしか解釈されません。この種のメッセージは、標準のプロダクション・メッセージ・ヘッダと標準のメッセージ・プロパティ (ID、Priority、SessionId など) で構成されています。しかし、メッセージ内のデータはメッセージ・プロパティとして提供されず、代わりに、処理速度を向上させるために内部使用のグローバルに直接格納されます。

InterSystems IRIS で提供されている各種のツールでは、仮想ドキュメント内の値にアクセスして、データ変換、ビジネス・ルール、およびメッセージの検索とフィルタリングに使用できます。背景情報は、"プロダクション内での仮想ドキュメントの使用法" を参照してください。

注釈
- InterSystems IRIS では、XML ドキュメントの読み取り時に、XML 宣言、すべての処理手順、およびすべてのコメントが削除されます。
- 要素または属性の名前にピリオド (.) が含まれている場合は、チルダ (~) に置換されます。

  例えば、XML 要素の名前が `My.Element` の場合、InterSystems IRIS では `My~Element` のように表示されます。

XML ドキュメントは標準プロダクション・メッセージとして扱うこともできます。そのためには、対応する XML スキーマからクラスを生成する必要があります。詳細は、"XML ツールの使用法" を参照してください。

プロダクションでは、XML 仮想ドキュメントを使用するよりも XML 対応オブジェクトを使用する方が効率的な場合があります。詳細は、"XML 対応オブジェクトと XML 仮想ドキュメントの比較" を参照してください。

2

XML のドキュメントとスキーマを操作するための各種ツール

ここでは、XML スキーマと XML ドキュメントの操作に使用可能な InterSystems IRIS® ツールの概要を示します。

2.1 XML スキーマ構造ページの使用法

[Interoperability]→[**相互運用**]→[XML]→[**XML スキーマ構造**] ページを使用すると、XML スキーマ仕様のインポートと表示を行うことができます。

このページの使用に関する一般情報は、"スキーマ構造ページの使用法" を参照してください。

スキーマ・ファイルをインポートする前に、名前をそのネームスペース内で一意のわかりやすいものに変更します。ファイル名は、管理ポータルなどでスキーマ・カテゴリ名として使用されます。ファイル名が、ファイル拡張子 `.xsd` で終わる場合、このファイル拡張子はスキーマ・カテゴリ名から省略されます。それ以外の場合、ファイル拡張子は名前に含まれます。

注釈　このドキュメントで説明しているように、これらのスキーマは、XML 仮想ドキュメントの処理をサポートするためだけに使用できます。InterSystems IRIS では、その他の目的では使用されません。

重要　スキーマ・ファイルのインポート後に、そのファイルをファイル・システム内の現在の場所から削除しないでください。XML パーサは、InterSystems IRIS データベースに格納されているスキーマではなくスキーマ・ファイルを使用するからです。

2.2 XML ドキュメント・ビューワ・ページの使用法

[Interoperability]→[**相互運用**]→[XML]→[**XML ドキュメント・ビューワ**] ページを使用すると、XML ドキュメントを表示して、さまざまな方法で解析することによって、使用する DocType を決定できます。また、変換をテストすることもできます。ドキュメントは、外部ファイルにすることも、プロダクション・メッセージ・アーカイブからのドキュメントにすることもできます。

このページを表示するには、[Interoperability]、[**相互運用**]、[XML] の順にクリックします。次に、[**XML ドキュメント・ビューワ**] をクリックし、[**進む**] をクリックします。

このページの使用に関する一般情報は、"ドキュメント・ビューワ・ページの使用法" を参照してください。

2.3 XML スキーマのプログラムによるインポート

スキーマは、プログラムによってロードすることもできます。それには、EnsLib.EDI.XML.SchemaXSD クラスを直接使用します。このクラスには、Import() クラス・メソッドが用意されています。このメソッドに対する最初の引数は、インポートするファイルの名前であり、そのディレクトリの完全パスを含めます。以下に例を示します。

```
set status= ##class(EnsLib.EDI.XML.SchemaXSD).Import("c:\iiris\myapp.xsd")
```

EnsLib.EDI.XML.SchemaXSD クラスには、ImportFiles() メソッドも用意されています。このメソッドの場合は、最初の引数を以下のいずれかの方法で指定できます。

- ファイルのインポート元のディレクトリの名前として指定する。ファイル拡張子に関係なく、このディレクトリ内のすべてのファイルのインポートが試行されます。以下に例を示します。

  ```
  set status=##class(EnsLib.EDI.XML.SchemaXSD).ImportFiles("c:\iiris\")
  ```

- セミコロンで区切られたファイル名のリストとして指定する。これらのファイル名の前にディレクトリの完全パスを含める必要があります。また、ファイル名にはワイルドカードを使用できます。以下に例を示します。

  ```
  set status=##class(EnsLib.EDI.XML.SchemaXSD).ImportFiles("c:\iiris\*.xsd;*.XSD")
  ```

詳細は、EnsLib.EDI.XML.SchemaXSD のクラス参照を参照してください。

重要　スキーマ・ファイルのインポート後に、そのファイルをファイル・システム内の現在の場所から削除しないでください。XML パーサは、InterSystems IRIS データベースに格納されているスキーマではなくスキーマ・ファイルを使用するからです。

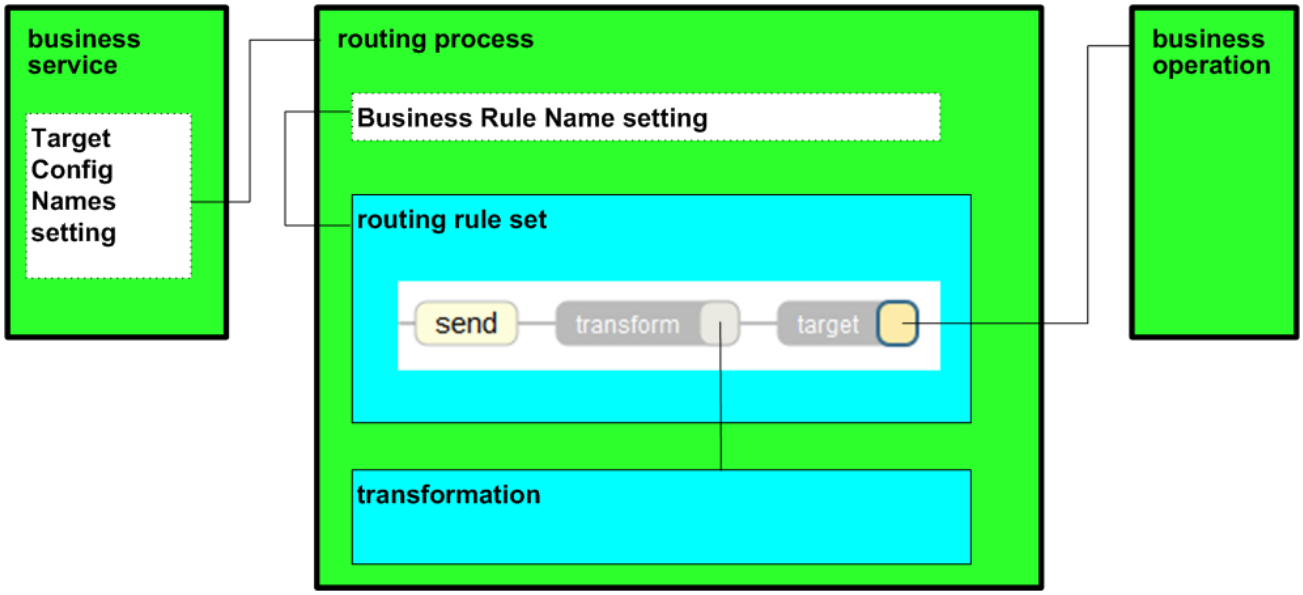
2.4 XML クラス

参照用として、ここでは XML ドキュメントを操作するために InterSystems IRIS から提供されているクラスを列挙します。

項目	クラス	メモ
XML ビジネス・サービス	• EnsLib.EDI.XML.Service.FileService • EnsLib.EDI.XML.Service.FTPService	クラス名が示しているように、これらのビジネス・サービス・クラスのそれぞれで別々のアダプタが使用されます。
XML ルーティング・プロセス	EnsLib.MsgRouter.VDocRoutingEngine	このクラスは標準の仮想ドキュメント・ルーティング・プロセスです。
XML ビジネス・オペレーション	• EnsLib.EDI.XML.Operation.FileOperation • EnsLib.EDI.XML.Operation.FTPOperation	クラス名が示しているように、これらのビジネス・オペレーション・クラスのそれぞれで別々のアダプタが使用されます。
メッセージ	EnsLib.EDI.XML.Document (ビジネス・ホスト・クラスで自動的に使用されます)	XML ドキュメントを仮想ドキュメントとして転送するための特殊なメッセージ・クラスです。
検索テーブル	EnsLib.EDI.XML.SearchTable	XML ドキュメント専用の検索テーブル・クラスです。

これらのクラスのサブクラスを作成して使用することもできます。

ビジネス・ホスト・クラスには構成可能なターゲットが含まれています。それらのいくつかを下の図に示します。



その他の構成可能なターゲットの詳細は、設定のリファレンスを参照してください。

3

プロダクションでの XML 仮想ドキュメントの構成

このトピックでは、XML 仮想ドキュメントをプロダクションで使用するために必要な構成手順を説明します。

ここで説明されていない設定の詳細は、設定のリファレンスを参照してください。

他のトピックでは、プロダクションで使用する項目であるデータ変換、ルール・セット、および検索テーブルを作成する方法について説明します。

3.1 InterSystems IRIS への XML スキーマのロード

XML 仮想ドキュメントの場合は、InterSystems IRIS に該当する XML スキーマをロードすると便利ですが、必須ではありません。InterSystems IRIS でスキーマが使用できると、InterSystems IRIS によるドキュメントの検証が可能になると共に、(DOM スタイル・パスだけでなく) スキーマ依存仮想プロパティ・パスも使用できるようになります。また、ドキュメント構造に関して、DTL エディタおよびビジネス・ルール・エディタの補助が得られます。

XML スキーマを InterSystems IRIS にロードするには、[XML スキーマ構造] ページを使用します。

3.2 受信 XML を仮想ドキュメントとして処理するためのビジネス・サービスの追加

受信 XML ドキュメントを仮想ドキュメントとして処理するためにビジネス・サービスを追加するには、以下の操作を行います。

1. プロダクションに対して、EnsLib.EDI.XML.Service.FileService または EnsLib.EDI.XML.Service.FTPService クラスに基づくビジネス・サービスを追加します。
2. このビジネス・サービスによる受信 XML ドキュメントの検索先を指定します。

   例えば、EnsLib.EDI.XML.Service.FileService に対して [ファイルパス] 設定を指定します。これは、ビジネス・サービスが新しいファイルをチェックするディレクトリです。
3. 必要に応じて、他の設定も指定します。具体的には、以下の設定を指定できます。

- **[Docスキーマカテゴリ]**。受信ドキュメントに適用する XML スキーマを指定します。事前にロードした XML スキーマを選択してください。

  メッセージを検証したい場合は、スキーマを選択する必要があります。スキーマは、検索テーブルを定義する場合にも使用できます。

- **[文字セット]**。受信データの文字セットを指定します。InterSystems IRIS は、自動的にこの文字エンコーディングから変換します。その他のオプションについては、設定のリファレンスの "文字セット" を参照してください。

- **[テーブルクラス検索]**。"XML 仮想ドキュメントの検索テーブルの定義" を参照してください。

  この検索テーブル・クラスが、そのビジネス・ホストで受信するメッセージの種類と合致するようにしてください。例えば、ルート要素が `<Transaction>` のメッセージをビジネス・ホストで受信する場合は、`<Employee>` 要素内でプロパティを使用した検索テーブル・クラスを使用するのは適切ではありません。

4. XML ドキュメントの送信先を指定します。そのためには、**[ターゲット構成名]** 設定で値のカンマ区切りリストを指定します。各値には、ビジネス・プロセスまたはビジネス・オペレーションの名前を指定してください。

3.3 XML 仮想ドキュメントを処理するためのビジネス・プロセスの追加

XML 仮想ドキュメントを処理するためにビジネス・プロセスを追加するには、以下の操作を行います。

1. プロダクションに対して、EnsLib.MsgRouter.VDocRoutingEngine クラスに基づくビジネス・プロセスを追加します。
2. このビジネス・プロセスの場合は、**[ビジネスルール名]** 設定を指定します。XML 仮想ドキュメントに処理を行う適切なビジネス・ルール・セットを選択します。

   これらの定義に関する詳細は、"XML 仮想ドキュメントのルール・セットの定義" を参照してください。
3. 必要に応じて、他の設定も指定します。
4. XML 仮想ドキュメントをこのビジネス・プロセスに送信するために、以下のように同じプロダクション内の該当する 1 つまたは複数のビジネス・ホストを構成します。
   - ビジネス・サービスの場合は、このビジネス・プロセスの名前を含めるように **[ターゲット構成名]** 設定を編集します。
   - ビジネス・プロセスの場合は、メッセージをこのビジネス・プロセスにルーティングする **[ビジネスルール名]** を指定します。

3.4 XML 仮想ドキュメントを処理するためのビジネス・オペレーションの追加

XML 仮想ドキュメントをプロダクションの外部宛てに送信するためにビジネス・オペレーションを追加するには、以下の操作を行います。

1. プロダクションに対して、EnsLib.EDI.XML.Operation.FileOperation または EnsLib.EDI.XML.Operation.FTPOperation クラスに基づくビジネス・オペレーションを追加します。

2. 必要に応じて、このビジネス・オペレーションの設定を指定します。

   例えば、EnsLib.EDI.XML.Operation.FileOperation に対して **[ファイルパス]** 設定を指定します。これは、ビジネス・オペレーションがファイルを書き込むディレクトリです。このデイレクトリは、存在するアクセス可能なディレクトリでなければなりません。

3. 必要に応じて、**[テーブルクラス検索]** 設定を指定します。"XML 仮想ドキュメントの検索テーブルの定義" を参照してください。

   この検索テーブル・クラスが、そのビジネス・ホストで受信するメッセージの種類と合致するようにしてください。例えば、ルート要素が `<Transaction>` のメッセージをビジネス・ホストで受信する場合は、`<Employee>` 要素を参照した検索テーブル・クラスを使用するのは適切ではありません。

4. XML 仮想ドキュメントをこのビジネス・オペレーションに送信するために、以下のように同じプロダクション内の該当する 1 つまたは複数のビジネス・ホストを構成します。

   - ビジネス・サービスの場合は、このビジネス・オペレーションの名前を含めるように **[ターゲット構成名]** 設定を編集します。
   - ビジネス・プロセスの場合は、メッセージをこのビジネス・オペレーションにルーティングする **[ビジネスルール名]** を指定します。

不正なメッセージを処理するためのビジネス・オペレーションを追加することもできます (背景については、"仮想ドキュメント用のビジネス・プロセス" を参照してください)。

4

XML 仮想ドキュメントのプロパティ・パスの概要

このトピックでは、XML 仮想ドキュメントのプロパティ・パスの概要を示します。

次の 2 つのトピックでは、プロパティ・パスの作成方法について詳しく説明します。

注釈　データ変換では通常、ルール・セットや検索テーブルよりもより多くのプロパティ・パスのセットを使用するため、このトピックで示すコード例はデータ変換の一部です。また、DOM スタイル・パスは手動で作成する必要のあるパスなので、ここでは DOM スタイル・パスに重点を置いています (一方、使用するスキーマを指定する場合、InterSystems IRIS にドキュメントの構造が表示されるので、ドラッグ・アンド・ドロップを行うか、自動完了機能を使用すると、スキーマ依存パスが自動的に生成されます)。

4.1 XML 仮想ドキュメントの仮想プロパティ・パスの手引き

ここでは、XML 仮想ドキュメントの仮想プロパティ・パスについて簡単に説明します。

前述のとおり、スキーマ依存パスを使用できるのは、該当する XML スキーマをロード済みの場合のみです。一方、DOM スタイル・パスは、使用可能なスキーマがない場合でも常に使用できます。

4.1.1 スキーマ依存パスの基本構文

XML 仮想ドキュメントでは、スキーマ依存パスは以下の例のようにピリオドで区切られた一連のパス・ユニットで構成されます。

```
unit1.unit2.unit3
```

ここで、unit1 はドキュメント内の子 XML 要素の名前、unit2 は unit1 内の子要素の名前、というように続きます。リーフ・ユニットは、子 XML 要素と XML 属性のどちらかの名前です。

以下に例を示します。

```
HomeAddress.City
```

詳細は、"スキーマ依存パスの指定" を参照してください。

4.1.2 DOM スタイル・パスの基本構文

DOM スタイル・パスは、常にスラッシュで始まり、以下の例のような基本構造を持ちます。

```
/root_unit/unit1/unit2/unit3
```

各パス・ユニットのフォームは以下のとおりです。

```
namespace_identifier:name
```

namespace_identifier は XML ネームスペースを示します。これは、InterSystems IRIS によって実際のネームスペースの URI で置換されるトークンです (詳細は、後の項で説明します)。このトークンは、その要素または属性がネームスペースに含まれている場合にのみ必要です。

name は、XML 要素または属性の名前です。

以下に例を示します。

```
/$2:Patient/$2:HomeAddress/$2:City
```

詳細は、"DOM スタイル・パスの指定" を参照してください。

4.1.2.1 XML ネームスペース・トークン

スキーマを InterSystems IRIS にロードすると、そのスキーマで使用されるネームスペースの一連のトークンが確立され、あらゆる DOM スタイル・パスで使用できるようになります。

トークン `$1` はスキーマで宣言された最初のネームスペースで使用され、これは通常、XML スキーマのネームスペース (`http://www.w3.org/2001/XMLSchema`) と一致します。トークン `$2` はスキーマで宣言された次のネームスペースで使用され、`$3` は 3 番目のネームスペースで使用される、というように続きます。

InterSystems IRIS では、スキーマで宣言されたすべてのネームスペースに対してネームスペース・トークンが割り当てられます。それらのネームスペースが実際に使用されるかどうかは関係ありません。そのため、スキーマで追加のネームスペースが宣言された場合、対象の項目に対して `$2` ではなく `$3` 以上の値が使用される可能性があります。次の節で説明するように、特定のパス・ユニットに正しいトークンを使用していることを確認するには、管理ポータルを使用して個々のパス・ユニットを表示するのが実際に便利です。

ネームスペース・トークンを使用できるのは、該当するスキーマもロード済みの場合 (また、そのスキーマを使用するための適切なビジネス・ホストが構成済みの場合) です。それ以外の場合は、XML ドキュメントで指定されているとおりのネームスペース接頭語を使用する必要があります。

4.2 XML 仮想ドキュメントのパス・ユニットの表示

XML 仮想ドキュメントのプロパティ・パスを十分に理解するまでは、管理ポータルを使用して個々のパス・ユニットを表示すると便利です。そのためには、該当するスキーマをロードしておく必要があります。

スキーマ内の要素および属性のパス・ユニットを表示するには、以下の操作を行います。

1. 前述の説明に従って、スキーマをロードします。

   例えば、以下の参照用の XML スキーマを考えてみましょう。これは、XML スキーマに精通した読者に役立ちます。

   XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified" targetNamespace="http://myapp.com"
xmlns:myapp="http://myapp.com">
  <element name="Patient" type="myapp:Patient"/>
  <complexType name="Patient">
    <sequence>
      <element minOccurs="0" name="Name" type="string"/>
      <element minOccurs="0" name="FavoriteColors"
                  type="myapp:ArrayOfFavoriteColorString" />
      <element minOccurs="0" name="Address" type="myapp:Address" />
      <element minOccurs="0" name="Doctor" type="myapp:Doctor" />
    </sequence>
```

```
    <attribute name="MRN" type="string"/>
    <attribute name="DL" type="string"/>
  </complexType>
  <complexType name="ArrayOfFavoriteColorString">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="FavoriteColor"
               nillable="true" type="string"/>
    </sequence>
  </complexType>
  <complexType name="Address">
    <sequence>
      <element minOccurs="0" name="Street" type="string"/>
      <element minOccurs="0" name="City" type="string"/>
      <element minOccurs="0" name="State" type="string"/>
      <element minOccurs="0" name="ZIP" type="string"/>
    </sequence>
  </complexType>
  <complexType name="Doctor">
    <sequence>
      <element minOccurs="0" name="Name" type="string"/>
    </sequence>
  </complexType>
</schema>
```

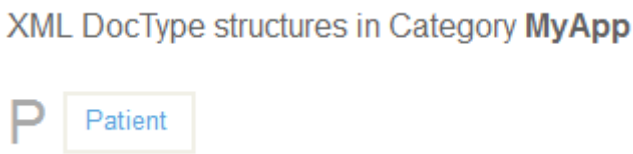
以下に、この節で示すスキーマに従った XML ドキュメントの例を紹介します。

XML

```
<?xml version="1.0" ?>
<Patient MRN='000111222' xmlns='http://myapp.com'>
    <Name>Georgina Hampton</Name>
    <FavoriteColors>
        <FavoriteColor>Red</FavoriteColor>
        <FavoriteColor>Green</FavoriteColor>
    </FavoriteColors>
    <Address>
        <Street>86 Bateson Way</Street>
        <City>Fall River</City>
    </Address>
    <Doctor>
        <Name>Dr. Randolph</Name>
    </Doctor>
</Patient>
```

2. [Interoperability]→[**相互運用**]→[XML]→[**XML スキーマ構造**] ページを選択します。左の列には、このネームスペースにロードされた XML スキーマがリストされます。
3. 目的の XML スキーマに対応する行内の [**カテゴリ**] リンクを選択します。

   上記の XML スキーマに対応する行のリンクをクリックした場合、以下のように表示されます。



4. 目的のドキュメント・タイプのリンクを選択します。

   [Patient] を選択した場合、以下のように表示されます。

XML Document Structure / Document Type Definition

MyApp:Patient

Type: CT:Patient
xsd: element:complexType
Top Element: $2:Patient

	Name	Type	Required	Element
1	Name		No	$2:Name
2	FavoriteColors()	ArrayOfFavoriteColorString()	No	$2:FavoriteColors/$2:FavoriteColor[]
3	Address	Address	No	$2:Address
4	Doctor	Doctor	No	$2:Doctor/$2:Name
5	MRN		No	@MRN
6	DL		No	@DL

このページでは、以下の操作を実行できます。

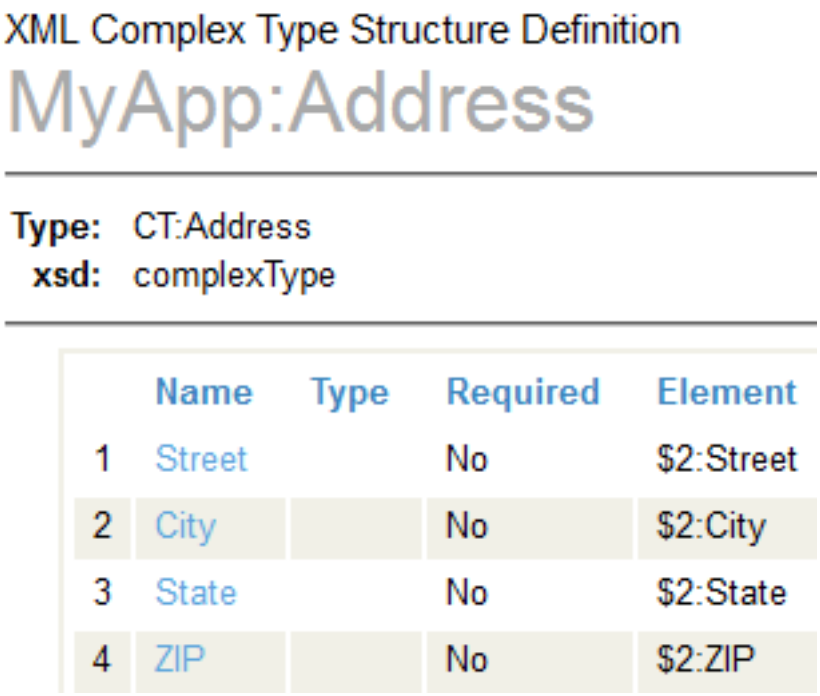
- 上の表で、大文字フォントの値はこの XML 要素の DocType 値を表しています。この場合の DocType は `MyApp:Patient` です。
- **[名前]** 列には、スキーマ依存パスで必要な形式のパス・ユニットが表示されています。

  この場合は、スキーマ依存パス内のパス・ユニットとして、`Name`、`FavoriteColors`、`Address`、`Doctor`、`MRN`、`DL` を使用できることがこのページからわかります。
- **[要素]** 列には、DOM スタイル・プロパティ・パスで必要な形式のパス・ユニットが表示されています。

  この場合は、DOM スタイル・パス内のパス・ユニットとして、`$3:Name`、`$2:FavoriteColors/$2:FavoriteColor`、`$2:Address`、`$2:Doctor/$2:Name`、`@MRN`、`@DL` を使用できることがこのページからわかります。`@MRN` と `@DL` にはネームスペース接頭語が含まれていないことに注意してください。これらの属性はどのネームスペースにも含まれていません。

5. 必要に応じて、追加のサブ項目をクリックします。

   [名前] 列の **[アドレス]** をクリックすると、以下のように表示されます。


XML Complex Type Structure Definition

MyApp:Address

Type: CT:Address
xsd: complexType

	Name	Type	Required	Element
1	Street		No	$2:Street
2	City		No	$2:City
3	State		No	$2:State
4	ZIP		No	$2:ZIP

このページには、Address 内のすべての追加パス・ユニットが表示されています。

この場合は、このページを表示するために使用したパス・ユニットとの組み合わせでこれらの追加パス・ユニットが使用できることがこのページからわかります。以下に例を示します。

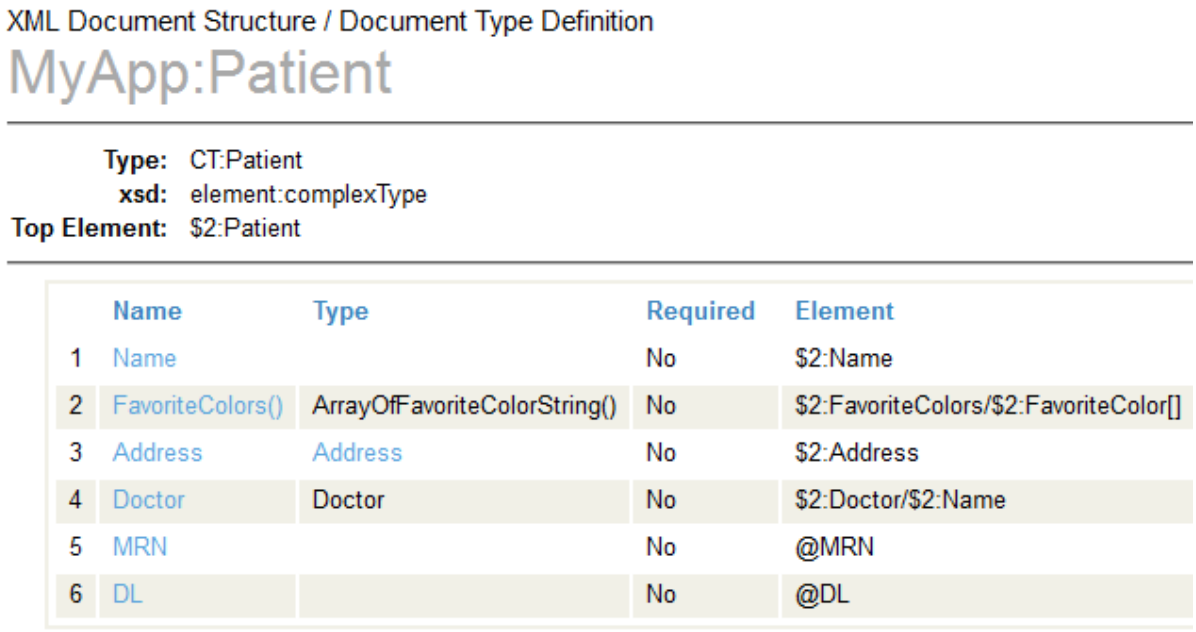
パス・タイプ	例
スキーマ依存パス（部分）	`...Address.Street`
DOM スタイル・パス（部分）	`/.../$2:Address/$2:Street`

以下の節では、スキーマの変動による特定の変動に注目しています。

4.3 スキーマ依存パスの冗長な内部要素

スキーマ依存パスに対して、InterSystems IRIS では冗長な内部要素が縮小されます。これについては、以下の例で最もよく説明しています。

- <FavoriteColors> 要素には、複数の <FavoriteColor> 要素のシーケンスが含まれています。スキーマ・ビューワ・ページの（スキーマ依存パスのパス・ユニットが表示される）[名前] 列には、<FavoriteColors> が FavoriteColors() とだけ表示されます。この列は以下の図では青で表示されています。



XML Document Structure / Document Type Definition

MyApp:Patient

Type: CT:Patient
xsd: element:complexType
Top Element: $2:Patient

	Name	Type	Required	Element
1	Name		No	$2:Name
2	FavoriteColors()	ArrayOfFavoriteColorString()	No	$2:FavoriteColors/$2:FavoriteColor[]
3	Address	Address	No	$2:Address
4	Doctor	Doctor	No	$2:Doctor/$2:Name
5	MRN		No	@MRN
6	DL		No	@DL

これに対して、右側の [**要素**] 列には、同じ要素が `$2:FavoriteColors/$2:FavoriteColorsItem` として表示されます。この列には、DOM スタイル・パスのパス・ユニットが表示されます。

同じタイプを持つ複数の項目のシーケンスの場合、スキーマ依存パスでは内部要素の名前は使用されません (一方、DOM スタイル・パスではすべての要素名が使用されます)。多くの場合、スキーマに含まれる冗長な内部レベルはすべて、スキーマ依存パスでは無視されます。以下の項目で別の例を示します。

- `<Doctor>` 要素には、単一の `<Name>` 要素が含まれています。前の図で示したように、スキーマ・ビューワ・ページの [**名前**] 列には、`<Doctor>` 項目が Doctor として表示されます。

  `<Doctor>` 内のデータへのスキーマ依存パスでは内部要素の名前が使用されないことに注意してください。

  これに対して、右側の [**要素**] 列には、同じ項目が `$3:Doctor/$3:Name` として表示されます。この列には、DOM スタイル・パスのパス・ユニットが表示されます。

注釈　単一の繰り返されない外側要素があると、それはたたんで表示されます。これにより、プロパティ・パスが短くなるので内側要素のみを使用できます。外側要素が繰り返されている場合、このパスは短くならないので、すべての内側要素にアクセスできます。

ただし、外側要素のタイプを参照で定義していると、このパスが短くなるという制限があります。その場合、スキーマ依存のパスを使用すると、外側要素の先頭ノードよりも後の内側要素にはアクセスできません。DOM スタイルのパスを使用すれば、すべての内側要素にアクセスできます。この状態を防止するには、唯一の外側要素が繰り返されている場合、そのタイプを参照で定義しないようにします。

4.4 繰り返しフィールド

指定された要素が複数回生じる場合、[**名前**] 列には要素名の末尾に括弧 **( )** が表示されます。例えば、前の図の FavoriteColors() 行を参照してください。

[**タイプ**] 列と [**要素**] 列には、要素の繰り返し可能な回数が指定されます。この例の場合、要素は 5 回繰り返すことができます。[**タイプ**] 列の括弧に数字が表示されていない場合、要素は何回でも繰り返すことができます。

4.5 繰り返しシーケンス

構造の中でのみ繰り返され、周囲の要素から区別するための要素名を持たない繰り返しシーケンスを XML スキーマで使用できます。これにより、VDoc プロパティ・パスに `sequence(rep)` が記述されますが、DOM パスには同等の記述はありません。この場合は、このプロパティ・パスを使用して、シーケンスの繰り返し全体の値（`People(1).sequence(2)` など）またはシーケンス内部の値（`People(1).sequence(2).Color.Tint` など）を取得できます。このプロパティ・パスを使用すると、シーケンス内部にある要素の値を設定することもできます。

シーケンス全体には値を設定できないことに注意します。例えば、以下のように指定するとエラーが返され、どの値も変更されません。

```
SetValueAt("<Color><ColorName>pink</ColorName><Tint>bright</Tint></Color><Value>001</Value>","People(1).sequence(4)")
```

4.6 重複名

XML スキーマに含まれる同じレベルの複数の要素が同じ名前を持ち、かつそれぞれのタイプが異なる場合は、そのレベルで一意の名前にする必要があるため、_2 や _3（以下同様）が InterSystems IRIS によって付加されます。この処理は、スキーマ依存パスにのみ適用されます。例えば、`<Contact>` という名前の 2 つの要素を含む `<Person>` 要素を定義するスキーマについて考えてみましょう。1 つは `<Phone>` タイプ、もう 1 つは `<Assistant>` タイプとします。InterSystems IRIS では、`<Person>` 要素のスキーマが以下のように表示されます。

Type: DS:Person
xsd: complexType

	Name	Type	Required	Element
1	Contact	Phone	No	$2:Contact
2	Contact_2	Assistant	No	$2:Contact

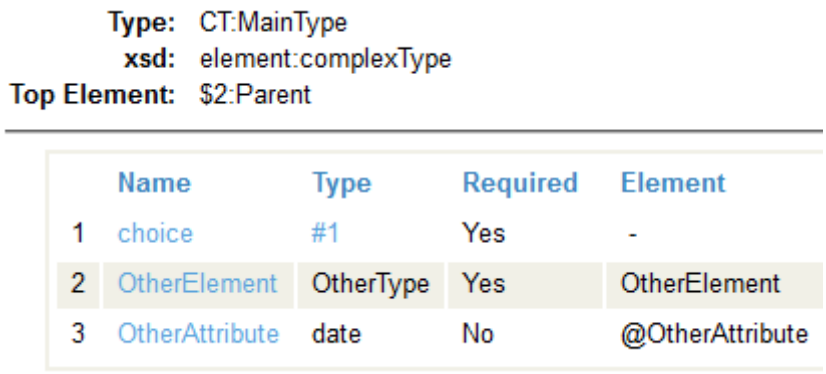
同様に、このスキーマに含まれる同じレベルの複数の要素が同じ名前を持ち、かつそれぞれが異なるネームスペースに含まれる場合は、そのレベルで一意の名前にする必要があるため、_2 や _3（以下同様）が InterSystems IRIS によって付加されます。この処理は、スキーマ依存パスにのみ適用されます。

4.7 Choice 構造

一部のスキーマには、下の例のように、<choice> 構造が含まれています。

```
<xsd:choice>
  <xsd:element name="OptionA"  type="my:OptionType"/>
  <xsd:element name="OptionB"  type="my:OptionType"/>
  <xsd:element name="OptionC"  type="my:OptionType"/>
</xsd:choice>
```

InterSystems IRIS は、この構造を 2 種類のパスとして別々に表現します。以下に例を示します。

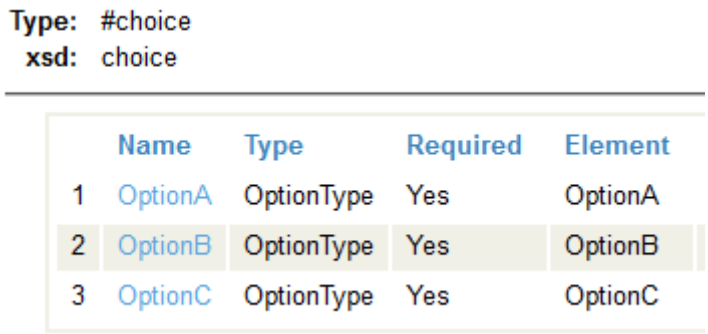


Type: CT:MainType
xsd: element:complexType
Top Element: $2:Parent

	Name	Type	Required	Element
1	choice	#1	Yes	-
2	OtherElement	OtherType	Yes	OtherElement
3	OtherAttribute	date	No	@OtherAttribute

スキーマ依存パスの場合は、[**名前**] に <choice> 構造の一般名が、[**タイプ**] 列に数値プレースホルダが表示されます。[**要素**] 列には何も表示されません。

`choice` をクリックすると、以下のように表示されます。



Type: #choice
xsd: choice

	Name	Type	Required	Element
1	OptionA	OptionType	Yes	OptionA
2	OptionB	OptionType	Yes	OptionB
3	OptionC	OptionType	Yes	OptionC

この場合は、これらのページから、以下のパスを使用して `OptionB` にアクセスできることがわかります。

パス・タイプ	例
スキーマ依存パス（部分）	`...Parent.choice.OptionB`
DOM スタイル・パス（部分）	`/.../Parent/OptionB`

4.8 参照によって追加されたグループ

スキーマには、ref 属性経由で追加された <group> を含めることができます。以下に例を示します。

```
<s01:complexType name="Patient">
   <s01:sequence>
      <s01:element name="Name" type="s01:string" minOccurs="0"/>
      <s01:element name="Gender" type="s01:string" minOccurs="0"/>
      <s01:element name="BirthDate" type="s01:date" minOccurs="0"/>
      <s01:element name="HomeAddress" type="s02:Address" minOccurs="0"/>
      <s01:element name="FavoriteColors"
        type="s02:ArrayOfFavoriteColorsItemString" minOccurs="0"/>
      <s01:element name="Container" type="s02:ContainerType" minOccurs="0"/>
      <s01:element name="LatestImmunization" type="s02:Immunization" minOccurs="0"/>
      <s01:element ref="s02:Insurance" minOccurs="0"/>
      <s01:group ref="s02:BoilerPlate" minOccurs="1" maxOccurs="1"/>
   </s01:sequence>
...
<s01:group name="BoilerPlate">
   <s01:sequence>
      <s01:element name="One" type="s01:string"/>
      <s01:element name="Two" type="s01:string"/>
```

```
      <s01:element name="Three" type="s01:string"/>
   </s01:sequence>
</s01:group>
```

InterSystems IRIS は、この構造を 2 種類のパスとして別々に表現します。以下に例を示します。

Type: CT:Patient
xsd: element:complexType
Top Element: $2:Patient

	Name	Type	Required	Element
1	Name		No	$2:Name
2	Gender		No	$2:Gender
3	BirthDate	date	No	$2:BirthDate
4	HomeAddress	Address	No	$2:HomeAddress/$2:Address
5	FavoriteColors()	ArrayOfFavoriteColorsItemString()	No	$2:FavoriteColors/$2:FavoriteColorsItem[]
6	Container	ContainerType	No	$2:Container/$2:Intermediate/$2:Final
7	LatestImmunization	Immunization	No	$2:LatestImmunization
8	Insurance	Insurance	No	$2:Insurance
9	BoilerPlate	#9	Yes	BoilerPlate
10	MRN		No	@MRN

スキーマ依存パスの場合は、[**名前**] にグループの名前が、[**タイプ**] 列に数値プレースホルダが表示されます。[**要素**] 列にもグループの名前が表示されます。

`BoilerPlate` をクリックすると、以下のように表示されます。

Type: CG:BoilerPlate
xsd: sequence:group

	Name	Type	Required	Element	Default	Description
1	One		Yes	$2:One		
2	Two		Yes	$2:Two		
3	Three		Yes	$2:Three		

この場合は、これらのページから、以下のパスを使用して `Two` にアクセスできることがわかります。

パス・タイプ	例
スキーマ依存パス（部分）	`...Patient.BoilerPlate.Two`
DOM スタイル・パス（部分）	`/.../$2:Patient/$2:Two`

5

XML 仮想ドキュメントのスキーマ依存パスの指定

ここでは、XML 仮想ドキュメントのスキーマ依存パスを指定する方法を説明します。

これらのパスを使用して、値にアクセスしたり値を設定したりすることができます。

このトピックの例では、前のトピックで示したスキーマが使用されています。

5.1 XML 要素の内容の取得または設定

要素の内容にアクセスするには、以下のスキーマ依存パスのいずれかを使用します。また、これらのパスは、より複雑なスキーマ依存パスを作成する場合にも使用します (これについては、後の項で説明します)。

構文	参照先
element_name	指定された要素の内容。element_name は、ルート要素の子である必要があります。
parent . element_name	指定された要素の内容。parent は、要素への完全パス (つまり、このテーブルに示す任意の構文) です。この場合、element_name は、parent によって参照される要素の子となります。
parent . element_name (n)	指定された親に含まれる、element_name という名前の n 番目の要素の内容。
parent . element_name (-)	指定された親に含まれる、element_name という名前の最後の要素の内容。

以下の XML ドキュメントについて考えてみましょう。

XML

```
<?xml version="1.0" ?>
<Patient MRN='000111222' xmlns='http://myapp.com'>
    <Name>Georgina Hampton</Name>
    <FavoriteColors>
        <FavoriteColor>Red</FavoriteColor>
        <FavoriteColor>Green</FavoriteColor>
    </FavoriteColors>
    <Address>
        <Street>86 Bateson Way</Street>
        <City>Fall River</City>
    </Address>
    <Doctor>
        <Name>Dr. Randolph</Name>
    </Doctor>
</Patient>
```

以下のテーブルに、このドキュメントのパスの例をいくつか示します。

パスの例	現在のパスの値
	Georgina Hampton
FavoriteColors(1)	Red
FavoriteColors(2)	Green
FavoriteColors(-)	Green
Address	86 Bateson WayFall River
Address.Street	86 Bateson Way
Doctor	Dr. Randolph

5.2 XML 属性の値の取得または設定

属性の値にアクセスするには、以下のスキーマ依存パスのいずれかを使用します。この節の以降の部分では、上記のテーブルで説明したように、element_reference は完全なスキーマ依存パスです。

構文	参照先
element_reference . attribute_name	element_reference によって示される要素の attribute_name 属性の値。

下の表に、前のドキュメントのパスの例を示します。

パスの例	現在のパスの値
MRN	000111222

5.3 コメントと説明

InterSystems IRIS® では、XML ファイルの読み取り時にコメントが削除されます。したがって、スキーマを記述するときにコメントを使用すべきではありません。コメントを使用する代わりに、ほとんどのスキーマ要素で利用できる description または altdesc 属性を使用できます。

多くの状況では使用できませんが、以下のいずれかのスキーマ依存パスを使用することでコメントにアクセスできます。

構文	参照先
element_reference.#	指定された要素の最初のコメントのテキスト。
element_reference.#(n)	指定された要素の n 番目のコメントのテキスト。
element_reference.#(-)	最後のコメントのテキスト。

注釈　InterSystems IRIS では、XML ファイル内の読み取り時にすべてのコメントが削除されます。存在する可能性のあるコメントは、XML ファイルの読み取り以降に追加されたコメントのみです。コメントを追加するには、SetValueAt() を前述の表に示されているパスなどと共に使用します。

5.4 スキーマ依存パス設定時の混合コンテンツの使用

要素とテキスト・ノードの両方を含む値へのパスを設定できます。以下に例を示します。

ObjectScript

```
set mixed="SOME TEXT<HOMETOWN>BELMONT</HOMETOWN>"
set status=target.SetValueAt(mixed,"Address")
```

要素とテキスト・ノードの組み合わせは、混合コンテンツと呼ばれます。

スキーマ依存パスの場合、InterSystems IRIS は、左山括弧 (**<**) 文字に続いて以下の文字セットのいずれかを含んでいる場合に、値が混合コンテンツであると判断します。

- スラッシュと右山括弧 (**/>**)
- 左山括弧とスラッシュ (**</**)

次のテーブルは、InterSystems IRIS が各種スキーマ依存パスの混合コンテンツを処理する方法を示しています。

パスの参照先	InterSystems IRIS が混合コンテンツを処理する方法
要素またはコメント	InterSystems IRIS は、要素またはコメントの現在のコンテンツを特定の混合コンテンツに置き換えます。
属性	サポート対象外

DOM スタイル・パスの混合コンテンツの詳細は、"DOM スタイル・パス設定時の混合コンテンツの使用" を参照してください。

5.5 スキーマ依存パス設定時の文字のエスケープ

スキーマ依存パスを左山括弧 (**<**) を含む値に設定したものの、その値が混合コンテンツの条件に適合していない場合、InterSystems IRIS では、その文字が対応する文字エンティティ参照 (`<`) に置き換えられます。左山括弧を明示的にエスケープする必要はありません。例えば、スキーマ依存パスの値を `"sample value < 20%"` として指定する場合、パスは `"sample value < 20%"` に設定されます。

注釈　DOM スタイル・パスの設定時に、InterSystems IRIS によって、左山括弧文字が自動的にエスケープされることはありません。

5.6 繰り返し要素の特殊なバリエーション

この節では、繰り返し要素を参照するときに適用される仮想プロパティ・パスのバリエーションについて説明します。

5.6.1 繰り返し要素を通した繰り返し

パスで繰り返し要素が参照されている場合は、以下の構文を使用してその要素のすべてのインスタンスを通して繰り返すことができます。

構文	参照先
element_name( )	指定されたコンテキスト内の指定された名前の要素を通して繰り返します。

ここで、以下のコードのみを含むデータ変換を使用するとします。

ObjectScript

```
 set status=target.SetValueAt("REPLACED COLOR","FavoriteColors()")
 if 'status {do $system.Status.DisplayError(status) quit}
```

このコード行は、前述したドキュメントを次のように変換します。

XML

```
<?xml version="1.0" ?>
<Patient MRN='000111222' xmlns='http://myapp.com'>
    <Name>Georgina Hampton</Name>
    <FavoriteColors>
        <FavoriteColor>REPLACED COLOR</FavoriteColor>
        <FavoriteColor>REPLACED COLOR</FavoriteColor>
    </FavoriteColors>
    <Address>
        <Street>86 Bateson Way</Street>
        <City>Fall River</City>
    </Address>
    <Doctor>
        <Name>Dr. Randolph</Name>
    </Doctor>
</Patient>
```

5.6.2 要素のカウント

パスで繰り返し要素が参照されている場合は、以下の構文を使用して要素の数を返すことができます。

構文	参照先
element_name(*)	指定されたコンテキスト内の指定された名前の要素の数。この構文は、スキーマで element_name が繰り返し要素として定義されている場合にのみ有効です。
element_name . *	指定されたコンテキスト内の指定された名前の要素の数。この構文は、任意の element_name に対して有効です。

以下の表に、前述したドキュメント用のパスの例を示します。

パスの例	現在のパスの値
FavoriteColors.*	2
FavoriteColors(*)	2

5.7 ターミナルでのスキーマ依存パスのテスト

特に、構文に精通している場合は、仮想ドキュメント・プロパティ・パスをビジネス・プロセスやデータ変換などで使用する前にターミナルでテストできると便利です。スキーマ依存 XML パスに対してこれを行うには、次の手順を実行します。

1. 対応する 1 つまたは複数の XML スキーマを InterSystems IRIS にロードします。そのためには、[XML スキーマ構造] ページを使用します。
2. 管理ポータルを使用して、テストするドキュメントのルート要素に関する DocType 値を探します。以下に例を示します。

   ″XML 仮想ドキュメントのパス・ユニットの表示″ を参照してください。
3. ターミナルまたはテスト・コード内:
   a. 適切な XML ドキュメントのテキストを含む文字列を作成します。
   b. EnsLib.EDI.XML.Document の ImportFromString() メソッドを使用してこの文字列から XML 仮想ドキュメントのインスタンスを作成します。
   c. このインスタンスの DocType プロパティを設定します。
   d. このインスタンスの GetValueAt() および SetValueAt() メソッドを使用します。

下のメソッドはステップ 3 を実行します。

```
ClassMethod TestSchemaPath()
{
    set string="<Patient xmlns='http://myapp.com'>"
    _"<Name>Jack Brown</Name>"
    _"<Address><Street>233 Main St</Street></Address>"
    _"</Patient>"
    set target=##class(EnsLib.EDI.XML.Document).ImportFromString(string,.status)
    if 'status {do $system.Status.DisplayError(status)  quit}

    //Use the DocType displayed in the Management Portal
    set target.DocType="MyApp:Patient"

    set pathvalue=target.GetValueAt("Address.Street",,.status)
    if 'status {do $system.Status.DisplayError(status)  quit}
    write pathvalue
}
```

このメソッドの出力を以下に示します。

```
SAMPLES>d ##class(Demo.CheckPaths).TestSchemaPath()
233 Main St
```

GetValueAt() のその他のオプションは、″pFormat 引数″ を参照してください。

6

XML 仮想ドキュメントの DOM スタイル・パスの指定

ここでは、XML 仮想ドキュメントの DOM スタイル・パスを指定する方法を説明します。

これらのパスを使用して、値にアクセスしたり値を設定したりすることができます (例外についても説明します)。

以下の節のほとんどでは、ドキュメントに XML ネームスペースを使用しないことを前提としています。また、最後の節では、XML ネームスペースを使用するドキュメントにこれらのパスを適合させる方法について説明します。

このトピックの例では、"XML 仮想ドキュメントのプロパティ・パスの概要" で示したスキーマが使用されています。

6.1 ノードの取得または設定 (基本的なパス)

XML 仮想ドキュメントには、ルート・ノード、要素、テキスト・ノード、コメント、および処理手順の 5 種類のノードがあります。ルート・ノードおよびいずれの要素も、あらゆるタイプの子ノードを持つことができます。その他の種類のノードは、子ノードを持つことができません。また、属性はノードではありません。

以下のテーブルは、XML 仮想ドキュメントのノードの多くを取得または設定するための、基本的な DOM スタイル・パスを示しています。同じタイプまた同じ名前を持つ複数のノードがある場合で、最初のノードを使用したくない場合には、次の節を参照してください。

また、これらのパスは、より複雑な DOM スタイル・パスを作成する場合にも使用します (これについては、後の項で説明します)。

構文	参照先
	ルート・ノードの内容。DOM スタイル・パスを使用することが明確である場合 (つまり、スキーマがロードされていない場合) は、"" を使うこともできます。
/root_element_name	名前が root_element_name であるルート要素の内容。
parent/element_name	指定された親に含まれる、指定された名前 (element_name) を持つ最初の要素の内容。parent は、その親への完全パスです (常に最初にスラッシュが含まれます)。
element_reference/text()	element_reference によって示される要素内の最初のテキスト・ノード。

構文	参照先
element_reference/comment()	element_reference によって示される要素内の最初のコメント。 返される値には、開始構文 (<!--) または終了構文 (-->) は含まれません。同様に、値の設定時にも、開始構文または終了構文は含めないでください。 InterSystems IRIS® では、XML ファイル内の読み取り時にすべてのコメントが削除されます。存在する可能性のあるコメントは、ユーザが追加したコメントのみです。(それらを追加するには、SetValueAt() をここで示しているようなパスと共に使用します。)
element_reference/instruction()	element_reference によって示される要素内の最初の処理手順。 返される値には、開始構文 (<?) または終了構文 (?>) は含まれません。同様に、値の設定時にも、開始構文または終了構文は含めないでください。 InterSystems IRIS では、XML ファイル内の読み取り時にすべての処理手順が削除されます。存在する可能性のある手順は、ユーザが追加した手順のみです。(それらを追加するには、SetValueAt() をここで示しているようなパスと共に使用します。)

以下の XML ドキュメントについて考えてみましょう。

XML

```
<?xml version="1.0" ?>
<Patient xmlns='http://myapp.com'>Sample text node
    <!--Sample comment-->
    <!--Another comment-->
    <Name>Jane Doe</Name>
    <Address>
        <Street>100 Blank Way</Street>
    </Address>
</Patient>
```

以下のテーブルに、このドキュメントのパスの例をいくつか示します。

パスの例	現在のパスの値
/Patient/Name	Jane Doe
/Patient/Address	<Street>100 Blank Way</Street> この場合、参照される要素には (前の例とは対照的に) 子要素が含まれます。DOM スタイル・パスと値の照合時には、空白は無視されることに留意してください。そのため、ドキュメントに改行やインデントが含まれているかどうかに関係なく、ここでの値は指定されたパスと一致します。
/Patient/Address/Street	100 Blank Way
/Patient/text()	Sample text node
/Patient/comment()	Sample comment

ここで、以下のコードのみを含むデータ変換を使用するとします。

ObjectScript

```
 set status=target.SetValueAt("892 Broadway","/Patient/Address/Street")
 if 'status {do $system.Status.DisplayError(status) quit}
 set status=target.SetValueAt("Dr. Badge","/Patient/Doctor/Name")
 if 'status {do $system.Status.DisplayError(status) quit}
```

これらのパスの 1 つは既に存在し、もう 1 つは存在しませんが、両方のパスが有効であることに注意してください。この変換を使用すると、新しいドキュメントは以下のようになります。

XML

```
<?xml version="1.0" ?>
<Patient xmlns='http://myapp.com'>Sample text node
   <!--Sample comment-->
   <!--Another comment-->
   <Name>Jane Doe</Name>
   <Address>892 Broadway</Address>
   <Doctor>
      <Name>Dr. Badge</Name>
   </Doctor>
</Patient>
```

6.2 DOM スタイル・パス設定時の混合コンテンツの使用

要素とテキスト・ノードの両方を含む値へのパスを設定できます。以下に例を示します。

ObjectScript

```
 set mixed="SOME TEXT<HOMETOWN>BELMONT</HOMETOWN>"
 set status=target.SetValueAt(mixed,"/Patient/Address/Street")
```

要素とテキスト・ノードの組み合わせは、混合コンテンツと呼ばれます。

DOM スタイル・パスの場合、InterSystems IRIS は、左山括弧 (**<**) 文字を含む場合に、値が混合コンテンツであると判断します。したがって、DOM スタイル・パスを、左山括弧を含み、かつ有効な XML ではない値に設定する必要がある場合は、文字エンティティ参照 (**<**) を使用して、エラーを回避します。

次のテーブルは、InterSystems IRIS が各種ノードの混合コンテンツを処理する方法を示しています。

ノードの種類	InterSystems IRIS がノード値に指定された混合コンテンツを処理する方法
root	サポート対象外
要素またはコメント	InterSystems IRIS は、ノードの現在のコンテンツを特定の混合コンテンツに置き換えます。
テキスト・ノードまたは命令	InterSystems IRIS は、XML の特殊文字をエスケープし、特定のノードの現在のコンテンツを置き換えます。

注釈　　また、属性はノードではありません。

スキーマ依存パスの混合コンテンツの詳細は、"スキーマ依存パス設定時の混合コンテンツの使用" を参照してください。

6.3 基本的なパス修飾子の使用

上記の節に示した基本的なパスの最後に、以下の基本的なパス修飾子を追加できます (例外についても説明します)。基本的なパスのいずれかを使用するのと同じ方法で、結果のパスを使用することが可能です。

[n]

項目の位置によって項目を参照します。その項目のインスタンスのみがカウントされ、他のタイプの項目は無視されます。

- 値を取得する場合、この構文によって基本的なパスの参照先の項目の n 番目のインスタンス (または空の文字列) が返されます。
- 値を設定する場合、この構文によって基本的なパスの参照先の項目の n 番目のインスタンスが上書きまたは作成されます。

代わりにハイフン (-) を指定すると、最後のインスタンスにアクセスできます。また、角かっこを省略することもできます。

/[n]

子要素の位置によって子要素を参照します。

代わりにハイフン (-) を指定すると、最後の子にアクセスできます。また、角かっこを省略することもできます。

制限事項:

- これは、要素を参照する基本パスのみで使用できます。comment() などの関数と併用することはできません。
- この構文を使用できるのは、値を設定する場合ではなく、値を取得する場合に限られます。

このパス修飾子を他のパス修飾子と組み合わせるには、/[n] 修飾子を最終修飾子として使用します。

[$n]

項目をノード位置によって示します。

- 値を取得する場合、この構文によって基本パスの参照先の項目の n 番目のインスタンスが返されます。それ以外の場合、このパスは無効となり、エラーが返されます。
- 値を設定する場合、この構文によって基本パスの参照先の項目の n 番目のインスタンスが上書きされます。それ以外の場合、このパスは無効となり、エラーが返されます。

後の節で示す各種のパス修飾子を使用することで、ノードを挿入したり追加したりできます。("パス修飾子のサマリ"も参照してください。)

以下の XML ドキュメントについて考えてみましょう。

XML

```
<?xml version="1.0" ?>
<Patient xmlns='http://myapp.com'>
    <!--Sample comment-->
    <!--Another comment-->
    Sample text node
    <Name>Fred Williams</Name>
    <FavoriteColors>
      <FavoriteColor>Red</FavoriteColor>
      <FavoriteColor>Green</FavoriteColor>
    </FavoriteColors>
    <Doctor>
      <Name>Dr. Arnold</Name>
    </Doctor>
</Patient>
```

以下のテーブルに、このドキュメントのパスの例をいくつか示します。

パスの例	現在のパスの値	メモ
/Patient/Name	Fred Williams	
	Fred Williams	このパスは、ドキュメントの最初の要素内の最初の子要素にアクセスします (XML 標準に従って、これがドキュメント内の唯一の要素です)。ここでは、角かっこは省略可能です。
/Patient/FavoriteColors/[1]	Red	ここでは、角かっこは省略可能です。
/Patient/FavoriteColors/[2]	Green	ここでは、角かっこは省略可能です。
	Red	ここでは、角かっこは省略可能です。
	Green	ここでは、角かっこは省略可能です。
/Patient/Name[$1]	空の文字列	このパスは無効です。<Patient> 内の最初のノードは <Name> 要素ではありません。
/Patient/Name[$4]	Fred Williams	
/Patient/Doctor[$6]	<Name xmlns='http://myapp.com'>Dr. Arnold/Name>	
/Patient/4	空の文字列	このパスは無効です。<Patient> には 4 つ目の要素がありません。
/Patient/comment()[1]	Sample comment	角かっこが必要です。角かっこを省略すると、このパスは要素名として解釈されるためです。
/Patient/comment()[2]	Another comment	角かっこが必要です。角かっこを省略すると、このパスは要素名として解釈されるためです。

パスの例	現在のパスの値	メモ
/Patient/comment()[$2]	Another comment	角かっこが必要です。角かっこを省略すると、このパスは要素名として解釈されるためです。
/Patient/comment()[-]	Another comment	角かっこが必要です。角かっこを省略すると、このパスは要素名として解釈されるためです。

6.4 Full() 関数の使用

要素を参照するパス(基本的なパス、または基本的な修飾子を使用するパスのいずれか)の場合、要素の開始タグおよび終了タグを取得することもできます。そのためには、パスの最後に `full()` を追加します。

値を設定する場合は、full() 関数が使用できます。DTL 内では、これが **[追加]** アクションを使用するデータ変換内でしか許可されません。"XML 仮想ドキュメントの割り当てアクション" を参照してください。

以下の XML ドキュメントについて考えてみましょう。

XML

```
<?xml version="1.0" ?>
<Patient xmlns='http://myapp.com'>
    <Name>Jack Brown</Name>
    <Address>
        <Street>233 Main St</Street>
    </Address>
</Patient>
```

以下のテーブルに、このドキュメントのパスの例をいくつか示します。

パスの例	現在のパスの値
/Patient/Name/full()	<Name xmlns='http://myapp.com'>Jack Brown</Name
/Patient/Address/full()	
/Patient/Address/Street/full()	

ルート・ノードの場合、full() 関数は暗黙的に示されます。つまり、以下の 2 つのパスは同等です。

```
/
/full()
```

注釈　GetValueAt() を使用する場合は、完全な要素を取得する追加のフォーマット引数 (`f`) も使用できます。詳細は、"pFormat 引数" を参照してください。

6.5 XML 属性の値の取得または設定

属性の値にアクセスするには、以下の DOM スタイル・パスのいずれかを使用します。この節の以降の部分では、element_reference は完全な DOM スタイル・パスです。

構文	参照先
element_reference/@attribute_name	指定された要素の指定された属性の値。
element_reference/@[n]	(値を取得する場合にのみ使用) 指定された要素の (アルファベット順で) n 番目の属性の値。
element_reference/@[-]	指定された要素の最後の属性の値。

角かっこを省略することもできます。

例えば、以下の XML ドキュメントについて考えてみましょう。

XML

```
<?xml version="1.0" ?>
<Patient MRN='000111222' DL='123-45-6789' xmlns='http://myapp.com'>
    <Name>Liz Jones</Name>
</Patient>
```

以下のテーブルに、このドキュメントのパスの例をいくつか示します。

パスの例	現在のパスの値
/Patient/@MRN	000111222
/Patient/@[1]	000111222
/Patient/@2	

6.6 ノードを挿入または追加するためのパス修飾子の使用

ノードを挿入または追加するには、以下のパス修飾子を基本的なパスの最後に追加します。ここに示すパス修飾子は、値を設定する場合にのみ使用してください。

いくつかの追加オプションについて、次の節も参照してください。

[~n]

指定されたコンテキストにおいて、基本的なパスの参照先の項目のインスタンスを、その項目の n 番目のインスタンスの直前に挿入します。上書きは一切されません。詳細は、以下のテーブルを参照してください。

この項の以降の部分では、n は整数です。

パスの例	動作
/Patient/Episode[~5]	<Patient> 内に存在する 5 番目の <Episode> 要素の前に、新しい <Episode> 要素を挿入します。 <Patient> に 5 番目の <Episode> 要素が含まれていない場合、InterSystems IRIS によってパディングが行われ、空の <Episode> 要素が作成されるため、挿入された <Episode> が 5 番目の <Episode> になります。新しく挿入される要素はすべて、<Patient> 要素の最後に挿入されます。 パスが中間の存在しない要素を参照する場合、InterSystems IRIS によってそれらが作成されます。

パスの例	動作
/Patient/element(Episode)[~5]	<Patient> 内に存在する 5 番目の要素の前に、<Episode> 要素を挿入します。 <Patient> に 5 番目のどのタイプの要素も含まれていない場合、このパスは無効になります。element() 関数によって、パディングのために空の要素が生成されることはありません。
/Patient/[~5]	指定できません。挿入する要素の種類に関する情報がないためです。
/Patient/element()[~5]	

例えば、以下の XML ドキュメントについて考えてみましょう。

XML

```
<?xml version="1.0" ?>
<Patient xmlns='http://myapp.com'>
    <Name>Betty Hodgkins</Name>
    <FavoriteColors>
        <FavoriteColor>Purple</FavoriteColor>
    </FavoriteColors>
</Patient>
```

また、データ変換内の以下のコードについても考えてみましょう。

ObjectScript

```
 set status=target.SetValueAt("INSERTED COLOR","/Patient/FavoriteColors/FavoriteColor[~4]")
 if 'status {do $system.Status.DisplayError(status) quit}
```

このコード行によって、元のドキュメントが以下のように変換されます。

XML

```
<?xml version="1.0" ?>
<Patient>
    <Name>Betty Hodgkins</Name>
    <FavoriteColors>
        <FavoriteColor>Purple</FavoriteColor>
        <FavoriteColor/>
        <FavoriteColor/>
        <FavoriteColor>INSERTED COLOR</FavoriteColor>
    </FavoriteColors>
</Patient>
```

もう 1 つの例として、以下の XML ドキュメントについて考えてみましょう。

XML

```
<Patient xmlns='http://myapp.com'>
    <Name>Colin McMasters</Name>
    <Address>
        <Street>102 Windermere Lane</Street>
    </Address>
</Patient>
```

また、データ変換内の以下のコードについても考えてみましょう。

ObjectScript

```
 set status=target.SetValueAt("INSERTED ADDRESS","/Patient/Address/Street[~2]")
 if 'status {do $system.Status.DisplayError(status) quit}
```

このコード行によって、元のドキュメントが以下のように変換されます。

```
<?xml version="1.0" ?>
<Patient>
    <Name>Colin McMasters</Name>
    <Address>
        <Street>102 Windermere Lane</Street>
        <Street>INSERTED ADDRESS</Street>
    </Address>
</Patient>
```

[~$n]

基本的なパスの参照先の項目のインスタンスを、指定された親に含まれるn番目のノードの直前に挿入します。上書きは一切されません。親に n 個以上のノードが含まれていない場合、このパスは無効になります。

パスの例	動作
/Patient/Episode[~$3]	<Patient> 内に存在する 3 番目のノードの前に、新しい <Episode> 要素を挿入します。親に 3 つ以上のノードが含まれていない場合、このパスは無効になります。
/Patient/element(Episode)[~$3]	指定できません。element() 関数は、要素の位置に対してのみ機能します。
/Patient/[~3]	指定できません。挿入する要素の種類に関する情報がないためです。
/Patient/element()[~3]	複数の理由によって指定できません。上記の項目を参照してください。

基本的なパスの参照先の項目のインスタンスを、指定された親の (新しい) 最後のノードとして追加します。上書きは一切されません。

パスの例	動作
/Patient/Episode[~]	<Patient> 内の最後のノードとして、新しい <Episode> 要素を追加します。パスが中間の存在しない要素を参照する場合、InterSystems IRIS によってそれらが作成されます。
/Patient/element(Episode)[~]	<Patient> 内の最後のノードとして、<Episode> 要素を追加します。パスが中間の存在しない要素を参照する場合、このパスは無効になります。
/Patient/[~]	指定できません。追加する要素の種類に関する情報がないためです。
/Patient/element()[~]	指定できません。追加する要素の種類に関する情報がないためです。

例えば、以下はデータ変換におけるコード要素の一部を示しています。

ObjectScript

```
set status=target.SetValueAt("orange","/Patient/FavoriteColors/Color[~]")
if 'status {do $system.Status.DisplayError(status) quit}
set status=SetValueAt("pink","/Patient/FavoriteColors/Color[~]")
if 'status {do $system.Status.DisplayError(status) quit}
```

これによって、2 つの <Color> が新しい子要素として <FavoriteColors> 要素に追加されます。<FavoriteColors> 要素が存在しない場合は、InterSystems IRIS によって作成されます。

"パス修飾子のサマリ" も参照してください。

6.7 element() 関数の使用

値を取得または設定する場合は、element() 関数が使用できます。

6.7.1 値を取得する際の element() の使用

構文	動作
element_reference/element()	指定された要素の最初の子要素を返します。
element_reference/element()[n]	指定された要素の n 番目の子要素を返します。
element_reference/element()[-]	指定された要素の最後の子要素を返します。

6.7.2 値を設定する際の element() の使用

構文	動作
parent_element/element(element_name)[~n]	指定された親の n 番目の子要素の直前に、(element_name 引数で指定された) 要素を挿入します。指定された要素に n 個以上の子要素が含まれていない場合、このパスは無効になります。
parent_element/element(element_name)[~]	(element_name 引数で指定された) 要素を、指定された親の最後のノードとして追加します。

6.8 要素の位置の取得

要素の位置を取得するには、以下の構文を使用します。

構文	戻り値
element_reference/position()	指定された要素のその親内での要素の位置。
element_reference/node-position()	指定された要素のその親内でのノードの位置。ノードの位置については、要素だけでなく、すべての種類のノードが考慮されます。

6.9 要素の数の取得

要素の数を取得するには、以下の構文を使用します。

構文	戻り値
・ element_reference/[*] ・ element_reference/count()	指定された親内の子要素の数。

構文	戻り値
· parent/element_name[*] · parent/element_name.count()	指定された親内の指定された名前を持つ要素の数。(前述の一連のパスとは対照的に) 要素の名前の後にスラッシュがないことに注意してください。
· element_reference/[$*] · element_reference/node-count()	指定された要素の子ノードの数。
· element_reference/@[*] · element_reference/@.count()	指定された要素の属性の数。

/[*] 以外のすべての場合に、角かっこを省略することができます。InterSystems IRIS では、last() 関数 (count() 関数と同等) および node-last() 関数 (node-count() 関数と同等) もサポートされています。同様の last() 関数が用意されている XPATH に慣れている場合は、last() および node-last() を使用することもできます。

6.10 その他のメタデータへのアクセス

XML 仮想ドキュメントのその他のメタデータにアクセスするには、以下の関数を使用します。これらの関数は、パスの最後でのみ使用できます。

対象機能	戻り値
/node-type()	指定されたノードのタイプ。この関数によって、以下のいずれかの値が返されます。 · root · element · text · comment · instruction
/name()	指定されたノードの完全名。例 :s01:Patient
/local-name()	指定されたノードのローカル名。例 :Patient
/prefix()	指定されたノードのネームスペース接頭語。例 :s01
/namespace-uri()	指定されたノードが属しているネームスペースの URI。例 :www.myapp.org
/prefixes()	指定された要素の範囲内の、すべてのネームスペース接頭語およびそれぞれに対応する URI。この情報は、カンマ区切りリストとして返されます。各リスト項目は、ネームスペース接頭語と、それに続く等号記号 (=) および URI で構成されます。デフォルトのネームスペース URI は、接頭語なしで最初にリストされます。例 : =http://tempuri.org,s01=http://myns.com

6.11 パス修飾子のサマリ

以下の表に、DOM スタイル・パスのパス修飾子のサマリを示します。

パス修飾子	Uses	この修飾子が含まれているパスを使用できるメソッド	SetValueAt() と同時に使用する場合、(必要に応じて) パディングを追加するか？
[n]	n 番目のインスタンスの取得または設定。	GetValueAt() および SetValueAt()	はい
/[n]	n 番目の子要素の取得。	GetValueAt()	なし
[~n]	n 番目のインスタンスの挿入。	SetValueAt()	はい
	インスタンスの追加。	SetValueAt()	いいえ
[$n]	n 番目のノード位置でのインスタンスの取得または設定。	GetValueAt() および SetValueAt()	いいえ
[~$n]	n 番目のノード位置でのインスタンスの挿入。	SetValueAt()	いいえ

6.12 ネームスペースを使用するドキュメントのバリエーション

ドキュメントで XML ネームスペースが使用されている場合、ネームスペース内の要素または属性ごとに、そのパスのセクションをネームスペース接頭語とそれに続くコロン (:) を含めるように変更する必要があります。ネームスペース接頭語は、以下のいずれかです。

- 対応する XML スキーマをロード済みの場合は、"XML ネームスペース・トークン" で説明しているとおりに、ネームスペース・トークンを使用します。例 :element_name ではなく、$2:element_name を使用
- XML スキーマをロードしていない場合、ドキュメントで指定されているとおりのネームスペース接頭語を使用します。例 :s01:Patient
- ネームスペースを無視するには、ワイルドカード * を使用します。例 :*:Patient

別のオプションとして、ドキュメント内のすべてのネームスペースを無視することもできます。そのためには、パスを / でなく、ワイルドカード *:/ で始めます。

例 :*:/Patient/@MRN

パスの値を設定する場合は、そのパスにワイルドカードを使用することはできません。

注釈　DTL の出力ドキュメントは、入力ドキュメントと同じネームスペース接頭語を使用するとは限りません。ネームスペースは同じであっても、接頭語は生成されます。XML 標準によると、接頭語の選択は重要ではありません。

6.13 ターミナルでの DOM スタイル・パスのテスト

特に、構文に精通している場合は、仮想ドキュメント・プロパティ・パスをビジネス・プロセスやデータ変換などで使用する前にターミナルでテストできると便利です。DOM スタイル XML パスに対してこれを行うには、ターミナルまたはテスト・コードで次の手順を実行します。

1. 適切な XML ドキュメントのテキストを含む文字列を作成します。
2. EnsLib.EDI.XML.Document の ImportFromString() メソッドを使用してこの文字列から XML 仮想ドキュメントのインスタンスを作成します。
3. このインスタンスの GetValueAt() および SetValueAt() メソッドを使用します。

以下のメソッドがこれらのステップを実行します。

```
ClassMethod TestDOMPath()
{
    set string="<Patient xmlns='http://myapp.com'>"
    _"<Name>Jolene Bennett</Name>"
    _"<Address><Street>899 Pandora Boulevard</Street></Address>"
    _"</Patient>"
    set target=##class(EnsLib.EDI.XML.Document).ImportFromString(string,.status)
    if 'status {do $system.Status.DisplayError(status)  quit}

    set pathvalue=target.GetValueAt("/Patient/Name",,.status)
    if 'status {do $system.Status.DisplayError(status)  quit}
    write pathvalue
}
```

このメソッドの出力を以下に示します。

```
SAMPLES>d ##class(Demo.CheckPaths).TestDOMPath()
Jolene Bennett
```

GetValueAt() のその他のオプションは、"pFormat 引数" を参照してください。

7

XML 仮想ドキュメントのデータ変換の定義

ここでは、ルール・セットで使用できるように、XML 仮想ドキュメントのデータ変換 (特に、DTL ベースの変換) を作成する方法を示します。

7.1 データ変換の作成

XML 仮想ドキュメントのデータ変換を作成するには、以下の操作を行います。

1. 必要に応じて、1 つまたは複数の適用可能な XML スキーマを InterSystems IRIS® にロードします。

   ″InterSystems IRIS への XML スキーマのロード″ を参照してください。

2. ″DTL 変換の開発″ 内の説明に従って、管理ポータルまたは IDE で DTL エディタを使用します。

3. データ変換内で、以下の値を使用します。

   - **[ソースクラス]** および **[ターゲットクラス]** で、InterSystems IRIS で XML 仮想ドキュメントを示すためのクラスである **[EnsLib.EDI.XML.Document]** を使用します。
   - **[ソース・ドキュメント・タイプ]** で、必要に応じてそのメッセージで想定される XML タイプを選択します。InterSystems IRIS にロードした XML スキーマのいずれかから XML タイプを選択します。

     スキーマをロードしていない、またはスキーマを使用したくない場合は、この値を空白のままにしてください。
   - **[ターゲット・ドキュメント・タイプ]** で、必要に応じて異なる XML タイプを選択するか、値を削除します。

     [ソース・ドキュメント・タイプ] でユーザが選択した値がある場合はそれを使用して、**[ターゲット・ドキュメント・タイプ]** が初期化されます。

4. 前のトピックで説明した XML プロパティ・パスを使用して、通常どおりデータ変換内にアクションを作成します。これには、以下の 2 つの基本的なシナリオがあります。

   - スキーマをロードし、ソースおよびターゲットの各ドキュメント・タイプを指定済みの場合は、DTL エディタに各ドキュメント構造がツリーとして表示されます。これにより、ドラッグ・アンド・ドロップすることで変換を作成できます。InterSystems IRIS によって、スキーマ依存パスを使用するアクションが作成されます。何らかの理由で DOM スタイル・パスをそれらのアクションで使用する必要がある場合は、そのように編集することもできます。
   - ドキュメント・タイプを指定しない場合、ドキュメント構造はツリーとして表示されません。その場合、アクションを手動で追加および編集する必要があります。使用できるのは、DOM スタイル・パスのみです。

   いずれの場合も、より複雑な処理をサポートするためにコード要素を追加できます。

データ変換を保存してコンパイルすると、ルール・セットで使用できるようになります。これについては、"XML 仮想ドキュメントのルール・セットの定義" を参照してください。

7.2 XML 仮想ドキュメントの使用可能な割り当てアクション

XML 仮想ドキュメントに対して、InterSystems IRIS では以下の割り当てアクションがサポートされています。

- [set] ― 値を設定します。ターゲット要素のタイプが "any" の場合は、テキストに XML マークアップを含めることができます。XML マークアップは適切に構成される必要がありますが、何らかのスキーマに対して検証されることはありません。

  ターゲット要素をソース・プロパティ・パスの値に設定する場合は、ソース・プロパティ・パスが存在している必要があります。そうでなければ、InterSystems IRIS によってターゲットが設定されず、エラーが返されます。これは、**[存在しないソースセグメントとプロパティを無視する]** 設定を有効にすることにより、必要に応じて無視できます。詳細は、"変換詳細の指定" を参照してください。
- [append] ― ターゲット要素内のサブノードの後に新しい値を追加します。
- [clear] ― ターゲットのテキスト・コンテキストは消去しますが、その要素およびすべての子は保持します。または、ターゲットが属性の場合、このアクションによってその値は消去されますが、属性は保持されます。
- [remove] ― ターゲット要素または属性を削除します。

[insert] はサポートされていません。

7.3 コードの使用

より複雑な処理をサポートするためにコード要素を追加する必要がある場合は、source および target 変数の GetValueAt() と SetValueAt() メソッドを直接呼び出します。EnsLib.EDI.XML.Document の場合、これらのメソッドは以下のようになります。

GetValueAt()

```
method GetValueAt(pPropertyPath As %String,
                  pFormat As %String,
                  Output pStatus As %Status) as %String
```

説明 :

- pPropertyPath は、前述したように、XML プロパティ・パスです。
- pFormat は、返される文字列の形式を制御する一連のフラグです。次の項を参照してください。
- pStatus は、成功または失敗を示すステータスです。

このメソッドによって、指定されたプロパティ・パスの現在の値、またはパスが有効でない場合は空の文字列が返されます。

SetValueAt()

```
method SetValueAt(pValue As %String,
                  pPropertyPath As %String,
                  pAction As %String = "set",
                  pKey As %String = "") as %Status
```

説明：

- pValue は、指定された XML プロパティ・パスの適切な値です。
- pPropertyPath は、前のトピックで説明したように、XML プロパティ・パスです。
- pAction は、"set"、"append"、"clear"、"remove" のいずれかです。詳細は、前の節を参照してください。
- pKey は、XML 仮想ドキュメントには使用されません。

このメソッドによって、指定されたプロパティ・パスが評価され、(パスが有効な場合は) そのパスで pValue および pAction を使用して値が変更されます。

重要　これらのメソッドによって返されたステータス値をチェックすると便利です。無効なパスを指定した場合や、許可されていないアクションを試行した場合、ステータスに具体的な情報が含まれます。特にこの情報はデバッグを行う際に役立ち、時間を節約することができます。

7.3.1 pFormat 引数

GetValueAt() の pFormat 引数は、返される文字列の形式を制御するオプション文字列です。この文字列には、以下の表に示す任意の文字の適切な組み合わせを含めることができます。

概要	形式設定に含める文字	特定の動作
改行と復改	w	テキストが含まれていないすべての要素の後に、Windows スタイルの復改と改行を追加します。
改行と復改	r	保存された改行と復改を使用します。このオプションは、オプションの w および n より優先されます。
改行と復改	n	テキストが含まれていないすべての要素の後に、新しい行 (改行) を追加します。w とは対照的に、このオプションは復改を追加しません。
インデント。これらのオプションは出力に新しい行が含まれている場合にのみ使用されます。	i	新しい行ごとに 4 つのスペースを使ってインデントします。
インデント	1 ～ 9 の整数	新しい行ごとにこの数のスペースを使ってインデントします。このオプションは前のインデント・オプションより優先されます。
インデント	t	新しい行ごとに 1 つのタブを使ってインデントします。このオプションは前のどちらのインデント・オプションより優先されます。
インデント	s	保存されたインデント空白文字を使用します。このオプションは前のインデント・オプションより優先されます。
属性の処理	a	要素内の属性をアルファベットで表現します。
属性の処理	q	可能な場合は、二重引用符 (一重引用符ではなく) で属性値を囲みます。
ネームスペースの処理	p	ネームスペース接頭語の出力を抑制します。
ネームスペースの処理	x	ネームスペース宣言の出力を抑制します。

概要	形式設定に含める文字	特定の動作
空の要素の処理	e	オープン・タグとクローズ・タグのペアを使用して空の要素ごとの出力を生成します。このオプションが設定されていない場合は、空の要素が単一の空タグとして出力されます。このオプションは、オプション g よりも優先されます。
空の要素の処理	g	空のタグがある出力の抑制
その他	c	標準的な出力。このオプションは、e、i、n、t、および w の各オプションよりも優先されます。
その他	f	要素の中身だけでなく、完全な要素（開始タグと終了タグの両方を含む）を生成します。
その他	l	InterSystems IRIS にロードされたスキーマの場所に関する情報を追加します。このオプションは f が使用されている場合にのみ有効です。
その他	o	XML エンティティをそのまま追加します。これらのエンティティに対して XML エスケープ処理は行われません。
その他	C(e)	特定の文字エンコーディングを宣言する XML ヘッダ行を生成します。e は UTF-8 などの文字エンコーディングの引用符抜きの名前です。e が空の場合は、アダプタで定義されているエンコードを使用します。e が ! で始まる場合は、出力ストリームのエンコーディングを強制します。これは、UTF-8 以外の文字セットで構成されるファイル・オペレーションに自動的に適用されることに留意してください。

前述したように、pFormat 引数はこれらの項目の組み合わせと一致させることができます。例えば、値の C(UTF-8)q を使用した場合は、送信ドキュメントが UTF-8 文字セットになり、属性が二重引用符で囲まれます。また、値の C(UTF-16)a を使用した場合は、送信ドキュメントが UTF-16 文字セットになり、属性がアルファベットで表現されます。

注釈　この情報は、XML ビジネス・オペレーションの [形式] 設定にも適用されます。

7.4 例 1 : ソース・ドキュメントの大部分をコピーする

ソース・ドキュメントの大部分をコピーするデータ変換を簡単に定義するには、データ変換ビルダで以下の操作を行います。

- [**変換**] タブで、[**作成**] ドロップダウン・リストから [copy] を選択します。

  これにより、デフォルトで、新しいドキュメントは元のドキュメントのコピーになります。

- 選択された要素または属性を部分的または完全に削除するアクションを定義します。そのようなアクションを定義するには、以下の操作を行います。

  1. [**アクション追加**] で、[clear] または [remove] をクリックします。
  2. クリアまたは削除したいターゲット・プロパティをダブルクリックします。
  3. [**値**] に任意の値を入力します。このフィールドは必要ですが、この場合は無視されます。

以下に、スキーマ依存パスを使用する例を紹介します。

```
Class Demo02.MyDTL Extends Ens.DataTransformDTL
{

XData DTL [ XMLNamespace = "http://www.intersystems.com/dtl" ]
{
<transform sourceClass='EnsLib.EDI.XML.Document' targetClass='EnsLib.EDI.XML.Document'
sourceDocType='Demo02:Patient' targetDocType='Demo02:Patient' create='copy' language='objectscript' >
<assign value='"this value is ignored"' property='target.{WorkAddress}' action='remove' />
<assign value='"this value is ignored"' property='target.{HomeAddress}' action='remove' />
</transform>
}

Parameter REPORTERRORS = 1;

}
```

このデータ変換では、ソース・ドキュメントがターゲットにコピーされた後、<WorkAddress> および <HomeAddress> 要素がターゲットから削除されます。

以下に、DOM スタイル・プロパティ・パスを使用する同等の例を紹介します。

```
Class Demo02A.MyDTL Extends Ens.DataTransformDTL
{

XData DTL [ XMLNamespace = "http://www.intersystems.com/dtl" ]
{
<transform sourceClass='EnsLib.EDI.XML.Document' targetClass='EnsLib.EDI.XML.Document'
create='copy' language='objectscript' >
<assign value='"this value is ignored"' property='target.{/Patient/WorkAddress}' action='remove' />
<assign value='"this value is ignored"' property='target.{/Patient/HomeAddress}' action='remove' />
</transform>
}

Parameter REPORTERRORS = 1;

}
```

この場合、このデータ変換では、ドキュメント・タイプが不要なために指定されていないことに注意してください。

7.5 例 2 :ソース・ドキュメントのいくつかの部分のみを使用する

ソース・ドキュメントのいくつかの部分のみを使用するデータ変換を簡単に定義するには、データ変換ビルダで以下の操作を行います。

- [**変換**] タブで、[**作成**] ドロップダウン・リストから [new] を選択します。

  これにより、デフォルトで、新しいドキュメントが空になります。

- 選択された要素または属性をコピーするアクションを定義します。そのようなアクションを定義するには、ソース・ドキュメント領域からターゲット・ドキュメント領域にドラッグ・アンド・ドロップします。この方法で追加する各アクションは、[set] アクションです。

以下に、スキーマ依存パスを使用する例を紹介します。

```
Class Demo05.MyDTL Extends Ens.DataTransformDTL
{

XData DTL [ XMLNamespace = "http://www.intersystems.com/dtl" ]
{
<transform sourceClass='EnsLib.EDI.XML.Document' targetClass='EnsLib.EDI.XML.Document'
sourceDocType='Demo05:Patient' targetDocType='Demo05:Patient' create='new' language='objectscript' >
<assign value='source.{MRN}' property='target.{MRN}' action='set' />
<assign value='source.{PrimaryCarePhysician}' property='target.{PrimaryCarePhysician}' action='set' />
</transform>
}

Parameter REPORTERRORS = 1;

}
```

このデータ変換では、MRN および PrimaryCarePhysician プロパティのみがソースからターゲットにコピーされます。

以下に、DOM スタイル・プロパティ・パスを使用する同等の例を紹介します。

```
Class Demo05A.MyDTL Extends Ens.DataTransformDTL
{

XData DTL [ XMLNamespace = "http://www.intersystems.com/dtl" ]
{
<transform sourceClass='EnsLib.EDI.XML.Document' targetClass='EnsLib.EDI.XML.Document'
create='new' language='objectscript' >
<assign value='source.{/Patient/MRN}' property='target.{/Patient/MRN}' action='set' />
<assign value='source.{/Patient/PrimaryCarePhysician}'
property='target.{/Patient/PrimaryCarePhysician}' action='set' />
</transform>
}

Parameter REPORTERRORS = 1;

}
```

7.6 例 3 : code および SetValueAt() を使用する

以下の例では、[code] アクション・タイプ、および DOM スタイル・パスが使用されています。これにより、ルート要素に属性と XML コメントが追加されます。

```
Class Demo06.MyDTL Extends Ens.DataTransformDTL
{

XData DTL [ XMLNamespace = "http://www.intersystems.com/dtl" ]
{
<transform sourceClass='EnsLib.EDI.XML.Document' targetClass='EnsLib.EDI.XML.Document'
create='copy' language='objectscript' >
<code>
 //this part adds an attribute to the document
 set path="/1/@NewAttribute"
 set status=target.SetValueAt("New attribute value",path)
 if 'status {do ##class(MyApp.Utils).Trace("Demo06.MyDTL","Error setting path: ",path)}

 //this part adds a comment to the document
 set path="/1/comment()"
 set status=target.SetValueAt("This is an XML comment",path)
 if 'status {do ##class(MyApp.Utils).Trace("Demo06.MyDTL","Error setting path: ",path)}



</code>
</transform>
}

Parameter REPORTERRORS = 1;

}
```

SetValueAt() メソッドによってエラーが返されたら、この変換ではユーティリティ・メソッドを使用して詳細が記録されます。

8

XML 仮想ドキュメントのルール・セットの定義

ここでは、ビジネス・プロセスで使用できるように、XML 仮想ドキュメントのルール・セットを作成する方法を示します。

ルール・セットを使用するようにビジネス・プロセスを構成するには、その **[ビジネスルール名]** 設定を指定します。"XML 仮想ドキュメントを処理するためのビジネス・プロセスの追加" を参照してください。

8.1 ルール・セットの作成

XML 仮想ドキュメントのルール・セットを作成するには、以下の操作を行います。

1. 必要に応じて、1 つまたは複数の適用可能な XML スキーマを InterSystems IRIS® データ・プラットフォームにロードします。

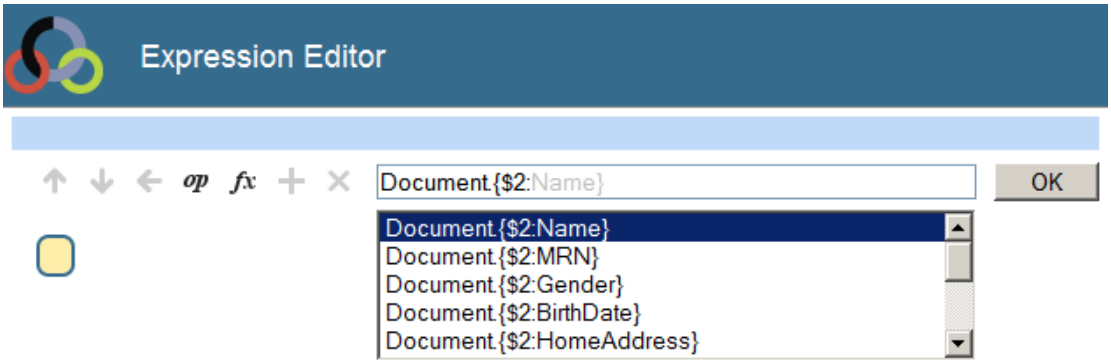
   "InterSystems IRIS への XML スキーマのロード" を参照してください。

2. "ビジネス・ルールの開発" の説明に従って、管理ポータルまたは IDE でルール・セット・エディタを使用します。
3. ルール・セットの基本的な定義については、**[タイプ]** で **[仮想ドキュメントのメッセージ・ルーティング・ルール]** を使用します。

   この選択によって、**[コンテキスト・クラス]** が EnsLib.MsgRouter.VDocRoutingEngine に設定されます。また、**[ルール・アシスト・クラス]** に EnsLib.MsgRouter.VDocRuleAssist が設定されます。
4. ルール・セットで何らかのルールの制約がある場合、以下の値を使用します。
   - **[メッセージ・クラス]** で、InterSystems IRIS で XML 仮想ドキュメントを示すための **[EnsLib.EDI.XML.Document]** を使用します。
   - **[スキーマ・カテゴリ]** で、必要に応じて InterSystems IRIS に事前にロードした XML スキーマを選択します。

     スキーマをロードしていない、またはスキーマを使用したくない場合は、この値を空白のままにしてください。
   - **[ドキュメント名]** で、必要に応じてそのスキーマで定義されているドキュメント・タイプを選択します。

     [スキーマ・カテゴリ] を指定していない場合は、この値を空白のままにしてください。
5. 前述したように、XML プロパティ・パスを使用して、通常どおりルールを作成します。これには、以下の 2 つの基本的なシナリオがあります。
   - スキーマをロードし、ターゲット・ドキュメント・タイプを指定済みの場合は、`Document` を入力し始めると式編集機能による支援が得られます。



これらのプロパティ・パスは、スキーマ依存パスであることに注意してください。ただし、何らかの理由で DOM スタイル・パスを使用する必要がある場合は、そのように編集することもできます。

- スキーマをロードしておらず、ドキュメント・タイプを指定済みでない場合は、このパスを手動で入力する必要があります。スキーマ依存パスまたは DOM スタイル・パスのいずれかを使用できます。

ルール・セットを保存してコンパイルすると、ビジネス・プロセスで使用できるようになります。

8.2 例

以下に、単純なルール・セットのクラス定義を紹介します。このルール・セットには DOM スタイル・パスを使用する 1 つのルールが含まれており、<Patient> ドキュメントの <MRN> 要素がチェックされます。このルールでは、戻り値に応じてメッセージが FileOut1 または FileOut2 にルーティングされます。この場合、このルールの制約は XML スキーマまたはタイプを参照しないことに注意してください。

```
Class Demo09.MyRules Extends Ens.Rule.Definition
{

Parameter RuleAssistClass = "EnsLib.MsgRouter.VDocRuleAssist";

XData RuleDefinition [ XMLNamespace = "http://www.intersystems.com/rule" ]
{
<ruleDefinition alias="" context="EnsLib.MsgRouter.VDocRoutingEngine">
<ruleSet name="" effectiveBegin="" effectiveEnd="">
<rule name="CheckMRN" disabled="false">
<constraint name="msgClass" value="EnsLib.EDI.XML.Document"></constraint>
<when condition="Document.{/$2:Patient/$2:MRN}=&quot;123456789&quot;">
<send transform="" target="FileOut1"></send>
<return></return>
</when>
<when condition="Document.{/$2:Patient/$2:MRN}!=&quot;123456789&quot;">
<send transform="" target="FileOut2"></send>
<return></return>
</when>
</rule>
</ruleSet>
</ruleDefinition>
}

}
```

9

XML 仮想ドキュメントの検索テーブルの定義

ここでは、XML 仮想ドキュメント用の検索テーブルの定義方法について簡単に説明します。

検索テーブル・クラスを使用するようにビジネス・サービスまたはビジネス・オペレーションを構成するには、そのビジネス・ホストの [**テーブルクラス検索**] 設定を指定します。"構成手順" を参照してください。

9.1 概要

XML 検索テーブル・クラスの EnsLib.EDI.XML.SearchTable は、XML ドキュメントのルート要素の名前にだけインデックスを付けます。

検索する項目を増やす場合は、サブクラスを作成できます。詳細は、"検索テーブル・クラスの定義" を参照してください。

注釈　InterSystems IRIS® では、検索テーブル・クラスを追加する前に受信したメッセージに対し、遡ってインデックスを割り当てることはありません。

9.2 例

以下に例を示します。

```
XData SearchSpec [ XMLNamespace = "http://www.intersystems.com/EnsSearchTable" ]
{
<Items>
   <Item DocType="MyApp:Patient" PropName="Gender" >{*:/Patient/Gender}</Item>
   <Item DocType="MyApp:Patient" PropName="MRN" >{*:/Patient/@MRN}</Item>
</Items>
}
```

10

XML 対応オブジェクトと XML 仮想ドキュメントの比較

XML メッセージでプロダクションを開発する場合、以下の構造のいずれかを使用して XML ドキュメントを含めることができます。

- XML 仮想ドキュメント
- XML 対応オブジェクト

入力 XML ドキュメントの要素のうちアクセスする必要があるものが少数の場合、XML 仮想ドキュメントを使用できますが、入力 XML ドキュメントのほとんどの要素にアクセスする必要がある場合、より効率的な XML 対応オブジェクトを選択する必要があります。これは特に、多くの要素を含むオブジェクトが変換によって処理される場合に当てはまります。XML 対応オブジェクトを処理するために、以下のビジネス・サービスとビジネス・オペレーションが用意されています。

- EnsLib.XML.Object.Service.FileService
- EnsLib.XML.Object.Service.FTPService
- EnsLib.XML.Object.Operation.FileOperation
- EnsLib.XML.Object.Operation.FTPOperation

XML 対応オブジェクトのビジネス・サービスは、XML ドキュメントを含むファイルを読み取り、これを 1 つ以上のオブジェクトに変換します。XML 要素を定義し、オブジェクトに変換するプロパティを指定します。XML ルート・ドキュメントに 1 つの要素が含まれている場合、サービスはこれを 1 つのオブジェクトに変換しますが、XML ルート・ドキュメントに一連の要素が含まれている場合、別々のオブジェクトに変換します。

XML 対応オブジェクトのサービスを使用するには、以下を実行します。

1. 処理する入力 XML ドキュメントの構造に一致するクラスを定義します。このクラスは、XML ドキュメント全体に対応するか、またはルート XML ドキュメント内の繰り返し要素を照合できます。XML スキーマ・ウィザードを使用してこのクラスを定義できます。オプションでこのクラスに NAMESPACE パラメータを定義できます。このパラメータは XML ネームスペースを指定します。
2. ビジネス・サービスの **[クラス名]** フィールドでクラス名を指定します。
3. オプションで、**[要素名]** フィールドで要素名を指定します。このフィールドを指定する場合、サービスは、ルート XML オブジェクト内でこの名前を持つ 1 つ以上の XML 要素を探します。要素が見つかるたびに、指定したクラスのインスタンスに変換されます。このフィールドを指定しない場合、サービスは、ルート・ドキュメントを指定したクラスと照合します。

4. オプションで [形式] パラメータを指定し、さらにオプションで [Null を無視] を選択します。[形式] パラメータの値には、"literal"、"encoded"、または "encoded12" のいずれかを指定できます。これらのパラメータは、%XML.Adaptor クラスの対応するパラメータを指定します。

XML 対応オブジェクトのビジネス・オペレーションは、オブジェクトを XML ドキュメントに変換し、そのドキュメントをファイルに書き込みます。XML クラスおよび要素についての情報の指定に加えて、オペレーションが %XML.Writer クラスを呼び出すときに使用するプロパティを指定できます。

オペレーションの以下のプロパティを指定します。

- [ルート要素名] – このプロパティを指定すると、これはルート要素名として使用されます。この要素を指定しないと、オペレーションは入力要素名を使用します。
- [ネームスペース] – XML ネームスペースを指定します (クラスが NAMESPACE プロパティを定義している場合を除く)。クラスが NAMESPACE プロパティを定義している場合、オペレーションは常にクラスで定義された XML ネームスペースを使用します。
- [想定されるクラス名] – XML 対応オブジェクトのクラス名。想定される名前が実際の名前と一致しない場合、%XML.Writer は xsi:type 属性を XML 要素に追加します。
- [インデント・タイプ] – %XML.Writer の対応するプロパティを指定します。[インデント・タイプ] は、XML 出力のインデントを実行するかどうかとインデントのタイプを指定します。
- [インデントの幅] – %XML.Writer の対応するプロパティを指定します。[インデントの幅] は、インデントに使用するインデント文字数を指定します。"タブ" のデフォルトは 1 です。"スペース" のデフォルトは 4 です。
- [文字セット] – %XML.Writer の対応するプロパティを指定します。文字セットは、XML 出力のエンコードに使用する文字セットです。デフォルトは出力先によって決まります。"UTF-8" は、ファイルまたはバイナリ・ストリームへの出力のデフォルトです。Unicode システムでは、"UTF-16" が文字ストリームおよび文字列への出力のデフォルトです。8 ビット・システムでは、ロケールのデフォルト文字セットは、文字ストリームおよび文字列への出力のデフォルト文字セットです。
- [XML 宣言なし] – %XML.Writer の対応するプロパティを指定します。[XML 宣言なし] が 1 (真) の場合、%XML.Writer は XML 宣言を書き込みません。 デフォルトは %XML.Writer による XML 宣言の書き込みありです。ただし、文字セットが指定されておらず、出力先が文字列または文字ストリームである場合を除きます。この条件が当てはまる場合には、XML 宣言を書き込みません。
- [実行時に Null を無視] – %XML.Writer の対応するプロパティを指定します。
- [要素が限定] – %XML.Writer の対応するプロパティを指定します。
- [属性が限定] – %XML.Writer の対応するプロパティを指定します。
- [デフォルトネームスペース] – %XML.Writer の対応するプロパティを指定します。
- [xmlns を抑制] – %XML.Writer の対応するプロパティを指定します。
- [形式] – %XML.Writer の対応するプロパティを指定します。
- [インラインを参照] – %XML.Writer の対応するプロパティを指定します。

XML 仮想ドキュメント・ビジネス・サービスとビジネス・オペレーション設定

このセクションでは、XML ビジネス・ホストの参照情報を提供します。

ルーティング・プロセス (EnsLib.MsgRouter.VDocRoutingEngine) の設定に関する詳細は、"仮想ドキュメント・ルーティング・プロセスの設定" を参照してください。

XML ビジネス・サービスに関する設定

XML 仮想ドキュメント・ビジネス・サービスの設定に関する参照情報を提供します。

概要

XML 仮想ドキュメント・ビジネス・サービスには次のような設定があります。

グループ	設定	参照先
基本設定	[ターゲット構成名]、[Docスキーマカテゴリ]	ビジネス・サービスに関する設定
追加設定	[テーブルクラス検索]、[検証]	このトピック内の節
追加設定	[リプライターゲット構成名]	このトピック内の節

残りの設定は、すべてのビジネス・サービスに共通しているか、ファイル・アダプタによって異なるかのどちらかです。詳細は、以下を参照してください。

- すべてのビジネス・サービスに含まれる設定
- ファイル受信アダプタに関する設定

リプライターゲット構成名

([ファイル] および [FTP] のみ) ビジネス・サービスがすべての XML 仮想ドキュメント応答メッセージを中継する必要があるプロダクション内の構成項目のカンマ区切りリスト。通常、リストには、1 つの項目が含まれますが、それ以上の項目数の場合があります。リストには、ビジネス・プロセスとビジネス・オペレーションの両方を指定できます。

[ターゲット構成名] と比較してください。

検証

デフォルトでは、XML 仮想ドキュメントの検証では、DocType が定義されているかどうかが確認されるだけです。XML 仮想ドキュメントに対する追加の検証を実行するには、EnsLib.MsgRouter.VDocRoutingEngine クラスをサブクラス化し、OnValidate メソッドをオーバーライドして、XML ドキュメントを検証するためのカスタム・コードを追加する必要があります。

ドキュメントを検証する場合は、すべてのデフォルト検証を抑止するゼロ以外の値を返します。ドキュメントが検証にパスした場合は、`pStatus` 内に 1 (`$$$OK`) を返して、問題がなかったことが示されます。ドキュメントが検証にパスしなかった場合は、`pStatus` 内にエラー・コードを返します。

XML ビジネス・オペレーションに関する設定

XML 仮想ドキュメント・ビジネス・オペレーションの設定に関する参照情報を提供します。

概要

XML 仮想ドキュメント・ビジネス・オペレーションには次のような設定があります。

グループ	設定	
基本設定	Format	このトピック内の節
追加設定	[テーブルクラス検索]	ビジネス・オペレーションに関する設定

残りの設定は、すべてのビジネス・オペレーションに共通しているか、ファイル・アダプタによって異なるかのどちらかです。詳細は、以下を参照してください。

- すべてのビジネス・オペレーションに含まれる設定
- ファイル送信アダプタに関する設定

Format

送信ドキュメントを構成する方法を指定します。この設定は空白のままにすることができます。その場合、デフォルトが使用されます。または、"pFormat 引数" に列挙された文字の適切な組み合わせを含む文字列を指定できます。

例えば、値の `C(UTF-8)q` を使用した場合は、送信ドキュメントが UTF-8 文字セットになり、属性が二重引用符で囲まれます。また、値の `C(UTF-16)a` を使用した場合は、送信ドキュメントが UTF-16 文字セットになり、属性がアルファベットで表現されます。