オブジェクトの XML への投影

Version 2024.1
2024-06-03

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

オブジェクトの XML への投影

InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

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

目次

テーブル一覧

1

オブジェクトの XML への投影の概要

このトピックでは、オブジェクトを XML に投影する方法を紹介し、その使用目的について説明します。

InterSystems IRIS® データ・プラットフォームでサポートされる XML 規格の詳細は、"InterSystems IRIS でサポートされる規格" を参照してください。

1.1 基礎編

オブジェクトを XML に投影するという表現は、オブジェクトを XML ドキュメントとして使用可能にする方法を定義することを意味します。オブジェクトを XML に投影するには、オブジェクトを定義するクラスのスーパークラス・リストに %XML.Adaptor を追加します。そのクラスで使用する他のオブジェクト・クラスも同様です。ただし、少数の例外があります。

また、この作業は、オブジェクトを定義したりそのクラスを XML 対応にするクラスの XML プロジェクションの定義とも呼びます。

1.2 動作内容

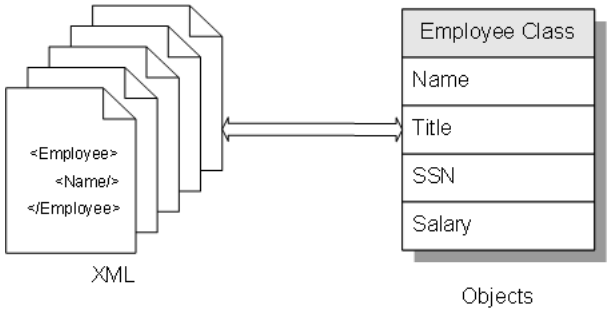
クラスの XML プロジェクションを定義すると、以下のようになります。

- システムでは、さらにクラスの INT コードを生成して、クラスのインスタンスを XML ドキュメントとして使用できるようにします(このコードはクラスのコンパイル時に生成されます。編集はしないでください)。
- クラスの各プロパティは、%XML.PropertyParameters から自動的に継承します。
- %XML.Adaptor により、クラスの XML 関連パラメータがクラスに追加されます。
- %XML.PropertyParameters により、プロパティの XML 関連パラメータがクラスのプロパティに追加されます。
- データ型プロパティにより、LogicalToXSD() メソッドと XSDToLogical() メソッドが定義されます。これによって、XML への出力または XML からの入力が行われたときにデータがどのように変換されるかが制御されます。

次に、既定のプロジェクションがニーズに適合しない場合は、必要に応じて、クラスの XML 関連パラメータを編集します。

1.3 プロジェクション・オプション

指定されたクラスに対する XML プロジェクションは、そのクラスのインスタンスがどのように XML ドキュメントに対応するか、また XML ドキュメントがどのようにそのクラスのインスタンスに対応するかを決定します。以下はその例です。



以下の方法を含めて、XML プロジェクションは多くの方法で制御できます。

- プロパティの投影先の構造を制御する。例えば、単純なプロパティは要素または属性として投影できます。それ以外は、これまでの例で示してきたように、投影できません。
- XML の要素名と属性名を制御する。
- 要素と属性が割り当てられた XML ネームスペースを制御する。
- InterSystems IRIS クラスが XML スキーマにどのようにマッピングされるかについて、詳細を制御する。

これらのパラメータの大半は、XML 対応クラスを使用するすべてのシナリオに影響します。このドキュメントに記載されているとおり、いくつかのパラメータは、特定のシナリオでのみ適用されます。

1.4 InterSystems IRIS における関連ツール

クラスの XML プロジェクションを定義すると、InterSystems IRIS のツールの多くにアクセスできます。それらのツールは、多くの実用的アプリケーションに適してします。以下のいずれの方法でも、これらのツールを使用してクラスを操作できます。

- そのクラスのオブジェクトを XML ドキュメントにエクスポートする。
- XML ドキュメントを InterSystems IRIS にインポートする。これによって、そのクラスの新規インスタンスが作成され、保存できます。
- そのクラスのオブジェクトを、Web サービスおよび Web クライアントの引数として使用する。
- XML スキーマを生成する。InterSystems IRIS では、クラスの XML タイプが暗黙的に定義され、そのクラスのオブジェクトを使用する際に、前述した方法のいずれかで検証に使用されます。

InterSystems IRIS には、InterSystems IRIS クラスに対応していない任意の XML ドキュメントを含めて、XML ドキュメントを操作するためのツールが用意されています。これらのツールでは、DOM、XPath、および XSLT がサポートされています。

InterSystems IRIS では、InterSystems IRIS SAX (Simple API for XML) パーサを使用して、着信と発信の XML ドキュメントを検証し、解析します。InterSystems IRIS SAX パーサは、標準 Xerces ライブラリを使用した、組み込みの SAX XML 検証パーサです。InterSystems IRIS SAX は、高性能なプロセス内コールイン・メカニズムを使用して、InterSystems IRIS プロセスと通信します。パーサを微調整したり、独自のカスタム SAX インタフェース・クラスを用意できます。

ここに記載されている XML ツールのいずれかを使用する方法の詳細は、"XML ツールの使用法" を参照してください。

Web サービスおよびクライアントの詳細は、"Web サービスおよび Web クライアントの作成" を参照してください。

1.5 XML ドキュメントに対して考えられるアプリケーション

XML ドキュメントは、以下のように、さまざまな実用的アプリケーションで使用できます。

- メッセージ・アプリケーションで、標準形式として使用できます。これには、業界標準のプロトコルのほかに、ローカルなソリューションも含まれます。
- アプリケーションとユーザ間の、データ変換のための標準形式として使用できます。
- 外部データ・ストレージに対する、標準的な表現として使用できます。これには、従来のデータベース・レコードや、ドキュメントのようなより複雑なコンテンツも含まれる場合があります。
- Web サービスと Web クライアントの間で送信される SOAP メッセージのペイロードとして使用できます。
- クラス定義における XData ブロックの内容として使用できます。"XData ブロック" を参照してください。

2
オブジェクトの XML への投影

ここでは、InterSystems IRIS® データ・プラットフォーム・オブジェクトを XML ドキュメントとして投影する方法を説明します。

このページで説明されているクラスとプロパティのパラメータ
・ ELEMENTTYPE
・ XMLPROJECTION
・ XMLFORMAT
・ XMLENABLED
・ XMLSUMMARY
・ XMLDEFAULTREFERENCE
・ GUIDENABLED
・ XMLIO

2.1 InterSystems IRIS オブジェクトの XML への投影

オブジェクトを XML に投影 (または、そのオブジェクトの XML プロジェクションを定義) するには、以下の手順を実行します。

1. オブジェクトを定義するクラスのスーパークラス・リストに %XML.Adaptor を追加します。

   以下はその例です。

   Class Definition

   ```
   Class MyApp.MyClass Extends (%Persistent, %XML.Adaptor)
   {
   //class details
   }
   ```

   この手順でクラスが XML 対応になります。

   また、投影するオブジェクトがシステム・クラスのインスタンスの場合は、代わりにサブクラスを作成して使用します。このサブクラスで、スーパークラス・リストに %XML.Adaptor を追加します。

また、InterSystems IRIS には、XML 対応の専用クラスが数多く、そのクラス・ライブラリに用意されています。これらには、%Net、%SOAP などのパッケージにおけるクラスが含まれます。これらの XML 対応クラスはすべて、%XML.Adaptor から継承されます。

2. %ListOfDataTypes、%ArrayOfDataTypes、%ListOfObjects、または %ArrayOfObjects のXML 対応サブクラスを作成する場合は、そのサブクラスで ELEMENTTYPE クラス・パラメータを指定します。以下はその例です。

   Class Definition

   ```
   Class MyApp.MyIntegerCollection Extends %ListOfDataTypes
   {
   Parameter ELEMENTTYPE="%Library.Integer";

   }
   ```

   ELEMENTTYPE には、コレクションで使用されるクラスの詳細なパッケージ名およびクラス名を指定します。そのパラメータを指定しないと、タイプは文字列と見なされます。

   この手順は、完全な XML スキーマが必要な場合にのみ必要です。

3. 次のセクションの説明に従って、XML 対応クラスの各プロパティに、適切であれば XML プロジェクションがあることを確認してください。

   ほとんどの場合、プロパティにオブジェクト値があれば、プロパティを定義するクラスを XML 対応にする必要があります。例外は、プロパティとして使用される際のコレクションとストリームです。詳細は次のセクションを参照してください。

   データ型クラスでは、作業は不要です。

4. 変更されたクラスをリコンパイルします。

これで、オブジェクトを XML ドキュメントに書き込むことなどができます。詳細は、"XML ツールの使用法" を参照してください。

2.1.1 Web メソッドで使用されるオブジェクトの例外

%ListOfDataTypes、%ListOfObjects、%ArrayOfDataTypes、%ArrayOfObjects を Web メソッドの入力や出力として使用する場合、XML 対応サブクラスの作成は不要です。ELEMENTTYPE を指定することは必要ですが、その処理はメソッド・シグニチャでできます。

ストリーム・クラスを Web メソッドの入力や出力として使用する場合、XML 対応サブクラスの作成は不要です。

詳細は、"Web サービスおよび Web クライアントの作成" を参照してください。

2.2 プロパティに XML へのプロジェクションがあることの確認

オブジェクトの各プロパティに XML プロジェクションがあることを確認するには、以下の手順を実行します。

- このオブジェクトの単純な (オブジェクトでない) 各プロパティに対しては、作業は不要です。InterSystems IRIS の各データ型には XML プロジェクションがあります。
- 各ストリーム・プロパティに対しては、作業は不要です。XML 対応クラスのプロパティとして使用される場合、XML ツールではストリーム・オブジェクトが特別に処理されます。
- コレクション以外のオブジェクト値プロパティの場合、参照されるクラスのスーパークラス・リストに %XML.Adaptor を追加します。

  これには、リレーションシップ・プロパティなどがあります。

- コレクションの詳細は、以下のサブセクションを参照してください。

InterSystems IRIS XML ツールで各種タイプのプロパティ値をどのように扱うかの詳細は、"値の変換の制御" を参照してください。プロパティ・タイプを XML タイプにどのように投影するかの詳細は、"XML スキーマへの投影の制御" を参照してください。

2.2.1 配列のリストで定義されるプロパティ

`Property PropName As List of classname` 構文や `Property PropName As Array of classname` 構文で定義されているプロパティごとに、以下の手順を実行します。

- classname がオブジェクト・クラスである場合、クラスを XML 対応にします。つまり、classname のスーパークラス・リストに **%XML.Adaptor** を追加します。
- classname がデータ型クラスの場合、必要な作業はありません。
- classname がストリーム・クラスであり、プロパティがリストの場合は、必要な作業はありません。ストリームは、文字列として XML に投影されます。

注釈　InterSystems IRIS では、ストリーム配列の XML 投影をサポートしていません。オブジェクトがストリーム配列として定義されているプロパティを持つ場合は、プロパティに `XMLPROJECTION="none"` を追加します。

以下はその例です。

Class Definition

```
Class MyApp.MyXMLObject Extends (%RegisteredObject, %XML.Adaptor)
{
Property MyListOfObjects As list Of MyApp.OtherXMLObject;

Property MyArrayOfObjects As array Of MyApp.OtherXMLObject;

Property MyListOfDT As list Of %String;

Property MyArrayOfDT As array Of %String;

Property MyListOfStreams As list Of %GlobalCharacterStream;

Property MyArrayOfStreams As array Of %GlobalCharacterStream(XMLPROJECTION = "NONE");
}
```

2.2.2 %ListOfDataTypes または %ArrayOfDataTypes のプロパティ

XML ツールでは、**%ListOfDataTypes** や **%ArrayOfDataTypes** のタイプのプロパティがコンテナとして自動的に投影されます。既定では、コンテナには文字列要素が含まれます。

適切な XML スキーマが必要な場合と、要素を文字列と想定することが適切でない場合は、コレクション・クラスのサブクラスを作成して使用します。サブクラスで、ELEMENTTYPE クラス・パラメータを指定します。以下はその例です。

Class Definition

```
Class MyApp.MyIntegerCollection Extends %ListOfDataTypes
{
Parameter ELEMENTTYPE="%Library.Integer";

}
```

ELEMENTTYPE には、コレクションで使用されるクラスの詳細なパッケージ名およびクラス名を指定します。

2.2.3 %ListOfObjects タイプまたは %ArrayOfObjects タイプのプロパティ

XML ツールでは、**%ListOfObjects** や **%ArrayOfObjects** のタイプのプロパティがコンテナとして自動的に投影されます。ただし、コレクションで使用されるクラスを XML 対応にする必要があります。

%ArrayOfObjects タイプのプロパティでは、コレクション内でストリーム・クラスを使用できません。

完全な XML スキーマが必要な場合、コレクションの要素タイプを指定する必要があります。そのためには、コレクション・クラスのサブクラスを作成して使用します。前のセクションで示したように、サブクラスで、ELEMENTTYPE クラス・パラメータを指定します。

2.2.4 例外

指定されたプロパティが XML に投影されない場合、参照されるクラスを XML 対応にする必要はありません。以下のプロパティは、XML に投影されません。

- プライベート・プロパティ
- 多次元プロパティ
- XMLPROJECTION パラメータを `"NONE"` として指定したプロパティ

`"ELEMENT"` などの適切な値に XMLPROJECTION プロパティ・パラメータを設定すると、プライベート・プロパティと多次元プロパティを投影できます。多次元プロパティの場合、プロジェクションに含めることができるのは最上位ノードのみです。

2.3 既定のプロジェクションの概要

既定の XML プロジェクションは、以下のとおりです。

- オブジェクトのインスタンスは、最上位レベルの XML 要素に対応します。
- 投影されるのはプロパティのみです。他のクラス・メンバは投影されません。

  また、プライベート・プロパティと多次元プロパティは無視されます。
- プロパティは、スタジオに表示された順に XML に投影されます。
- 指定された型のないプロパティは、文字列と見なされます。
- 各オブジェクト値プロパティは、囲まれる最上位レベルの XML 要素内にある XML 要素に対応します。

  そのプロパティは、この要素内で入れ子にされます。
- コレクションは、入れ子にされた要素として投影されます。これは、データ型のコレクションでもオブジェクトのコレクションでも変わりません。

  リストと配列では、下位レベルの詳細が少し異なります。
- リレーションシップは、リスト・プロパティと同じように扱われます。

  XML プロジェクションには、リレーションシップの片側のみが含まれます。両側を投影しようとすると、エラーが発生します。
- 文字ストリームは、文字列として投影されます。
- バイナリ・ストリームは、Base 64 のエンコードを行った文字列を使用して投影されます。

2.4 XML プロジェクションの例

このセクションでは、XML 対応クラスとその XML プロジェクションを示します。

2.4.1 XML 対応クラスの例

以下は、構造的プロパティの主要バリエーションがある XML 対応クラスを示します。

Class Definition

```
Class Basics.BasicDemo Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE = "mytypes";

Property SimpleProp As %String;

Property ObjProp As SimpleObject;

Property Collection1 As list Of %String;

Property Collection2 As list Of SimpleObject;

Property MultiDimProp As %String [ MultiDimensional ];

Property PrivateProp As %String [ Private ];

}
```

XMLTYPENAMESPACE パラメータは、このクラスで定義されたタイプのターゲット・ネームスペースを指定します。

`SimpleObject` クラスも XML 対応です。

Class Definition

```
Class Basics.SimpleObject Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE = "mytypes";

Property MyProp As %String;

Property AnotherProp As %String;

}
```

2.4.2 XML ドキュメントの例

以下は、`BasicDemo` クラスのインスタンスから生成された XML ドキュメントを示します。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<BasicDemo>
  <SimpleProp>abc</SimpleProp>
  <ObjProp>
    <MyProp>12345</MyProp>
    <AnotherProp>67890</AnotherProp>
  </ObjProp>
  <Collection1>
    <Collection1Item>list item 1</Collection1Item>
    <Collection1Item>list item 2</Collection1Item>
  </Collection1>
  <Collection2>
    <SimpleObject>
      <MyProp>12345</MyProp>
      <AnotherProp>67890</AnotherProp>
    </SimpleObject>
```

```
    <SimpleObject>
      <MyProp>12345</MyProp>
      <AnotherProp>67890</AnotherProp>
    </SimpleObject>
  </Collection2>
</BasicDemo>
```

2.4.3 スキーマの例

以下は、2 つのサンプル・クラスで使用されている XML タイプのネームスペースのスキーマを示します。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="BasicDemo">
    <sequence>
      <element minOccurs="0" name="SimpleProp" type="s:string"/>
      <element minOccurs="0" name="ObjProp" type="s01:SimpleObject" xmlns:s01="mytypes"/>
      <element minOccurs="0" name="Collection1" type="s02:ArrayOfCollection1ItemString"
xmlns:s02="mytypes"/>
      <element minOccurs="0" name="Collection2" type="s03:ArrayOfSimpleObjectSimpleObject"
xmlns:s03="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="SimpleObject">
    <sequence>
      <element minOccurs="0" name="MyProp" type="s:string"/>
      <element minOccurs="0" name="AnotherProp" type="s:string"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfCollection1ItemString">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="Collection1Item" nillable="true"
type="s:string"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfSimpleObjectSimpleObject">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="SimpleObject"
               nillable="true" type="s04:SimpleObject" xmlns:s04="mytypes"/>
    </sequence>
  </complexType>
</schema>
```

2.5 投影された XML ドキュメントのフォーマット・オプションの指定

XML ドキュメントの基本的な形式には、リテラルとエンコード (SOAP エンコード) の 2 つがあります。これらの形式の例については、"InterSystems XML ツールの概要" を参照してください。XML 対応クラスからデータをエクスポートしたりこのクラスにデータをインポートする際に、これらの形式のいずれかを指定します。

クラスに **%XML.Adaptor** を追加してコンパイルすると、InterSystems IRIS により、生成されたルーチンに追加のコードが記述されます。既定では、この追加コードで両方の形式がサポートされています。1 つの形式のみが必要な場合、もう一方の形式を抑制すると、生成されたコードの量を軽減できます。そのためには、そのクラスの XMLFORMAT パラメータを指定します。以下の値のいずれかを使用します (大文字と小文字は区別されません)。

- `"LITERAL"` ― このクラスではリテラル形式のみがサポートされます。
- `"ENCODED"` ― このクラスではエンコード形式のみがサポートされます (SOAP 1.1 と SOAP 1.2 の両方がサポートされています)。
- Null (既定) ― このクラスでは、リテラル形式とエンコード形式の両方がサポートされます。

2.6 単純なプロパティのプロジェクションが持つ形式の制御

単純なプロパティとは、その型がデータ型クラスであるプロパティやその型が宣言されていないプロパティのことです（指定された型のないプロパティは、文字列と見なされます）。

単純なプロパティの XML プロジェクションが持つ形式を制御するには、そのプロパティの XMLPROJECTION パラメータを設定します。以下の値のいずれかを使用します（大文字と小文字は区別されません）。

テーブル 2-1: 単純なプロパティに XMLPROJECTION が与える影響

XMLPROJECTION の値	非コレクション・プロパティへの影響
"ELEMENT"	プロパティは、要素として投影されます。これは非コレクション・プロパティに対する既定の設定です。
"ATTRIBUTE"	プロパティは、属性として投影されます。
"XMLATTRIBUTE"	プロパティは、接頭語 xml を持つ属性として投影されます。
"WRAPPED"	このプロパティは、下位要素を持つ要素として投影されます。下位要素の名前は、このプロパティのデータ型に基づいて決まります。
"CONTENT"	プロパティは、このクラスのプライマリ・コンテンツとして投影されます（つまり、プロパティのコンテンツは要素で囲まれることなく記述されます）。いずれのクラスでも、複数のプロパティにこの値を指定することはできません。
"NONE"	プロパティは、XML に投影されません。

XMLPROJECTION パラメータでは "COLLECTION" または "ELEMENTREF" の値も受け付けますが、お勧めはできないのでここでは説明しません。詳細は、%XML.PropertyParameters のクラス・ドキュメントを参照してください。

2.6.1 XMLPROJECTION の基本的なバリエーション

以下のクラスにより、"CONTENT" を除き、XMLPROJECTION のバリエーションをすべて使用する単純なプロパティを定義します。

Class Definition

```
Class xmlproj.SimpleProps Extends (%RegisteredObject, %XML.Adaptor)
{
Property Simple1 As %String (XMLPROJECTION="attribute");
Property Simple2 As %String (XMLPROJECTION="xmlattribute");
Property Simple3 As %String;
Property Simple4 As %String (XMLPROJECTION="element");
Property Simple5 As %String (XMLPROJECTION="wrapped");
Property Simple6 As %String (XMLPROJECTION="none");
}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<SimpleProps Simple1="The quick" xml:Simple2="brown fox">
  <Simple3>jumps</Simple3>
  <Simple4>over</Simple4>
  <Simple5>
    <string>the lazy</string>
  </Simple5>
</SimpleProps>
```

2.6.2 コンテンツとしてのプロパティの投影

"CONTENT" 値により、テキスト・コンテンツと属性はあるが下位要素はない単純な要素として、クラスを投影できます。

この値を使用するクラスの例を以下に示します。

Class Definition

```
Class xmlproj.SimpleContentProp Extends (%RegisteredObject, %XML.Adaptor)
{

Property Simple1 As %String(XMLPROJECTION = "content");

Property Simple2 As %String(XMLPROJECTION = "attribute");

Property Simple3 As %String(XMLPROJECTION = "element");

}
```

そのようなクラスのインスタンスをエクスポートすると、以下のようになります。

- "CONTENT" とマークされたプロパティに値がある場合、この値はクラスのコンテンツとしてエクスポートされます。属性として投影されたプロパティもエクスポートされます。その他のプロパティは無視されます。以下はその例です。

  XML

  ```
  <SimpleContentProp Simple2="other value">The quick brown fox jumps over the lazy
  dog</SimpleContentProp>
  ```

- "CONTENT" とマークされたプロパティが NULL の場合、このプロパティは無視され、XMLPROJECTION の値で指定されたように、他のプロパティはすべてエクスポートされます。以下はその例です。

  XML

  ```
  <SimpleContentProp Simple2="other value">
    <Simple3>yet another value</Simple3>
  </SimpleContentProp>
  ```

いずれのクラスでも、複数のプロパティで XMLPROJECTION を "CONTENT" と指定することはできません。また、この値の使用対象にできるのは、単純でリテラル値があり、オブジェクトのタイプがコレクションなどではないプロパティのみです。

2.7 オブジェクト値プロパティのプロジェクションが持つ形式の制御

オブジェクト値プロパティごとに、そのオブジェクト・クラスにある XML プロジェクションのオプションで制御されるように、既定の XML プロジェクションは下位要素や属性のある XML 要素 (オブジェクト自体を表す) で構成され、そのオブジェクトのプロパティを表します。例は、″XML プロジェクションの例″ を参照してください。

注釈　後述のセクションでは、特殊なオブジェクト値プロパティ(コレクション、リレーションシップ、およびストリーム)について説明します。

2.7.1 オブジェクト値プロパティのプロジェクションが持つ形式の指定

オブジェクト・プロパティの投影方法を制御するには、以下のように、そのプロパティの XMLPROJECTION パラメータを設定します。

テーブル 2-2: オブジェクト・プロパティに XMLPROJECTION が与える影響

XMLPROJECTION の値	コレクション・プロパティへの影響
"WRAPPED"	プロパティは、下位要素を持つ要素として投影されます。この要素は、オブジェクト・クラスに対応します。それぞれの下位要素は、そのクラスのプロパティに対応します。これは、オブジェクト・プロパティ(ストリーム以外)の既定値です。
"ELEMENT"	オブジェクト・クラスの各プロパティは要素として投影されます。親要素でラッピングされません。
"NONE"	プロパティは、XML に投影されません。
"ATTRIBUTE"、"XMLATTRIBUTE"、または "CONTENT"	コンパイル時エラー。

例えば、以下のクラスを考えてみます。

Class Definition

```
Class Basics.ObjectPropsDemo Extends (%RegisteredObject, %XML.Adaptor)
{

Property Object1 As SimpleObject(XMLPROJECTION = "wrapped");

Property Object2 As SimpleObject(XMLPROJECTION = "element");

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<ObjectPropsDemo>
  <Object1>
    <SimpleObject>
      <MyProp>abcdef</MyProp>
      <AnotherProp>qrstuv</AnotherProp>
    </SimpleObject>
  </Object1>
  <Object2>
    <MyProp>abcdef</MyProp>
    <AnotherProp>qrstuv</AnotherProp>
  </Object2>
</ObjectPropsDemo>
```

2.7.2 XMLSUMMARY の指定

クラスがプロパティとして使用される際に XML に投影するクラスのプロパティを簡単に指定できます。

- そのクラスでは、クラス定義で指定されているように大文字と小文字を区別して、XML に投影するプロパティのコンマ区切りリストとして XMLSUMMARY クラス・パラメータを指定します。XMLSUMMARY には擬似プロパティの ID および OID を記述できません。

以下のパラメータのいずれかまたは両方とも指定しない限り、このパラメータの効果はありません。

- 同じクラスでは、オプションで XMLDEFAULTREFERENCE を "SUMMARY" または "COMPLETE" (既定) として指定します。これらの値では、大文字と小文字は区別されません。"SUMMARY" オプションは、このクラスがプロパティとして使用される際に、XMLSUMMARY にリストされたプロパティのみをプロジェクションで使用することを意味します。"SUMMARY" オプションは、XML プロジェクションのあるプロパティをすべて使用することを意味します。

  これらの値では、大文字と小文字は区別されません。

- このクラスをプロパティとして使用するクラスでは、オプションで XMLREFERENCE プロパティ・パラメータを "SUMMARY" または "COMPLETE" (既定) として指定します。これらの値では、大文字と小文字は区別されません。これは、XMLDEFAULTREFERENCE クラス・パラメータをオーバーライドします。

  このプロパティ・パラメータを設定できるのは、プロパティがオブジェクト値を持つ場合のみです。

例えば、以下の Address クラスを考えてみます。

Class Definition

```
Class xmlsummary.Address Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLSUMMARY = "City,ZipCode";

Parameter XMLDEFAULTREFERENCE = "SUMMARY";

Property Street As %String;

Property City As %String;

Property State As %String;

Property ZipCode As %String;

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<Address>
  <Street>47 Winding Way</Street>
  <City>Middlebrook</City>
  <State>GA</State>
  <ZipCode>50291</ZipCode>
</Address>
```

すべてのプロパティが含まれていることに注意してください。

ここで、Address クラスをプロパティとして使用する別のクラスを考えてみます。

Class Definition

```
Class xmlsummary.Person Extends (%RegisteredObject, %XML.Adaptor)
{

Property Name As %String;

Property Address as Address;

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<Person>
  <Name>Penelope Farnsworth</Name>
  <Address>
    <City>Middlebrook</City>
    <ZipCode>50291</ZipCode>
  </Address>
</Person>
```

ここでは、Address クラスはこのクラスのプロパティであるため、パラメータとして XMLSUMMARY と XMLDEFAULTREFERENCE が使用され、プロジェクションで使用されるのは XMLSUMMARY にリストされたクラス・プロパティのみです。

"COMPLETE" オプションを使用すると、オーバーライドを強制適用できます。例えば、以下のクラスでは Address クラスをプロパティとして使用しますが、XMLREFERENCE を "COMPLETE" と指定します。

Class Definition

```
Class xmlsummary.Employee Extends (%RegisteredObject, %XML.Adaptor)
{

Property Name As %String;

Property Address As Address(XMLREFERENCE = "COMPLETE");

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<Employee>
  <Name>Malcom Winters</Name>
  <Address>
    <Street>770 Enders Lane</Street>
    <City>Middlebrook</City>
    <State>GA</State>
    <ZipCode>50293</ZipCode>
  </Address>
</Employee>
```

XML プロジェクションのあるすべてのプロパティが含まれます。この場合は、すべてのプロパティが含まれることを意味します。

2.7.3 オブジェクト識別子のみの投影

既定の方法でオブジェクト値プロパティを投影するのではなく、オブジェクトの識別子のみを投影できます。そのためには、XMLDEFAULTREFERENCE クラス・パラメータまたは XMLREFERENCE プロパティ・パラメータに、以下の値のいずれかを使用します。

- "ID" オプションを指定すると、ディスクに保存されている、オブジェクトの内部 ID のみが投影されます。プロパティは、どれも投影されません。例えば、以下のクラスを考えてみます。

  Class Definition

  ```
  Class xmlidentifiers.Person Extends (%Persistent, %XML.Adaptor)
  {

  Property Name As %String;

  Property PrimaryCarePhysician As Person (XMLREFERENCE = "ID");

  }
  ```

  以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<Person>
  <Name>Sam Smith</Name>
  <PrimaryCarePhysician>24</PrimaryCarePhysician>
</Person>
```

- "OID" オプションを指定すると、package.class,ID の形式でオブジェクトの OID のみが投影されます。プロパティは、どれも投影されません。PrimaryCarePhysician プロパティでこのオプションを使用すると、前述の Person オブジェクトは以下のように投影されます。

XML

```
<Person>
  <Name>Sam Smith</Name>
  <PrimaryCarePhysician>xmlidentifiers.Person,24</PrimaryCarePhysician>
</Person>
```

- "GUID" オプションを指定すると、可能であれば、オブジェクトの GUID (グローバルに一意な ID) のみが投影されます。GUIDENABLED クラス・パラメータが 1 である場合を除いて、オブジェクトの GUIDは NULL になります。Person クラスを以下のように再定義するとします。

Class Definition

```
Class xmlidentifiers.Person Extends (%Persistent, %XML.Adaptor)
{

Parameter GUIDENABLED=1;

Property Name as %String;

Property PrimaryCarePhysician As Person (XMLREFERENCE = "GUID");

}
```

この場合、このクラスのインスタンスの XML 表現は以下のようになることがあります。

XML

```
<Person>
  <Name>Sam Smith</Name>
  <PrimaryCarePhysician>D0F383EB-DB31-4C11-AD56-AA14EB37B734</PrimaryCarePhysician>
</Person>
```

注釈　プロパティ・パラメータ XMLREFERENCE の場合、"ID"、"OID"、"GUID" の各オプションを使用できるのは、そのオブジェクト値が永続オブジェクトの場合のみです。それ場合以外は、コンパイル時エラーとなります。

同様に、クラス・パラメータ XMLDEFAULTREFERENCE を "ID"、"OID"、または "GUID" に設定したクラスに、永続オブジェクトではない値を持つプロパティがある場合は、それらのプロパティのプロパティ・パラメータ XMLREFERENCE を明示的に "COMPLETE" または "SUMMARY" に設定する必要があります。

2.8 コレクション・プロパティのプロジェクションが持つ形式の制御

コレクション・プロパティの XML プロジェクションが持つ形式を制御するには、そのプロパティの XMLPROJECTION パラメータを以下のように設定します。

テーブル 2-3: コレクション・プロパティに XMLPROJECTION が与える影響

XMLPROJECTION の値	コレクション・プロパティへの影響
"WRAPPED"	このプロパティは、下位要素を持つ要素として投影されます。各下位要素はコレクションの項目に対応します。これはコレクション・プロパティに対する既定の設定です。
"ELEMENT"	コレクションの各項目は要素として投影されます。親プロパティでラッピングされません。
"NONE"	プロパティは、XML に投影されません。
"ATTRIBUTE"、"XMLATTRIBUTE"、または "CONTENT"	コンパイル時エラー。

後続のセクションでは、データ型のリストまたは配列であるプロパティの例を紹介します。オブジェクトのコレクションでは、投影された要素には、それらのオブジェクトの XML プロジェクションに応じて、さらに構造体が再帰的に存在する場合があります。

2.8.1 リスト・プロパティのプロジェクションが持つ形式の指定

以下のクラスにより、"WRAPPED" と "ELEMENT" の値を使用するコレクション・プロパティを定義します。

Class Definition

```
Class xmlproj.DataTypeColls Extends (%RegisteredObject, %XML.Adaptor)
{

Property Collection1 As list Of %String;

Property Collection2 As list Of %String (XMLPROJECTION="wrapped");

Property Collection3 As list Of %String (XMLPROJECTION="element");

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DataTypeColls>
  <Collection1>
    <Collection1Item>list item 1</Collection1Item>
    <Collection1Item>list item 2</Collection1Item>
  </Collection1>
  <Collection2>
    <Collection2Item>list item 1</Collection2Item>
    <Collection2Item>list item 2</Collection2Item>
  </Collection2>
  <Collection3>list item 1</Collection3>
  <Collection3>list item 2</Collection3>
</DataTypeColls>
```

Collection3 プロパティでは "ELEMENT" が使用されますが、XML プロジェクションでは、プロパティがリストであることは無視され、各リスト項目がクラスの別プロパティとして扱われます。

2.8.2 配列プロパティのプロジェクションが持つ形式の指定

配列の場合、それぞれの配列項目には値とキーの両方があり、それらの情報は両方とも XML で表現されている必要があります。キーは常に、XML 属性として要素内に投影されます。以下のクラスを考えてみます。

Class Definition

```
Class xmlproj.DataTypeArray Extends (%RegisteredObject, %XML.Adaptor)
{

Property ArrayProp As array Of %String;

}
```

以下は、このクラスのインスタンスの既定の XML 表現の例です。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DataTypeArray>
  <ArrayProp>
    <ArrayPropItem ArrayPropKey="1">apples</ArrayPropItem>
    <ArrayPropItem ArrayPropKey="2">bananas</ArrayPropItem>
    <ArrayPropItem ArrayPropKey="3">chocolate</ArrayPropItem>
  </ArrayProp>
</DataTypeArray>
```

XMLPROJECTION を **"ELEMENT"** に指定した場合、XML プロジェクションは代わりに以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<DataTypeArray>
  <ArrayProp ArrayPropKey="1">apples</ArrayProp>
  <ArrayProp ArrayPropKey="2">bananas</ArrayProp>
  <ArrayProp ArrayPropKey="3">chocolate</ArrayProp>
</DataTypeArray>
```

2.9 リレーションシップのプロジェクションが持つ形式の制御

リレーションシップは、使用されているコレクションの特性に応じて、他のプロパティと同じように XML に投影されます。

- 親子リレーションシップの場合 :
  - 親オブジェクトのリレーションシップ・プロパティはコレクション・プロパティです。具体的にはオブジェクトのリストになります。"コレクション・プロパティのプロジェクションが持つ形式の制御" を参照してください。
  - 子オブジェクトのリレーションシップ・プロパティはオブジェクト値プロパティです。

    このリレーションシップは、既定では XML に投影されません。
- 一対多のリレーションシップの場合 :
  - 単一オブジェクトのリレーションシップ・プロパティはコレクション・プロパティです。具体的にはオブジェクトのリストになります。
  - 他のオブジェクトのリレーションシップ・プロパティはオブジェクト値プロパティです。

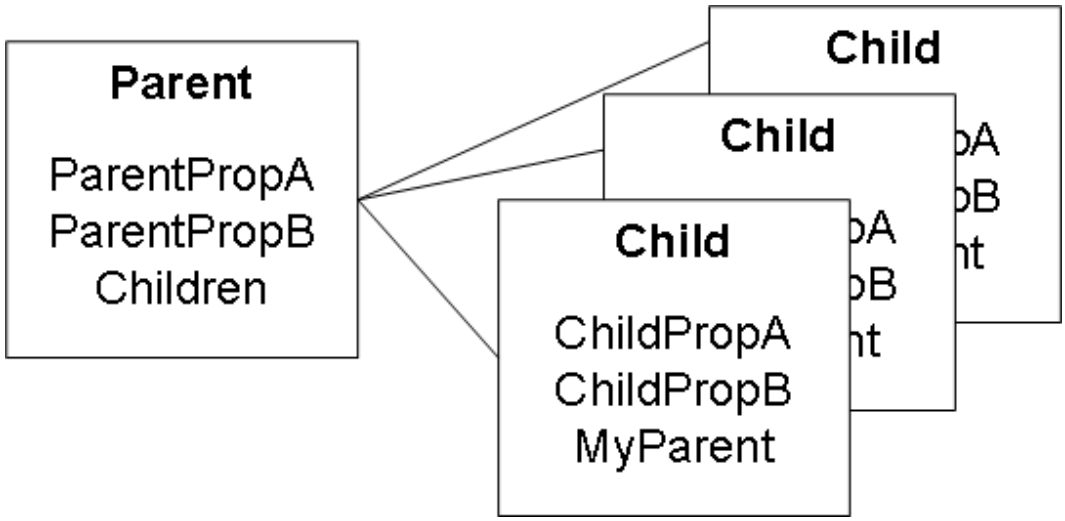
    このリレーションシップは、既定では XML に投影されません。

どの時点でも、リレーションシップの片側のみを投影できます。そうしないと、無限ループが発生するからです。投影のやり方を逆にするには、XMLPROJECTION プロパティ・パラメータを使用します。

この原則は、例を使用してうまく説明できます。

2.9.1 リレーションシップの既定のプロジェクション

まず、以下の 2 つのクラスについて考えてみます。



クラス `Parent` はクラス `Child` の親です。

クラス `Parent` には、リレーションシップ・プロパティ (`Children`) に加えて、2 つのプロパティ (`ParentPropA` と `ParentPropB`) があります。

同様に、クラス `Child` には、リレーションシップ・プロパティ (`MyParent`) に加えて、2 つのプロパティ (`ChildPropA` と `ChildPropB`) があります。

これらのクラスを XML に投影すると、既定では以下の結果が得られます。

- `Parent` の XML プロジェクションには、`ParentPropA`、`ParentPropB`、および `Children` の 3 つのプロパティのプロジェクションが含まれます。`Children` プロパティはその他のコレクションと同じように扱われます。つまり、既定では、入れ子にされた要素のセットとして投影されます。

  以下はその例です。

  XML

  ```
  <Parent>
    <ParentPropA>12345</ParentPropA>
    <ParentPropB>67890</ParentPropB>
    <Children>
      <Child>
        <ChildPropA>abc</ChildPropA>
        <ChildPropB>def</ChildPropB>
      </Child>
      <Child>
        <ChildPropA>ghi</ChildPropA>
        <ChildPropB>jkl</ChildPropB>
      </Child>
    </Children>
  </Parent>
  ```

- `Child` の XML プロジェクションには、`ChildPropA` および `ChildPropB` の 2 つのプロパティのプロジェクションが含まれます。`MyParent` プロパティは無視されます。

  以下はその例です。

XML

```
<Child>
  <ChildPropA>abc</ChildPropA>
  <ChildPropB>def</ChildPropB>
</Child>
```

一対多のリレーションシップでも同様です。具体的には、一側のオブジェクトには、リレーションシップ・プロパティのプロジェクションが含まれます。多側のオブジェクトには、リレーションシップ・プロパティのプロジェクションが含まれません。

2.9.2 代わりにリレーションシップの相手側を投影

両方のクラスのリレーションシップ・プロパティで XMLPROJECTION パラメータを指定することにより、代わりにリレーションシップの相手側を投影できます。次の例は、既定の例のバリエーションです。

Class Definition

```
Class Relationships2.Parent Extends (%Persistent, %XML.Adaptor)
{

Property ParentPropA As %String;

Property ParentPropB As %String;

Relationship Children As Child(XMLPROJECTION = "NONE") [ Cardinality = children, Inverse = MyParent ];

}
```

同様に、`Child` クラスは以下のようになります。

Class Definition

```
Class Relationships2.Child Extends (%Persistent, %XML.Adaptor)
{

Property ChildPropA As %String;

Property ChildPropB As %String;

Relationship MyParent As Parent (XMLPROJECTION="element") [ Cardinality = parent, Inverse = Children
];

}
```

これらのクラスを XML に投影すると、以下の結果が得られます。

- `Parent` の XML プロジェクションでは、`Children` プロパティは無視されます。

  XML

  ```
  <Parent>
    <ParentPropA>12345</ParentPropA>
    <ParentPropB>67890</ParentPropB>
  </Parent>
  ```

- `Child` の XML プロジェクションには、そのすべてのプロパティのプロジェクションが含まれます。

  XML

  ```
  <Child>
    <ChildPropA>abc</ChildPropA>
    <ChildPropB>def</ChildPropB>
    <MyParent>
      <ParentPropA>12345</ParentPropA>
      <ParentPropB>67890</ParentPropB>
    </MyParent>
  </Child>
  ```

2.10 ストリーム・プロパティのプロジェクションが持つ形式の制御

ストリーム・プロパティの場合、XMLPROJECTION のオプションは以下のようになります

テーブル 2-4: ストリーム・プロパティに XMLPROJECTION が与える影響

XMLPROJECTION の値	ストリーム・プロパティへの影響
"ELEMENT"	ストリーム・コンテンツは要素に格納されます。
"WRAPPED"	"ELEMENT" と同じように扱われます。
"CONTENT"	ストリーム・コンテンツは、"コンテンツとしてのプロパティの投影" の説明のとおりに投影されます。その他のプロパティについて、XMLPROJECTION パラメータは "NONE" にする必要があります。
"NONE"	プロパティは、XML に投影されません。
"ATTRIBUTE"	プロパティは、属性として投影されます。
"XMLATTRIBUTE"	プロパティは、接頭語 xml を持つ属性として投影されます。

このセクションでは、ストリーム投影の例を示します。

例えば、以下のクラスを考えてみます。

```
Class Basics.StreamPropDemo Extends (%Persistent, %XML.Adaptor)
{

Property BinStream As %Library.GlobalBinaryStream;

Property CharStream As %Library.GlobalCharacterStream;

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<StreamPropDemo>
  <BinStream>/9j/4AAQSkZJRgABAQEASABIAAD/2wBDAAUDBAQEAwUEBAQFBQUGBwwIBwcHBw8LCwkMEQ8SEhEP
ERETFhwXExQaFRERGCEYGh0dHx8fExciJCIeJBweHx7/2wBDAQUFBQcGBw4ICA4eFBEUHh4eHh4e
...
VcE/wkZ5wGJBH/joP50UVfQkqaS5dbi34EZtpJgPRlUkf1H402Fy9oIWHHlPj2K/Nn9cfhRRSGip
ZHzbmEPnEwZGGePu5/nj8qNJcpcrG4DxSuEkToDnPPtRRUyKGhPsuqlAxbypAhJ43A/y44q5HbNM
vmx3U9vuJDLG+ASCQW+pxRRSKP/Z</BinStream>
  <CharStream>This is a sample file.
This is line 2.
This is line 3.
This is line 4.

</CharStream>
</StreamPropDemo>
```

2.11 投影されるプロパティの可用性の制御

投影されたプロパティごとに、その使用をインポートのみ、エクスポートのみ、またはインポートとエクスポートの両方に指定できます。このためには、%XML.Writer クラスおよび %XML.Reader クラスのエクスポートおよびインポートのメソッドによるプロパティ処理方法を制御する XMLIO パラメータを設定します。このパラメータには以下の値のいずれかを指定できます (大文字と小文字は区別されません)。

- "INOUT" ー このプロパティは、エクスポートとインポートの両方に使用します。これは投影されるプロパティに対する既定の設定です。
- "IN" ー このプロパティは、インポートに使用しますが、エクスポートでは無視されます。
- "OUT" ー このプロパティは、エクスポートに使用しますが、インポートではエラーの原因になります。このプロパティに対応する XML 要素が XML ドキュメントに存在する場合、インポートではエラーが返されます。
- "CALC" ー このプロパティは、エクスポートに使用しますが、インポートでは無視されます。このプロパティに対応する XML 要素が XML ドキュメントに存在する場合、インポートではそれが無視されます。一般的に、この値は、計算されたプロパティに使用されます（値が他のプロパティの値を基に計算されているもの）。このため、エクスポートではすべての値を記録し、インポートでは計算された値を無視できます。

XML に投影されていないプロパティに対して、このパラメータは何の効果もありません。

2.12 プロジェクションの無効化

何らかの理由によるテスト時などに、XML 対応クラスが投影されないようにするには、そのクラスのパラメータ XMLENABLED を 0 に設定します。このパラメータの既定値は 1 です。

XMLENABLED を使用して、あるクラスが投影されないようにする場合、このクラスは、XML に投影されるどのクラスでもプロパティとして使用することはできません。XMLENABLED を 0 に設定することは、%XML.Adaptor をスーパークラス・リストから削除することと同じです。

2.13 %XML.Adaptor のメソッド

%XML.Adaptor のメソッドはお勧めできないので、ほとんど説明していません。代わりに、%XML.Writer や %XML.Schema の、ネームスペースに優れたサポートを提供するより強固なクラスを使用してください。詳細は、"XML ツールの使用法" を参照してください。

3

値の変換の制御

このトピックでは、オブジェクトを XML にエクスポートしたり XML をオブジェクトにインポートする際に、値がどのように変換されるかについて説明し、変換を制御するためのオプションについても説明します。

このトピックの XML 例はリテラル形式になります。

次のトピックでは NULL 値と欠落値について説明します。

このページで説明されているクラスとプロパティのパラメータ
· ESCAPE · CONTENT · XMLTIMEZONE · DISPLAYLIST · VALUELIST · XMLDEFAULTVALUE · XMLLISTPARAMETER · XMLSTREAMMODE

3.1 はじめに

XML 対応オブジェクトには通常、InterSystems IRIS® データ・プラットフォームのデータ型で定義されたプロパティがあります。それぞれのデータ型クラスでは、LogicalToXSD() メソッドと XSDToLogical() メソッドが定義されています。XML 出力がオブジェクトで要求されると、InterSystems IRIS XML ツールでは自動的に、プロパティごとに LogicalToXSD() メソッドを呼び出して、XML で使用できるようにデータを適切な形式に変換します。同様に、XML ドキュメントが入力として使用されると、InterSystems IRIS XML ツールでは XSDToLogical() メソッドを呼び出して、データを InterSystems IRIS 用の適切な形式に変換します。

例えば、%Binary データ型クラスでは、LogicalToXSD() メソッドにより、$SYSTEM.Encryption.Base64Encode() メソッドを使用して発信値を変換します。同様に、XSDToLogical() メソッドにより、$SYSTEM.Encryption.Base64Decode() メソッドを使用して着信値を変換します。

InterSystems IRIS クラスにはストリーム値プロパティを含めることもできますが、ストリーム・クラスでは LogicalToXSD() メソッドも XSDToLogical() メソッドも定義しません。その代わり、XML 対応クラスのプロパティとして使用される場合、XML ツールではストリーム・クラスが特別に処理されます。具体的には以下の手順を実行します。

- 文字ストリームは、文字列と同じように扱われます。次のセクションで説明するように、XML 特殊文字が存在するために必要な変更は別にして、既定では変更されません。
- InterSystems IRIS で XML にエクスポートすると、バイナリ・ストリーム・プロパティが文字列に Base 64 エンコードで変換されます (つまり、データがその方法でエンコードされてからエクスポートされます)。InterSystems IRIS で XML からインポートすると、逆が実行されます。

XML 対応のオブジェクトを使用する際、XML に値を直接投影できないとか他の理由で値を変換するとかいった、特殊な場合の検討が必要になる場合があります。

3.2 XML 特殊文字の処理

コンテキストに応じて、InterSystems IRIS XML サポートでは、タイプ文字列または文字ストリームのプロパティ内にアンド記号 (&) やその他の特定の文字を検出すると、それらの文字をエスケープします。

注釈　ESCAPE プロパティ・パラメータは、どの文字を特殊文字として認識するかを制御します。このパラメータは、"XML" (既定) または "HTML" (このドキュメントでは説明しません) のいずれかです。

それらの特殊文字では、CONTENT プロパティ・パラメータを設定することで、エスケープの実行方法を制御できます。以下に示すように、リテラル形式とエンコード形式では詳細が異なります。

テーブル 3-1: リテラル形式と SOAP エンコード形式のエスケープの形式

CONTENT の値 (大文字と小文字の区別なし)	リテラル形式の XML ドキュメント	SOAP エンコード形式の XML ドキュメント
"STRING" (既定)	CData	CData
"ESCAPE"	XML エンティティ	XML エンティティ
"ESCAPE-C14N"	XML エンティティ*	XML エンティティ*
"MIXED"	エスケープは行われません	CData

* "ESCAPE-C14N" の場合、エスケープ処理は XML Canonicalization 規格に従って行われます。主な違いは、キャリッジ・リターンが
 としてエスケープ処理されることです。

3.2.1 例

以下のクラスを考えてみます。

Class Definition

```
Class ResearchXForms.CONTENT Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLNAME = "Demo";

Property String1 As %String;

Property String2 As %String(CONTENT = "STRING");

Property String3 As %String(CONTENT = "ESCAPE");

Property String4 As %String(CONTENT = "MIXED");

}
```

String2 と String1 は、常に同様に処理されます。String2 では、CONTENT の既定値が使用されるからです。

このクラスのリテラル XML 出力は以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<Demo>
  <String1>value 1 & value 2

</String1>
  <String2>value 1 & value 2

</String2>
  <String3>value 1 &amp; value 2</String3>
  <String4>value 1 & value 2</String4>
</Demo>
```

SOAP エンコードの XML 出力は以下のように変わります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CONTENT xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <String1>value 1 & value 2

</String1>
  <String2>value 1 & value 2

</String2>
  <String3>value 1 &amp; value 2</String3>
  <String4>value 1 & value 2

</String4>
</CONTENT>
```

3.2.2 別のエスケープ回避方法

XML 特殊文字がエスケープ処理されないようにする別の方法があります。プロパティを特殊な XML タイプ、%XML.String、%XML.FileCharacterStream、または %XML.GlobalCharacterStream のいずれかとして定義できます。これらのデータ型クラスでは、CONTENT は "MIXED" になります。

使用されるシナリオでプロパティ値が有効であることを確認するのは、アプリケーション側で行う必要があります。%XML.String およびその他のクラスには、この検証機能はありません。

3.3 UTC タイム・ゾーン・インジケータの処理

XML 対応クラスでは、XML ドキュメントからインポートする際に UTC タイム・ゾーン・インジケータを使用するかどうかを指定できます。同様に、エクスポート時に UTC タイム・ゾーン・インジケータを含めるかどうかを指定できます。

そのためには、XMLTIMEZONE パラメータを指定します。以下の値のいずれかを使用します。

- "UTC" ― この場合、xsd:time または xsd:dateTime を含む要素をインポートすると、そのデータは UTC 時刻に変換されます。これが既定の動作です。

  XML スキーマの仕様に従い、InterSystems IRIS XML サポートではタイム・ゾーン・インジケータは純粋な期間として扱われ、EDT などの名前の付いたタイム・ゾーンはすべて無視されます。

- "IGNORE" ― この場合、xsd:time または xsd:dateTime を含む要素をインポートすると、UTC タイム・ゾーン・インジケータは無視されます。

エクスポート時には、UTC 時刻が常に使用されます。XMLTIMEZONE パラメータにより、UTC タイム・ゾーン・インジケータが含まれるかどうかが制御されます。

以下のクラスを考えてみます。

Class Definition

```
Class ResearchXForms.UTC Extends (%Persistent, %XML.Adaptor)
{

Parameter XMLNAME = "Demo";

Property Time1 As %Time;

Property Time2 As %Time(XMLTIMEZONE = "IGNORE");

Property TimeStamp1 As %TimeStamp;

Property TimeStamp2 As %TimeStamp(XMLTIMEZONE = "IGNORE");

}
```

このクラスの XML 出力は以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Demo>
  <Time1>17:52:06Z</Time1>
  <Time2>17:52:06</Time2>
  <TimeStamp1>1976-02-18T17:52:06Z</TimeStamp1>
  <TimeStamp2>1976-02-18T17:52:06</TimeStamp2>
</Demo>
```

3.4 DISPLAYLIST での値の投影

%String タイプ (またはサブクラス) のプロパティの場合、XML プロジェクションでは DISPLAYLIST プロパティを使用できます。

単純なプロパティにより、DISPLAYLIST パラメータと VALUELIST パラメータを指定できます。VALUELIST パラメータにより、プロパティに有効な値のリストを指定します。これによって列挙プロパティが定義されます。DISPLAYLIST パラメータも指定する場合があり、これによって、表示する対応値が指定されます。

既定では、XML プロジェクションではオブジェクトにある値が使用されます。その値は、VALUELIST で指定された値のいずれかです。%String タイプのプロパティの場合、XMLLISTPARAMETER パラメータは、プロジェクションに使用する別の値のリストがどのパラメータに含まれるかを示すようになっています。通常、これは "DISPLAYLIST" に設定します。例えば、以下のデータ型クラスを考えてみます。

Class Definition

```
Class xmldisplaylist.MyEnumString Extends %String
{

Parameter VALUELIST = ",a,b,c";

Parameter DISPLAYLIST = ",apples,bananas,chocolate";

Parameter XMLLISTPARAMETER = "DISPLAYLIST";

}
```

また以下のクラスも考えてみます。ここでは、その前のデータ型クラスが使用されます。

Class Definition

```
Class xmldisplaylist.Demo Extends (%RegisteredObject, %XML.Adaptor)
{

Property Property1 As MyEnumString;

Property Property2 As MyEnumString(DISPLAYLIST = ",red,green,blue", VALUELIST = ",r,g,b");

}
```

以下は、このクラスのインスタンスの XML 表現の例です。

XML

```
<Demo>
  <Property1>chocolate</Property1>
  <Property2>red</Property2>
</Demo>
```

一方、データ型クラスで XMLLISTPARAMETER パラメータを指定しない場合、XML 表現は以下のようになります。

XML

```
<Demo>
  <Property1>c</Property1>
  <Property2>r</Property2>
</Demo>
```

3.5 インポートされたストリーム・プロパティの行末の制御

XML からインポートする際、文字ストリームの各プロパティで、ストリームの行末を制御できます。そのためには、XMLSTREAMMODE プロパティ・パラメータを設定します。このプロパティ・パラメータには、以下のいずれかの値を設定できます (大文字と小文字は区別されません)。

- XMLSTREAMMODE が "block" (既定) の場合、正規化された XML データが、変更されずにストリームにコピーされます。ストリームの LineTerminator プロパティが $CHAR(10) に設定され、インポートが従来の改行シーケンス ($CHAR(10)、$CHAR(13)、$CHAR(13,10)) と互換性を持つようになります。
- XMLSTREAMMODE が "line" の場合、XML データはストリームの LineTerminator プロパティで指定された文字で区切った行に分けられます。"%Library.AbstractStream" を参照してください。

例えば、以下のデータがあるとします。

```
...
<Stream1>
this is a line
this is another line
this is another line


</Stream1>
...
```

このデータを %Stream.GlobalCharacter タイプの Stream1 プロパティを持つオブジェクトにインポートします。既定では、このプロパティでは XMLSTREAMMODE は "block" です。データをインポートした後、このプロパティは以下のデータを含みます。

```
this is a line
              this is another line
                              this is another line
```

このプロパティで XMLSTREAMMODE を "line" に設定し、データを再インポートすると、このプロパティは以下のデータを含みます。

```
this is a line
this is another line
this is another line
```

この場合、行末はストリーム・クラスの LineTerminator プロパティによって決定されます。このプロパティは、%Stream.GlobalCharacter で $char(13,10) となります。

3.6 既定の日付/時刻値の指定

%PosixTime、%TimeStamp、および %DateTime データ型クラスでは、日付が $zdatetimeh による有効性チェックに失敗した場合、XMLDEFAULTVALUE パラメータによって使用する値が指定されます。そのような場合、既定では NULL 文字列が使用され、これにより、XMLImport() を介してデータをインポートするとエラーが発生します。クラスに対して有効な値を指定します。

例えば、%TimeStamp や %DateTime に対して、1841 年以降の日付を YYYY-MM-DD HH:MM:SS.nnnnnnnnn 形式で指定します。例: `01.01.41 00:00:00`

3.7 XML への印字不能文字の投影

XML では、印字不能文字、特に ASCII 32 未満の文字は許可されません (キャリッジ・リターン、改行、およびタブは例外です)。

XML にプロパティを投影する必要があり、そのプロパティにこれら印字不能文字が含まれているとき、そのプロパティのタイプは %Binary または (同等の) %xsd.base64Binary とする必要があります。この値は、XML にエクスポートされるときに、Base 64 のエンコードに自動的に変換されます (またはインポートされるときに、Base 64 のエンコードから自動的に変換されます)。

4

空文字列および NULL 値の処理

このトピックでは、空文字列と NULL 値の処理方法について説明します。

このトピックの XML 例はリテラル形式になります。

このページで説明されているクラスとプロパティのパラメータ
· XMLUSEEMPTYELEMENT · XMLIGNORENULL · XMLNILNOOBJECT · XMLNIL

4.1 空文字列および NULL 値の既定のプロジェクション

以下のテーブルは、空文字列および NULL 値の既定の XML プロジェクションをまとめたものです。XML プロジェクションは、SQL プロジェクションと似ています。ここでは、比較のために、SQL プロジェクションも示しています。

テーブル 4-1: 空文字列および NULL 値の既定の SQL および XML プロジェクション

InterSystems IRIS 値	XML への既定のプロジェクション	SQL へのプロジェクション
`$char(0)`	空の要素または属性	SQL 空文字列
`""`	プロジェクションなし	SQL NULL 値

次のセクションでは、エクスポート時およびインポート時にこれらのプロジェクションがどのように機能するかを説明し、これらのプロジェクションを制御するためのオプションについて詳細に解説します。

4.2 値のエクスポート

以下のテーブルに、要素として XML に投影されるプロパティについて、XML 対応オブジェクトに含まれる空文字列と NULL 値を XML ドキュメントにエクスポートする方法を示します。

テーブル 4-2: 要素として XML に投影されるプロパティの空文字列と NULL 値のエクスポート

XML 対応クラスの詳細	"" になるプロパティ	$char(0) になるプロパティ
XMLIGNORENULL および XMLNIL の既定値を指定するクラス	エクスポートされた XML ドキュメントには、このプロパティに対応する要素が含まれない	エクスポートされたドキュメントには、このプロパティに対応する空要素が含まれる（"サブセクション" を参照）
XMLIGNORENULL=1 を指定するクラス	エクスポートされたドキュメントには、このプロパティに対応する空要素が含まれる（"サブセクション" を参照）	
XMLNIL=1（および XMLIGNORENULL が 1 以外）を指定するクラス	エクスポートされたドキュメントには、このプロパティと、この要素（xsi:nil="true" を指定）に対応する空要素が含まれる	

詳細は、属性として XML に投影されるプロパティの場合も同様です。

テーブル 4-3: 属性として XML に投影されるプロパティの空文字列と NULL 値のエクスポート

XML 対応クラスの詳細	"" になるプロパティ	$char(0) になるプロパティ
このテーブルで説明するパラメータの既定値を指定するクラス	エクスポートされた XML ドキュメントには、このプロパティに対応する属性が含まれない	エクスポートされたドキュメントには、このプロパティに対応する空の属性が含まれる
XMLIGNORENULL=1	エクスポートされたドキュメントには、このプロパティに対応する空の属性が含まれる。例：PropName=""	
XMLNIL=1（および XMLIGNORENULL が 1 以外）	エクスポートされた XML ドキュメントには、このプロパティに対応する属性が含まれない（既定のシナリオと同じ）	

4.2.1 空要素の形式の制御

既定では、InterSystems IRIS® データ・プラットフォームは、開始タグと終了タグで空要素を記述します。例えば、以下のようになります。

XML

```
<PropName></PropName>
```

InterSystems IRIS で自己終了型の空要素（上記と同等）を書き込むようにすることもできます。例えば、以下のようになります。

XML

```
<PropName />
```

このようにするには、XMLUSEEMPTYELEMENT クラス・パラメータに 1 を指定します。このパラメータの既定値は 0 です。

4.2.2 XMLIGNORENULL、XMLNIL、および XMLUSEEMPTYELEMENT の詳細

XMLIGNORENULL

XML へのエクスポート時 (および SOAP メッセージの書き込み時) に使用します。このパラメータは、NULL 文字列を無視する (エクスポートしない) かどうかを制御します。

このパラメータは、すべての XML 対応クラスのパラメータです。XMLIGNORENULL に指定できる値は、0 (既定値)、1、"INPUTONLY"、または "RUNTIME" (大文字/小文字の区別なし) です。

XMLIGNORENULL クラス・パラメータは、サブクラスによって継承されます。

XMLNIL

XML へのエクスポート時 (および SOAP メッセージの書き込み時) に使用します。このパラメータは、NULL 文字列に対する xsi:nil 属性の使用を制御します。

このパラメータは、すべての XML 対応クラスのクラス・パラメータおよびプロパティ・パラメータです。プロパティ・パラメータが優先されます。XMLNIL は、0 (既定値) または 1 です。

XMLNIL クラス・パラメータは、サブクラスに継承されません。XMLNIL プロパティ・パラメータは、継承されます。

XMLUSEEMPTYELEMENT

XML へのエクスポート時 (および SOAP メッセージの書き込み時) に使用します。このパラメータは、InterSystems IRIS で自己終了型の空のタグを書き込むかどうかを制御します。このパラメータは、以下の 2 つのシナリオで適用されます。

- クラスに対する XMLUSEEMPTYELEMENT が 1 の場合、このパラメータは、"" になる文字列値のプロパティに作用し、そのプロパティは要素として投影されます。これに該当するすべてのプロパティは、自己終了型の空要素としてエクスポートされます。
- クラスに対する XMLUSEEMPTYELEMENT が 1 の場合、プロパティが XML エクスポートに要素として出現しないときには、このパラメータは、クラスのインスタンスに対応する空要素の形式に作用します。この要素は、自己終了型の空要素としてエクスポートされます。

クラスの XMLUSEEMPTYELEMENT が 1 の場合、システムは、そのクラスに対して、わずかに多くのコードを生成します。また、そのクラスの XML 処理の効率も、わずかに低くなります。

4.3 値のインポート

XML から XML 対応オブジェクトへのインポート時に、空、NULL、または欠落している要素と属性を InterSystems IRIS が処理する方法を以下のテーブルに示します。

テーブル 4-4: 空、NULL、または欠落している要素と属性を含む XML ドキュメントのインポート

XML 対応クラスの詳細	インポートしたドキュメントに要素または属性が含まれない	インポートしたドキュメントでは、要素または属性が空になる	インポートしたドキュメントでは、要素は空になり、**xsi:nil="true"** を指定する
既定値に XMLNILNOOBJECT パラメータを指定するクラス	プロパティは設定されない	プロパティを `$char(0)` に設定する	· プロパティがリテラル値のプロパティの場合、プロパティは設定されない · プロパティがオブジェクト値の場合、プロパティは、参照されるクラスの新しいインスタンスに設定される。このインスタンスには、プロパティは設定されない
XMLNILNOOBJECT=1 を指定するクラス			プロパティは設定されない

5

XML の要素名と属性名の制御

InterSystems IRIS® データ・プラットフォーム・クラスおよびプロパティの名前と XML 要素および属性の名前との間には、既定の対応関係があります。この既定はオーバーライドできます。

複数の同名要素が XML ドキュメントに含まれる場合も、InterSystems IRIS でサポートされています。"特殊なトピック" を参照してください。

このトピックの XML 例はリテラル形式になります。

このページで説明されているクラスとプロパティのパラメータ
• XMLNAME • XMLTYPE • XMLITEMNAME • XMLKEYNAME

5.1 既定の XML の要素名と属性名

InterSystems IRIS の名前と XML の要素名や属性名との間にある既定の対応関係は、以下のとおりです。

- クラスの場合、それに対応する XML の要素名または属性名は短いクラス名と同じです。
- このクラスのプロパティの場合、それに対応する XML の要素名または属性名はプロパティ名と同じです。

  XML の要素または属性として投影されるかどうかはプロパティ定義によって決まります。"単純なプロパティのプロジェクションが持つ形式の制御" を参照してください。

  プロパティ名に引用符が含まれている場合、引用符は XML の要素名や属性名には含まれません。例えば、以下のプロパティを考えてみます。

  ```
  Property "Quoted Property" As %String;
  ```

  このプロパティをどのようにマッピングするかによって、このプロパティは要素 `<Quoted Property>` または属性 `Quoted Property` として投影されます。
- プロパティがリストまたは配列である場合、そのプロパティは自動的に下位要素で構成され、これらの各下位要素はそのリストまたは配列内の 1 つの項目に対応します。既定では、下位要素の名前はプロパティ名の末尾に `Item` を追加したものです。

- プロパティが配列である場合、下位要素にも、対応するキーを示す属性があります。既定では、その属性の名前はプロパティ名に Key を追加したものです。

5.2 最上位レベル要素として投影されたオブジェクトの要素名または属性名の制御

クラス・インスタンスを最上位レベル要素として投影すると、その XML 名は以下のように決定されます。

テーブル 5-1: 最上位レベル要素として投影されたオブジェクトのタグ

クラスの XMLNAME パラメータ	クラスの XMLType パラメータ	タグ (要素または属性)
指定あり	無視	XMLNAME の値
指定なし	指定あり	XMLTYPE の値
	指定なし	短いクラス名

XMLTYPE の詳細は、"XML スキーマへの投影の制御" を参照してください。

例えば、Sample.Address クラスのオブジェクトをエクスポートすると、既定ではこれらのオブジェクトは以下のように表されます。

XML

```
<Address>
  <Street>5064 Elm Street</Street>
  <City>Jackson</City>
  <State>PA</State>
  <Zip>27621</Zip>
</Address>
```

Sample.Address クラスの XMLNAME パラメータを指定したとします。以下はその例です。

Class Member

```
Parameter XMLNAME = "HomeAddress";
```

この場合の出力は以下のように変わります。

XML

```
<HomeAddress>
  <Street>5064 Elm Street</Street>
  <City>Jackson</City>
  <State>PA</State>
  <Zip>27621</Zip>
</HomeAddress>
```

これらのパラメータは、オブジェクトを XML にエクスポートするときにオーバーライドできます。詳細は、"XML ツールの使用法" を参照してください。

5.3 単純なプロパティのタグの制御

XML 対応オブジェクトでは、それぞれの単純なプロパティは、マッピングの方法に応じて、XML の要素または属性として投影されます。いずれの場合も、既定では InterSystems IRIS プロパティ名が XML の要素名または属性名として使用されます。プロパティに別の XML 名を指定するには、そのプロパティの XMLNAME パラメータで指定します。

以下はその例です。

```
Property Zip As %String (XMLNAME = "PostalCode");
```

前述の例で見た出力は以下のように変わります。

XML

```
<HomeAddress>
  <Street>5064 Elm Street</Street>
  <City>Jackson</City>
  <State>PA</State>
  <PostalCode>27621</PostalCode>
</HomeAddress>
```

プロパティが今度は別の InterSystems IRIS オブジェクト・クラスに入ると、XML プロジェクションではそのクラスのクラス名と XMLNAME パラメータが無視されることに注意してください。例えば、`Person` クラスに `Address` という名前のプロパティがあり、それが `Address` クラスへの参照であるとします。`Person` オブジェクトの投影は以下のようになります。

XML

```
<Person>
 <Name>Zevon,Juanita Q.</Name>
 <DOB>1986-08-18</DOB>
 <Address>
   <Street>5064 Elm Street</Street>
   <City>Jackson</City>
   <State>PA</State>
   <Zip>27621</Zip>
 </Address>
</Person>
```

`<Address>` 要素の名前は、`Person` オブジェクト内の対応するプロパティの名前によって決まります。これは、アドレス・オブジェクトが、直接インポートまたはエクスポートされるオブジェクトではなく、インポートまたはエクスポートされるオブジェクトのプロパティであるからです。

他のプロパティと同様に、この名前は、そのプロパティの XMLNAME パラメータを指定することでオーバーライドできます。以下はその例です。

```
Property Address As MyApp.Address (XMLNAME = "EmployeeAddress");
```

前述の例で見た出力は以下のように変わります。

XML

```
<Person>
 <Name>Zevon,Juanita Q.</Name>
 <DOB>1986-08-18</DOB>
 <EmployeeAddress>
   <Street>5064 Elm Street</Street>
   <City>Jackson</City>
   <State>PA</State>
   <Zip>27621</Zip>
 </EmployeeAddress>
</Person>
```

5.4 リスト型プロパティの要素名と属性名の制御

注釈　このセクションは、XMLPROJECTION を "ELEMENT" と指定するコレクション・プロパティには適用されません。そのようなプロパティの場合、各リスト項目はクラスの別のプロパティとして扱われます。"コレクション・プロパティのプロジェクションが持つ形式の制御" を参照してください。

XML 対応オブジェクトの場合、下位要素を持つ要素にリスト型プロパティが投影され、これらの各下位要素はそのリストの 1 つの項目に対応します。プロパティ ColorOptions を持つ InterSystems IRIS オブジェクトがあり、このプロパティが "Red"、"Green"、および "Blue" という 3 つの文字列のリストと等しいとします。既定では、このプロパティは以下の XML に対応します。

XML

```
<ColorOptions>
  <ColorOptionsItem>Red</ColorOptionsItem>
  <ColorOptionsItem>Green</ColorOptionsItem>
  <ColorOptionsItem>Blue</ColorOptionsItem>
</ColorOptions>
```

ここでは、リストの 1 項目に対応する ColorOptionsItem という下位要素が示されています。この下位要素の名前は以下のように決定されます。

テーブル 5-2: リスト項目のタグ

プロパティの XMLITEMNAME パラメータ	プロパティの XMLNAME パラメータ	タグ（要素または属性）
指定あり	無視	XMLITEMNAME の値
指定なし	指定あり	末尾に Item が連結された XMLNAME の値
	指定なし	リスト項目がデータ型プロパティに対応する場合、タグは末尾に Item が連結されたプロパティ名になります。リスト項目がオブジェクト・クラスに対応する場合、タグは短いクラス名になります。

配列の項目にも同じ考え方が適用されます。配列のキーは、別の方法で処理されます。

5.5 配列型プロパティの要素名と属性名の制御

注釈　このセクションは、XMLPROJECTION を "ELEMENT" と指定するコレクション・プロパティには適用されません。そのようなプロパティの場合、各配列項目はクラスの別のプロパティとして扱われます。"コレクション・プロパティのプロジェクションが持つ形式の制御" を参照してください。

XML 対応オブジェクトでは、リスト・プロパティが投影される方法と基本的に同じ方法で、下位要素を持つ要素に配列型プロパティが投影され、これらの各下位要素はそのリストの 1 つの項目に対応します。前のセクションを参照してください。

各下位要素は、項目に関連付けられているキーを示す別の属性を持っています。この属性の名前を制御できます。

以下のサンプル・プロパティについて考えてみましょう。

```
Property Tools As %ArrayOfDataTypes;
```

例えば、(あるオブジェクト・インスタンスに対して) このプロパティが以下のような配列で構成されるとします。

- キー 845 で格納されている値 Hammer
- キー 1009 で格納されている値 Monkey wrench
- キー 3762 で格納されている値 Screwdriver

既定では、このプロパティは以下の XML に対応します。

XML

```
<Tools>
  <ToolsItem ToolsKey="845">Hammer</ToolsItem>
  <ToolsItem ToolsKey="1009">Monkey Wrench</ToolsItem>
  <ToolsItem ToolsKey="3762">Screwdriver</ToolsItem>
</Tools>
```

ここでは、配列のキーに対応する ToolsKey という属性が作成されています。この属性の名前は以下のように決定されます。

テーブル 5-3: リスト項目のタグ

プロパティの XMLKEYNAME パラメータ	プロパティの XMLNAME パラメータ	キーがある属性の名前
指定あり	無視	XMLKEYNAME の値
指定なし	指定あり	末尾に Key が連結された XMLNAME の値
	指定なし	末尾に Key が連結されたプロパティ名

XMLITEMNAME プロパティ・パラメータは属性名に影響を与えないことに注目してください。このパラメータの詳細は、前のセクションを参照してください。

例えば、以下のように XMLKEYNAME は設定しないが、XMLNAME を MyXMLName に設定するとします。

```
Property Tools As %ArrayOfDataTypes(XMLNAME = "MyXMLName");
```

次に、同じプロパティが以下の XML フラグメントに対応するとします。

XML

```
<MyXMLName>
  <MyXMLNameItem MyXMLNameKey="845">Hammer</MyXMLNameItem>
  <MyXMLNameItem MyXMLNameKey="1009">Monkey Wrench</MyXMLNameItem>
  <MyXMLNameItem MyXMLNameKey="3762">Screwdriver</MyXMLNameItem>
</MyXMLName>
```

6

要素と属性に対するネームスペースの指定

XML の要素と属性は、異なるネームスペースに属すことができ、XML スキーマの仕様により、ネームスペースの割り当ての制御および表示には複数の方法が提供されています。%XML.Adaptor クラスは、該当するサポートを XML ドキュメントに提供します。

"タイプのネームスペースの指定" も参照してください。

このトピックの XML 例はリテラル形式になります。

このページで説明されているクラスとプロパティのパラメータ
· NAMESPACE · ELEMENTQUALIFIED · ATTRIBUTEQUALIFIED · XMLREF · REFNAMESPACE · XSDTYPE · XMLPREFIX

6.1 概要

ここでは、XML ネームスペースの更新処理について説明します。また、InterSystems IRIS® データ・プラットフォーム・オブジェクトが XML ネームスペースに割り当てられる方法の概要も説明します。

6.1.1 ネームスペース更新処理

このドキュメントでは全般的に、読者が XML に精通していると想定しています。ただし、XML ドキュメントで要素や属性に割り当てられるネームスペースがある場合には、そのネームスペースを決定する方法を見直すことが重要な場合があります。

まず、XML ドキュメントが既定のネームスペースもしくはネームスペースの接頭語を、XML ドキュメントが持つあらゆる要素や属性に対して含まない限り、対応する XML スキーマの確認が必要です。インポートされた要素や属性を除いて、どの要素や属性も次の内のいずれかとなります。

· Qualified (修飾された) はスキーマのターゲット・ネームスペース内にある要素または属性を意味します。

- Unqualified（無修飾）は要素と属性では意味が異なります。無修飾の要素とはネームスペースにない要素を意味します。無修飾の属性とは、要素を含んだ既定のネームスペースがある場合に、そこに存在する属性を意味します。

定義する各要素や属性について、スキーマは、その項目が修飾されたものか、もしくは無修飾のものかを示します。スキーマはこれを以下の組み合わせにより行います。

- `<schema>` 要素は `elementFormDefault` および `attributeFormDefault` 属性を指定できます。これにより、スキーマの任意の要素および属性の既定ネームスペース割り当てを制御します。指定できる値は `"qualified"` および `"unqualified"` になります。

  これらの属性はオプションです。既定値は両項目とも `"unqualified"` です。つまり、既定では要素もしくは属性が接頭語なしで使用される場合、それはネームスペースにはないことになります。

- 次に、要素もしくは属性の定義で `form` 属性の指定ができます。これはその項目をどのようにネームスペースに割り当てるかを示すものです。指定できる値は `"qualified"` および `"unqualified"` になります。

以下の XML ドキュメントについて考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ClassA xmlns="mynamespace" xmlns:s01="mynamespace" s01:String1="abcdef">
  <s01:ClassB xmlns="">
    <String3>qrstuv</String3>
    <String4>wxyz</String4>
  </s01:ClassB>
  <String2>ghijkl</String2>
</ClassA>
```

ここでは、わかりやすくするために、対応するスキーマ・ドキュメントが `elementFormDefault` および `attributeFormDefault` の既定値を使用し、`form` 属性は定義するどの項目に対しても指定しないことを想定しています。さらに、このドキュメントの項目は、以下のようにネームスペースにあります。

- 以下の 2 つの項目により、`<ClassA>` 要素は `mynamespace` にあります。
  - これは、この要素とその直接の子（`xmlns="mynamespace"`）の既定のネームスペース宣言で指定されたネームスペースです。
  - `<ClassA>` 要素には、他のネームスペースを示すネームスペース接頭語がありません。
- `String1` 属性は `mynamespace` にあります。この属性では `s01` 接頭語を使用しているからです。`xmlns:s01` ネームスペース宣言は、`s01` で `mynamespace` ネームスペースを参照することを示しています。

  スキーマは `attributeFormDefault`（`"unqualified"`）を既定で使用しているので、`String1` 属性はネームスペースの接頭語を使用しなくても、`mynamespace` にあることになります。
- `<ClassB>` 要素は `mynamespace` にあります。この属性では `s01` 接頭語を使用しているからです。
- 以下の 2 つの項目により、`<String3>` 要素と `<String4>` 要素は、いずれのネームスペースにもありません。
  - 親要素のネームスペース宣言は、ここでの既定のネームスペースが NULL（`xmlns=""`）であることを示します。
  - これらの要素には、他のネームスペースを示すネームスペース接頭語がありません。
- `<String2>` 要素は `mynamespace` ネームスペースになります。これが、その親要素で指定された既定のネームスペースだからです。

6.1.2 XML のネームスペースとクラス

InterSystems IRIS XML サポートでは、クラスごとにネームスペースを指定します。NAMESPACE クラス・パラメータを使用して、そのクラスおよびその直接の子オブジェクトのインスタンスのネームスペースを指定します。また、ELEMENTQUALIFIED パラメータと ATTRIBUTEQUALIFIED パラメータを使用して、そのオブジェクト値プロパティのプロパティがグローバル (親と同じネームスペースに属している) かローカルかを指定します。

注釈　プロパティ パラメータとして、ELEMENTQUALIFIED を指定することもできます (このドキュメントでは説明していない、特殊なシナリオで必要とされる場合)。

6.1.3 ネームスペースとコンテキスト

特にネームスペースでは、XML 対応オブジェクトがコンテキストに応じて処理が異なることに注意することが重要です。例えば、`Address` オブジェクトを最上位レベルでエクスポートすると、そのオブジェクトはグローバル要素となります。`Address` オブジェクトへの参照を含む `Person` オブジェクトをエクスポートすると、`Address` はローカル要素 (`Person` のその他すべてのプロパティとして) となります。グローバル要素とローカル要素は異なる方法でネームスペースに割り当てられます。

6.2 グローバル要素として扱われるオブジェクトのネームスペースの指定

XML 対応のオブジェクトを最上位レベルでインポートまたはエクスポートすると、そのオブジェクトは、グローバル要素になり、以下のようにネームスペースに割り当てられます。

- クラスの NAMESPACE パラメータを指定した場合は、要素はそのネームスペースに割り当てられます。
- クラスの NAMESPACE パラメータを指定しない場合は、要素はどのネームスペースにも属しません。ただし、エクスポート中にネームスペースを指定することができます。"オブジェクトからの XML 出力の記述" を参照してください。

例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class MyApp.Person Extends (%Persistent, %XML.Adaptor)
{
Parameter NAMESPACE = "http://www.person.org";

Property Name As %Name [ Required ];

Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h") [ Required ];
}
```

このクラスのオブジェクトをエクスポートまたはインポートする場合、プロジェクションは以下のようになります。

XML

```
<Person xmlns="http://www.person.org">
  <Name>Isaacs,Rob G.</Name>
  <DOB>1981-01-29</DOB>
</Person>
```

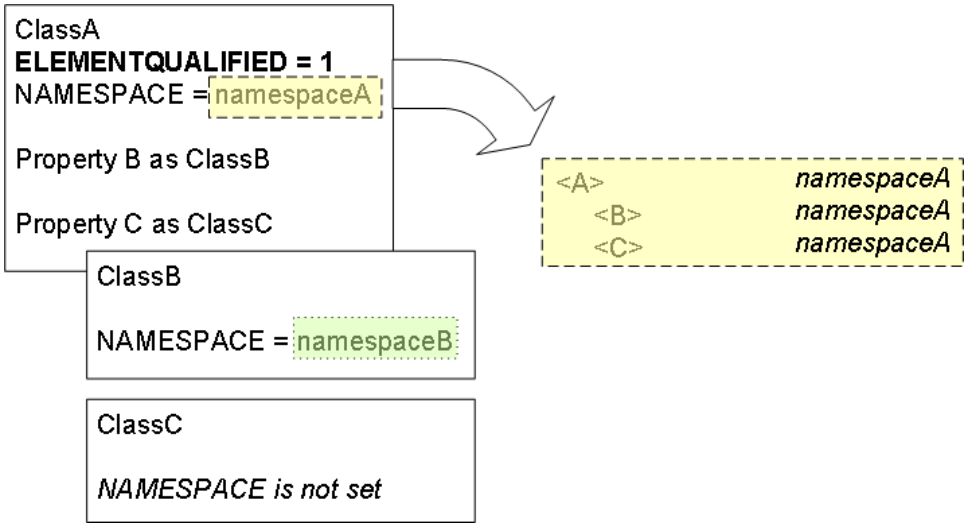
6.3 要素として投影されるプロパティのネームスペースの指定

このセクションでは、要素として投影されるプロパティのネームスペースを指定する方法について説明します。

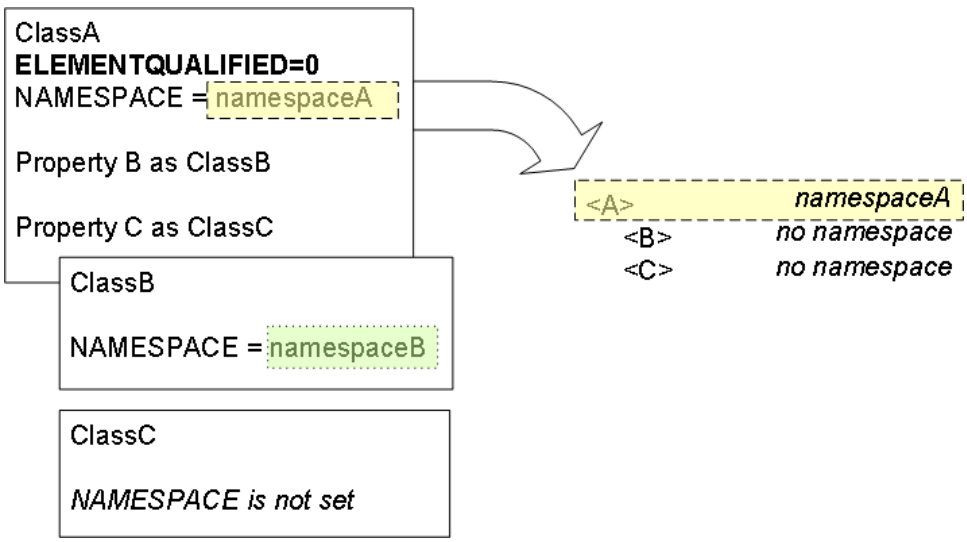
6.3.1 ケース 1 : プロパティがローカル要素として扱われる場合

XML 対応のオブジェクトを最上位レベルでインポートまたはエクスポートすると、要素として投影されるすべてのプロパティは、既定でローカル要素になります。これらのローカル要素には、以下の 2 つのネームスペースを割り当てることができます。

- 親クラスの ELEMENTQUALIFIED クラス・パラメータが 1 の場合、ローカル要素は修飾されており、親要素のネームスペースに明示的に含まれます。



- 親クラスの ELEMENTQUALIFIED クラス・パラメータが 0 の場合、ローカル要素は修飾されず、どのネームスペースにも属しません。(ただし、エクスポート中にネームスペースを指定することができます。"オブジェクトからの XML 出力の記述" を参照してください。



いずれの場合も、子クラスのネームスペースは無視されます。

注釈　ELEMENTQUALIFIED の既定は、入力または出力がリテラル形式かエンコード形式かによって異なります。リテラル形式は既定値で、最も一般的です。

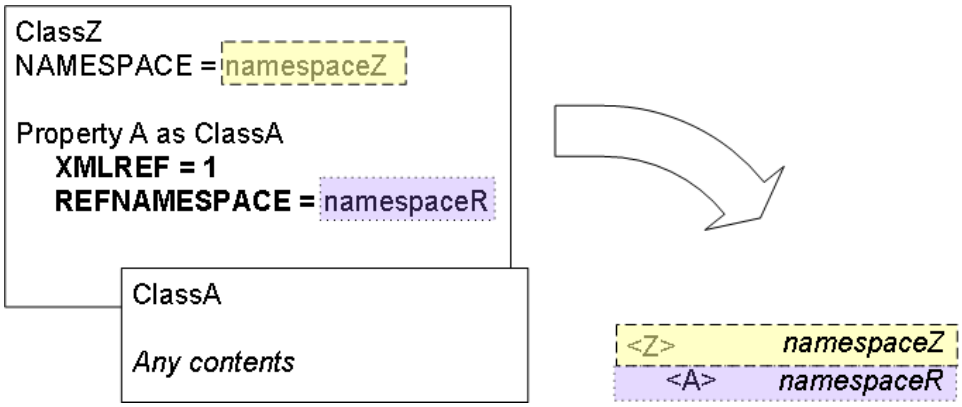
リテラル形式では、ELEMENTQUALIFIED の既定は 1 です。エンコード形式では、ELEMENTQUALIFIED の既定は 0 です。

これらの形式の詳細は、"XML ドキュメントのフォーマット・オプションの指定" を参照してください。

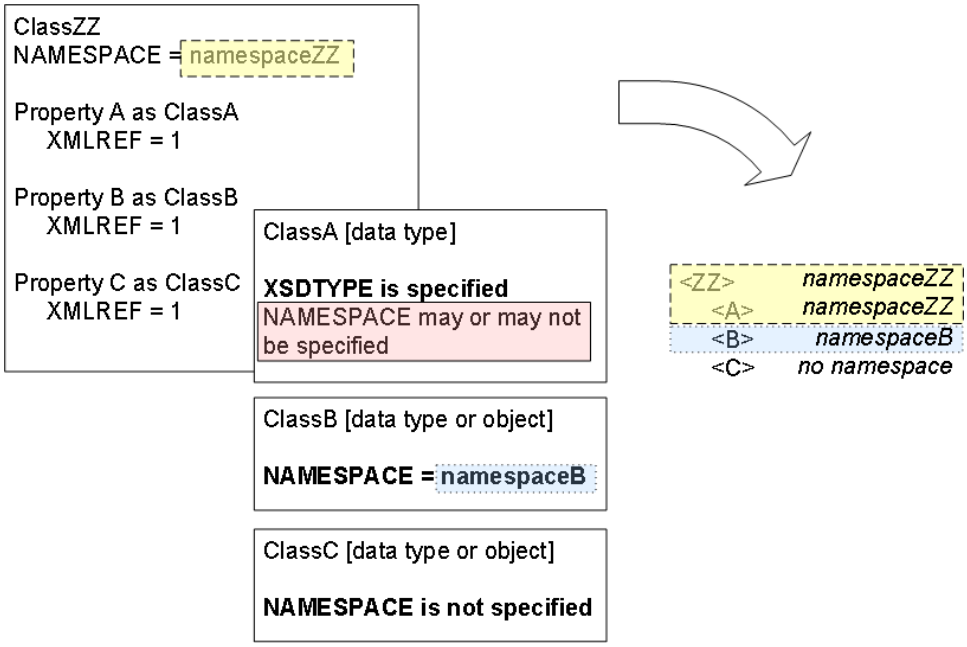
6.3.2 ケース 2：プロパティがグローバル要素として扱われる場合

プロパティをグローバル要素に入れ、それをネームスペースに割り当てることもできます。そのためには、XMLREF プロパティ・パラメータを 1 に設定します。以下では、対応する要素がどのようにネームスペースに割り当てられるのかについて説明します。

1. REFNAMESPACE プロパティ・パラメータを指定した場合、要素はそのネームスペースに入ります。



2. それ以外の場合、要素は、以下のようにネームスペースに割り当てられます。
   a. プロパティが任意のタイプで、かつ XSDTYPE クラス・パラメータを指定する場合、要素は親クラスのネームスペースに入ります。
   b. それ以外で、プロパティ・クラスが NAMESPACE クラス・パラメータを定義する場合は、要素はプロパティ・クラスのネームスペースに入ります。
   c. プロパティ・クラスがこれらのどのクラス・パラメータも定義しない場合は、要素はどのネームスペースにも入りません。



注釈　XMLREF プロパティ・パラメータは XMLELEMENTREF プロパティ・パラメータを置換します。ただし、XMLELEMENTREF パラメータは永久にサポートされます。

6.4 属性として投影されるプロパティのネームスペースの指定

このセクションでは、属性として投影されるプロパティのネームスペースを指定する方法について説明します。ATTRIBUTEQUALIFIED パラメータにより、属性がネームスペース接頭語で修飾されるかどうかを指定します。指定可能な値は以下のとおりです。

- 0 (既定値)：これはネームスペース接頭語がないことを示します。
- 1：これはネームスペース接頭語があることを示します。

XMLREF および REFNAMESPACE プロパティ・パラメータも、属性として投影するプロパティでサポートされています。

属性として投影されたプロパティでは、XMLREF プロパティ・パラメータを 1 に設定すると、対応する属性は以下のようにネームスペースに割り当てられます。

1. REFNAMESPACE プロパティ・パラメータを指定した場合、属性はそのネームスペースに入ります。
2. それ以外の場合
   a. プロパティが任意のタイプで、かつ XSDTYPE クラス・パラメータを指定する場合、属性は親クラスのネームスペースに入ります。
   b. それ以外の場合で、プロパティ・クラスが NAMESPACE クラス・パラメータを定義する場合は、属性はプロパティ・クラスのネームスペースに入ります。
   c. プロパティ・クラスがこれらのどのクラス・パラメータも定義しない場合は、属性はどのネームスペースにも入りません。

6.5 ネームスペースのカスタム接頭語の指定

オブジェクトの XML 出力を生成する場合、システムは必要に応じてネームスペース接頭語を生成します。最初のネームスペースの接頭語は s01、次は s02、のように付与されます。別の接頭語を指定することもできます。そのためには、XML 対応オブジェクト自体のクラス定義で XMLPREFIX パラメータを設定します。このパラメータには以下の 2 つの効果があります。

- 指定した接頭語が XML 出力で宣言されるようになります。つまり、不要な場合でも宣言されます。
- 自動生成の接頭語 (このパラメータを設定しない場合はこれが使用される) でない接頭語が使用されます。

例えば、以下のようなクラス定義があるとします。

Class Definition

```
Class GXML.Person Extends (%Persistent, %XML.Adaptor)
{

Parameter XMLPREFIX = "p";

Parameter NAMESPACE = "http://www.person.com";

Parameter XMLNAME = "Person";

Property Name As %Name;

}
```

このクラスの場合、XML 出力は以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Person xmlns="http://www.person.com" xmlns:p="http://www.person.com">
  <Name>Umansky,Jocelyn O.</Name>
</Person>
```

XML へのエクスポートの詳細は、″XML ツールの使用法″ を参照してください。

6.6 推奨事項

開発、デバッグ、およびトラブルシューティングを簡素化するには、以下の方法をお勧めします。

- ネームスペースを指定する必要がある場合は、すべての XML 対応クラスで NAMESPACE を指定します。そうでない場合、既定値の設定ルールが複雑になります。
- ローカル要素が修飾されるかどうかを制御する必要がある場合は、すべての XML 対応クラスで ELEMENTQUALIFIED パラメータも指定します。

7

XML スキーマへの投影の制御

XML 対応クラスの場合、そのクラスの暗黙的な XML スキーマがあります。これは表示することができます。InterSystems IRIS® データ・プラットフォームにはそのスキーマを変更する手段が用意されています。

このトピックの XML 例はリテラル形式になります。

このページで説明されているクラスとプロパティのパラメータ
• CONTENT • DISPLAYLIST • VALUELIST • ESCAPE • MAXLEN • MINLEN • MINVAL • XMLFractionDigits • XMLTotalDigits • XMLLISTPARAMETER • XSDTYPE • XMLTYPE • SUPPRESSTYPEPREFIX

7.1 XML 対応クラスのスキーマの表示

任意の XML 対応クラスのスキーマを表示するには、以下の 2 つのオプションがあります。

- %XML.Schema と %XML.Writer を使用して完全なスキーマ・ドキュメントを生成できます。詳細は、"クラスからの XML スキーマの生成" を参照してください。
- XML 対応クラスの XMLSchema() クラス・メソッドを使用できます。このメソッドは、このクラスの XML スキーマを現在のデバイスに記述します。このメソッドは XML 宣言を記述しません。また、ネームスペースを無視するので、使用には制限があります。ただし、XML タイプのみが必要な場合は、このメソッドは役に立ちます。

このトピックでは主に XMLSchema() クラス・メソッドを使用します。これは、このメソッドが 1 行のみのコードで使用できるためです。

7.1.1 例

例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class GXML.Person Extends (%Persistent, %Populate, %XML.Adaptor)
{
Property Name As %Name;
Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h");
Property GroupID As %String (XMLPROJECTION="ATTRIBUTE");
Property OtherID As %String(XMLPROJECTION = "NONE");
Property Address As GXML.Address;
Property Doctors As list Of GXML.Doctor;
}
```

GXML.Address クラスは以下のとおりです。

Class Definition

```
Class GXML.Address Extends (%Persistent, %Populate, %XML.Adaptor)
{
Property Street As %String;
Property City As %String;
Property State As %String(MAXLEN = 2, PATTERN = "2u");
Property Zip As %String(MAXLEN = 10, PATTERN = "5n.1(1""-""4n)");
}
```

GXML.Doctor クラスは以下のとおりです。

Class Definition

```
Class GXML.Doctor Extends (%Persistent, %Populate, %XML.Adaptor)
{
Property Name As %Name;
}
```

GXML.Person クラスのスキーマを表示するには、ターミナルに以下のコマンドを入力します。

ObjectScript

```
 do ##class(GXML.Person).XMLSchema()
```

以下のようになります。

```
<s:complexType name="Person">
    <s:sequence>
        <s:element name="Name" type="s:string" minOccurs="0" />
        <s:element name="DOB" type="s:date" minOccurs="0" />
        <s:element name="Address" type="s_Address" minOccurs="0" />
        <s:element name="Doctors" type="ArrayOfDoctorDoctor" minOccurs="0" />
    </s:sequence>
    <s:attribute name="GroupID" type="s:string" />
</s:complexType>
<s:complexType name="s_Address">
    <s:sequence>
        <s:element name="City" type="s:string" minOccurs="0" />
        <s:element name="Zip" type="s:string" minOccurs="0" />
    </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfDoctorDoctor">
    <s:sequence>
        <s:element name="Doctor" type="Doctor"
minOccurs="0" maxOccurs="unbounded" nillable="true" />
    </s:sequence>
</s:complexType>
<s:complexType name="Doctor">
    <s:sequence>
```

```
        <s:element name="Name" type="s:string" minOccurs="0" />
    </s:sequence>
</s:complexType>
```

これから以下の点がわかります。

- `<Person>`、`<Address>`、および `<Doctor>` 型のスキーマは、直接、対応するクラス定義に基づいています。
- スキーマは、投影されるプロパティのみで構成されます。
- スキーマは、各プロパティが要素として投影されるか、属性として投影されるかを識別します。例えば、`GroupID` は属性で、`Name` は要素です。
- プロパティのその他のパラメータは、スキーマに影響を与えることがあります。
- この場合、クラス・プロパティは文字列型で、基本的な XSD タイプ (https://www.w3.org/TR/xmlschema-2/ を参照) の 1 つです。

7.2 リテラル・プロパティの XML スキーマへの投影

このセクションでは、リテラル (非コレクション) プロパティが XML タイプにどのように投影されるか、および XML スキーマに影響を及ぼすオプションについて説明します。項目は以下のとおりです。

- データ型クラスの既定の XSD タイプ
- スキーマに影響するコンパイラ・キーワード
- スキーマに影響するパラメータ

7.2.1 InterSystems IRIS データ型クラスの既定の XSD タイプ

一般的な InterSystems IRIS データ型クラスのいずれかにクラスまたはクラス・プロパティが基づく場合、XML タイプは以下のテーブルに従って自動的に設定されます。以下のテーブルに示すように、`%xsd` パッケージのクラスは XML タイプに直接マッピングします。

テーブル 7-1: %Library および %xsd パッケージの InterSystems IRIS データ型の XML タイプ

%xsd パッケージの InterSystems IRIS クラス	%Library パッケージの InterSystems IRIS クラス	XML へのプロジェクションで使用される XSD タイプ
%xsd.anyURI		anyURI
%xsd.base64Binary	%Binary %Status	base64Binary
%xsd.boolean	%Boolean	boolean
%xsd.byte	%TinyInt	byte
%xsd.date	%Date	date
%xsd.dateTime	%PosixTime %StringTimeStamp %TimeStamp	dateTime

%xsd パッケージの InterSystems IRIS クラス	%Library パッケージの InterSystems IRIS クラス	XML へのプロジェクションで使用される XSD タイプ
%xsd.decimal	%Currency %Decimal %Numeric	decimal
%xsd.double	%Double	double
%xsd.float		float
%xsd.hexBinary		hexBinary
%xsd.int		int
%xsd.integer		integer
%xsd.long	%BigInt %Integer	long
%xsd.negativeInteger		negativeInteger
%xsd.nonNegativeInteger		nonNegativeInteger
%xsd.nonPositiveInteger		nonPositiveInteger
%xsd.positiveInteger		positiveInteger
%xsd.short	%SmallInt	short
%xsd.string	%Name %String %List	string
%xsd.time	%Time	time
%xsd.unsignedByte		unsignedByte
%xsd.unsignedInt		unsignedInt
%xsd.unsignedLong		unsignedLong
%xsd.unsignedShort		unsignedShort

XML データ型の詳細は、https://www.w3.org/TR/xmlschema-2/ を参照してください。

例えば、以下のクラスを考えてみます。

Class Definition

```
Class Schema.DataTypesDemo Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE="mytypes";

Property binaryprop As %xsd.base64Binary;

Property booleanprop As %Boolean;

Property dateprop As %Date;

Property datetimeprop As %TimeStamp;
```

```
Property decimalprop As %Numeric;

Property integerprop As %Integer;

Property stringprop As %String;

Property timeprop As %Time;

}
```

このクラスのスキーマは以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="DataTypesDemo">
    <sequence>
      <element minOccurs="0" name="binaryprop" type="s:base64Binary"/>
      <element minOccurs="0" name="booleanprop" type="s:boolean"/>
      <element minOccurs="0" name="dateprop" type="s:date"/>
      <element minOccurs="0" name="datetimeprop" type="s:dateTime"/>
      <element minOccurs="0" name="decimalprop" type="s:decimal"/>
      <element minOccurs="0" name="integerprop" type="s:long"/>
      <element minOccurs="0" name="stringprop" type="s:string"/>
      <element minOccurs="0" name="timeprop" type="s:time"/>
    </sequence>
  </complexType>
</schema>
```

7.2.2 スキーマに影響するコンパイラ・キーワード

必須キーワードは、`minOccurs="0"` 属性を削除することによって XML スキーマに影響します。例えば、以下のクラスを考えてみます。

Class Definition

```
Class Schema.PropKeywords Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE="mytypes";

Property Property1 As %String;

Property Property2 As %String [ Required ];

}
```

ここで使用されるネームスペースのスキーマを生成する場合、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="test">
  <complexType name="PropKeywords">
    <sequence>
      <element minOccurs="0" name="Property1" type="s:string"/>
      <element name="Property2" type="s:string"/>
    </sequence>
  </complexType>
</schema>
```

`minOccurs` の既定値は 1 です。つまり `Property2` は必須です。

注釈　互換性を維持するために、既定では、%XML.Reader は必須プロパティをチェックしませんが、チェックするよう指定することもできます。"必要な要素および属性のチェック" を参照してください。また、既定では、InterSystems IRIS Web サービスは必須プロパティをチェックしませんが、チェックするよう指定することもできます。"必要な要素および属性のチェック" を参照してください。

他のプロパティ・キーワードは、データ型クラスのスキーマに影響しません。

7.2.3 XML スキーマに影響するパラメータ

InterSystems IRIS データ型クラスは多数のパラメータを使用します (データ型クラスごとにサポートされるパラメータの一覧は、"データ型" を参照してください)。ほとんどの場合、これらをプロパティ・パラメータとして指定することもできます。

XML スキーマに影響するパラメータは以下のとおりです。

CONTENT

プロパティ値をエスケープ処理する方法に影響を与えます。"特殊 XML 文字の処理" を参照してください。

他の指定可能な値と比較すると、"MIXED" 値により、スキーマに変更が発生します。以下のクラスを考えてみます。

Class Definition

```
Class Schema.CONTENT Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE = "mytypes";

Property Property1 As %String;

Property Property2 As %String(CONTENT = "STRING");

Property Property3 As %String(CONTENT = "ESCAPE");

Property Property4 As %String(CONTENT = "MIXED");

}
```

ここで使用されるネームスペースのスキーマを生成する場合、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="CONTENT">
    <sequence>
      <element minOccurs="0" name="Property1" type="s:string"/>
      <element minOccurs="0" name="Property2" type="s:string"/>
      <element minOccurs="0" name="Property3" type="s:string"/>
      <element name="Property4">
        <complexType mixed="true">
          <choice maxOccurs="unbounded" minOccurs="0">
            <any processContents="lax"/>
          </choice>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

これらのプロパティの 3 つには同じタイプ情報があります。XML ではその 3 つが同じように扱われるからです。ただし、InterSystems IRIS ではプロパティの処理方法が異なります。詳細は、"特殊 XML 文字の処理" を参照してください。

オブジェクトを Web メソッドの入力や出力として使用する場合に、SoapBodyUse がそのメソッドの encoded であると、InterSystems IRIS では混在したコンテンツが文字列の内容のように扱われます (既定)。つまり、CONTENT を "MIXED" として指定すると、その値は無視されます。

DISPLAYLIST

VALUELIST も指定されていて、XMLLISTPARAMETER が "DISPLAYLIST" に等しいと、スキーマに影響します。これらの 2 つのパラメータに関する説明を参照してください。

MAXLEN

maxLength 属性を制御します。これは、ファセットまたは制限になります。ファセットにより、XML タイプの許容値を定義します。以下の例で、それらをいくつか示します。以下のクラスを考えてみます。

Class Definition

```
Class Schema.BasicFacets Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE = "mytypes";

Property Property1 As %Integer (MINVAL=10, MAXVAL=1000);

Property Property2 As %String (MINLEN=20, MAXLEN=100);

}
```

ここで使用されるネームスペースのスキーマを生成する場合、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:s="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="mytypes">
  <complexType name="BasicFacets">
    <sequence>
      <element minOccurs="0" name="Property1">
        <simpleType>
          <restriction base="s:long">
            <maxInclusive value="1000"/>
            <minInclusive value="10"/>
          </restriction>
        </simpleType>
      </element>
      <element minOccurs="0" name="Property2">
        <simpleType>
          <restriction base="s:string">
            <maxLength value="100"/>
            <minLength value="20"/>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</schema>
```

SOAP ウィザードや XML スキーマ・ウィザードでスキーマ内に maxLength 制限が検出されると、必要に応じて、生成されたクラスに MAXLEN プロパティ・パラメータが設定されます。

MAXVAL

maxInclusive 属性を制御します。MAXLEN の使用例を参照してください。

MINLEN

minLength 属性を制御します。MAXLEN の使用例を参照してください。

SOAP ウィザードや XML スキーマ・ウィザードでスキーマ内に minLength 制限が検出されると、必要に応じて、生成されたクラスに MINLEN プロパティ・パラメータが設定されます。

MINVAL

minInclusive 属性を制御します。MAXLEN の使用例を参照してください。

VALUELLIST

<enumeration> 制限をタイプに追加します。以下のクラスを考えてみます。

Class Definition

```
Class Schema.VALUELIST Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE = "mytypes";

Property Property1 As %String;

Property Property2 As %String (VALUELIST = ",r,g,b");

}
```

以下にこのクラスのスキーマを示します。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="VALUELIST">
    <sequence>
      <element minOccurs="0" name="Property1" type="s:string"/>
      <element minOccurs="0" name="Property2">
        <simpleType>
          <restriction base="s:string">
            <enumeration value="r"/>
            <enumeration value="g"/>
            <enumeration value="b"/>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</schema>
```

XMLFractionDigits

%Numeric プロパティに適用できます。このパラメータは、以下のフラグメントに示すように、〈fractionDigits〉ファセットに対応します。

```
<element minOccurs="0" name="Property2">
  <simpleType>
    <restriction base="s:decimal">
      <fractionDigits value="2"/>
      <totalDigits value="5"/>
    </restriction>
  </simpleType>
</element>
```

XMLTotalDigits

%Numeric プロパティまたは %Integer プロパティに適用できます。このパラメータは、以下のフラグメントに示すように、〈totalDigits〉ファセットに対応します。

```
<element minOccurs="0" name="Property2">
  <simpleType>
    <restriction base="s:decimal">
      <fractionDigits value="2"/>
      <totalDigits value="5"/>
    </restriction>
  </simpleType>
</element>
```

XMLLISTPARAMETER

VALUELIST パラメータを指定する %String プロパティに適用されます。オブジェクトにある値ではなく、XML に投影するための値のリストがあるパラメータの名前を指定します。ほとんどの場合、標準の DISPLAYLIST パラメータも指定して、XMLLISTPARAMETER を `"DISPLAYLIST"` に設定します。

XMLLISTPARAMETER パラメータにより、`<enumeration>` 制限で使用される `value` 属性を制御します。

これをプロパティ・パラメータとして指定することはできません。

XMLPATTERN

pattern 制限を制御します。以下のクラスを考えてみます。

Class Definition

```
Class Schema.Pattern Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE = "mytypes";

Property Property1 As %String;

Property Property2 As %String(XMLPATTERN = "[A-Z]");

}
```

このクラスのスキーマは以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="Pattern">
    <sequence>
      <element minOccurs="0" name="Property1" type="s:string"/>
      <element minOccurs="0" name="Property2">
        <simpleType>
          <restriction base="s:string">
            <pattern value="[A-Z]"/>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
</schema>
```

単純なタイプで複数のパターンが出現する場合、InterSystems IRIS は、https://www.w3.org/TR/xmlschema-2 に従ってパターンを結合します (セクション 4.3.4.3 の Constraints on XML Representation of pattern を参照してください)。パターンは、XMLPATTERN パラメータで同じパターンの異なる分岐 (縦棒で区切られます) として結合されます。

XSDTYPE

XML を投影するときに使用される XSD タイプを宣言します。このパラメータは、すべての InterSystems IRIS データ型クラスで適切に設定されます。InterSystems IRIS XML ツールでは、スキーマ生成時にこのパラメータが使用されます。このパラメータは、入出力変換に直接影響することはありませんが、それらの変換との一貫性は保持する必要があります。

7.3 ストリーム・クラスの XML タイプへの投影

クラスまたはプロパティが InterSystems IRIS ストリームに基づく場合、以下のテーブルに示すように XML タイプに投影されます。

テーブル 7-2: InterSystems IRIS ストリームに対する XML タイプ

InterSystems IRIS ストリーム・タイプ	XML へのプロジェクションで使用される XSD タイプ
%Library.GlobalCharacterStream、%Library.FileCharacterStream、%Stream.FileCharacter、および %Stream.GlobalCharacter	string

InterSystems IRIS ストリーム・タイプ	XML へのプロジェクションで使用される XSD タイプ
%Library.GlobalBinaryStream、%Library.FileBinaryStream、%Stream.FileBinary、および %Stream.GlobalBinary	base64Binary

例えば、以下のクラスを考えてみます。

Class Definition

```
Class Schema.StreamPropDemo Extends (%Persistent, %XML.Adaptor)
{

Parameter XMLTYPENAMESPACE="mytypes";

Property BinStream As %Library.GlobalBinaryStream;

Property CharStream As %Library.GlobalCharacterStream;

}
```

このクラスのスキーマは以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="StreamPropDemo">
    <sequence>
      <element minOccurs="0" name="BinStream" type="s:base64Binary"/>
      <element minOccurs="0" name="CharStream" type="s:string"/>
    </sequence>
  </complexType>
</schema>
```

7.4 コレクション・プロパティの XML スキーマへの投影

このセクションでは、XML 対応クラスについて、コレクション・プロパティがどのように XML スキーマに投影されるかについて説明します。このセクションでは、以下の項目について説明します。

- コレクション・プロパティの投影
- コレクション・クラス使用のオプション

7.4.1 コレクション・プロパティの XML スキーマへの投影

ほとんどの種類のプロパティでは、オブジェクトを XML ドキュメントとして投影することと完全な XML スキーマを検証目的で定義することの両方のために、完全な XML プロジェクションを指定するのに十分な情報がクラス定義に格納されています。ただし、コレクション・プロパティでは、InterSystems IRIS でサポートしている形式の定義の一部では、完全な XML スキーマに対して十分な情報が用意されていません。スキーマが必要なコンテキスト (Web サービスや Web クライアントなど) で XML プロジェクションを使用する場合、完全な XML スキーマが必要です。ない場合は、スキーマに対する検証が失敗します。スキーマに対して検証を行わない場合は、この考慮事項が適用されません。以下のテーブルは、シナリオのリストです。

テーブル 7-3: コレクション・プロパティとそれらの XML プロジェクション詳細の形式

プロパティ定義の形式	XML の使用可否	XML スキーマの使用可否
`Property PropName As List of classname` または `Property PropName As Array of classname`	あり	あり
`Property PropName As %ListOfDataTypes` または `Property PropName As %ArrayOfDataTypes`	あり	あり (ただし、コレクション項目の既定のタイプである文字列が適切でない場合があります)
`Property PropName As %ListOfObjects` または `Property PropName As %ArrayOfObjects`	あり	なし (スキーマでは、コレクション項目のタイプが指定されません)

以下のサブセクションでは、これらのシナリオに対応する XML スキーマを示します。

7.4.1.1 List of Classname

このセクションでは、`List of Classname` として定義されるプロパティが XML 対応クラスに含まれる場合、そのクラスから生成される XML スキーマの一部を示します。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As list Of %Integer(XMLITEMNAME = "MyXmlItemName");
```

Test.DemoList1 という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="mytypes">
  <complexType name="DemoList1">
    <sequence>
      <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNameLong" xmlns:s01="mytypes"/>

    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNameLong">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true" type="s:long"/>

    </sequence>
  </complexType>
...
</schema>
```

タイプ名には以下のルールが適用されます。

- PropName プロパティの場合、対応するタイプは `ArrayOf`XMLItemNameType という名前になります。
  - "リスト型プロパティの要素名と属性名の制御" で説明されているように、XMLItemName はコレクションの項目名です。データ型プロパティの場合、既定の項目名は末尾に `Item` が追加されたプロパティ名になります(オブジェクト・プロパティの場合、既定の項目名は短いクラス名になります)。
  - Type は、プロパティ・クラスが投影される XML タイプです。

```
<element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNameLong" xmlns:s01="mytypes"/>
```

注釈　XMLItemName が Type と同じ場合、PropName プロパティでは、対応するタイプは `ArrayOf`XMLItemName という名前になります。つまり、重複する配列項目がタイプ名から削除されます。タイプ名に重複する名前を含めるには、(%XML.Schema のインスタンスの) AllowRedundantArrayName プロパティを 1 に指定します。同様に、Web サービス・クラスで、重複する配列項目名を WSDL のタイプに含めるには、(Web サービス・クラスの) ALLOWREDUNDANTARRAYNAME パラメータを 1 に指定します。

- タイプ ArrayOfXMLItemNameType は、XMLItemName という名前の他のタイプの <sequence> として定義されます。

```
  <complexType name="ArrayOfMyXmlItemNameLong">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s:long"/>
    </sequence>
  </complexType>
```

- 要素 XMLItemName は、データ型クラスに対応する XSD タイプに基づいています。

```
<element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true" type="s:long"/>
```

Classname がオブジェクト・クラスを参照する際、同じルールが適用されます。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As list Of SimpleObject(XMLITEMNAME = "MyXmlItemName");
```

Simple.Object には、MyProp と AnotherProp の 2 つのプロパティが含まれています。Test.DemoObjList という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="mytypes">
  <complexType name="DemoObjList">
    <sequence>
      <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNameSimpleObject"
xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNameSimpleObject">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:SimpleObject" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="SimpleObject">
    <sequence>
      <element minOccurs="0" name="MyProp" type="s:string"/>
      <element minOccurs="0" name="AnotherProp" type="s:string"/>
    </sequence>
  </complexType>
...
</schema>
```

7.4.1.2 Array of Classname

このセクションでは、Array of Classname として定義されるプロパティが XML 対応クラスに含まれる場合、そのクラスから生成される XML スキーマの一部を示します。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As array Of %Integer(XMLITEMNAME = "MyXmlItemName", XMLKEYNAME = "MyXmlKeyName");
```

Test.DemoArray1 という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="DemoArray1">
    <sequence>
      <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNamePairOfMyXmlKeyNameLong"
xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNamePairOfMyXmlKeyNameLong">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:PairOfMyXmlKeyNameLong" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="PairOfMyXmlKeyNameLong">
    <simpleContent>
      <extension base="s:long">
        <attribute name="MyXmlKeyName" type="s:string" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
...
</schema>
```

タイプ名には以下のルールが適用されます。

- PropName プロパティの場合、対応するタイプは ArrayOfXMLItemNamePairOfXMLKeyNameType という名前になります。
  - "配列型プロパティの要素名と属性名の制御" で説明されているように、XMLItemName はコレクションの項目名です。データ型プロパティの場合、既定の項目名は末尾に Item が追加されたプロパティ名になります(オブジェクト・プロパティの場合、既定の項目名は短いクラス名になります)。
  - "配列型プロパティの要素名と属性名の制御" で説明されているように、XMLKeyName はコレクションのキー名です。既定は、末尾に Key が連結されたプロパティ名です。
  - Type は、プロパティ・クラスが投影される XML タイプです。

  ```
  <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNamePairOfMyXmlKeyNameLong"
  xmlns:s01="mytypes"/>
  ```

  注釈　XMLKeyName が Type と同じ場合、PropName プロパティでは、対応するタイプは ArrayOfXMLItemNamePairOfXMLKeyName という名前になります。つまり、重複する配列項目がタイプ名から削除されます。タイプ名に重複する名前を含めるには、(%XML.Schema のインスタンスの) AllowRedundantArrayName プロパティを 1 に指定します。同様に、Web サービス・クラスで、重複する配列項目名を WSDL のタイプに含めるには、(Web サービス・クラスの) ALLOWREDUNDANTARRAYNAME パラメータを 1 に指定します。

- タイプ ArrayOfXMLItemNamePairOfXMLKeyNameType は、PairOfXMLKeyNameType という名前の他のタイプの <sequence> として定義されます。

  ```
   <complexType name="ArrayOfMyXmlItemNamePairOfMyXmlKeyNameLong">
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
  type="s01:PairOfMyXmlKeyNameLong" xmlns:s01="mytypes"/>
      </sequence>
    </complexType>
  ```

- タイプ PairOfXMLKeyNameType は、指定された XSD タイプの拡張です。この拡張により、XMLKeyName という名前の属性が追加されます。

```
<complexType name="PairOfMyXmlKeyNameLong">
  <simpleContent>
    <extension base="s:long">
      <attribute name="MyXmlKeyName" type="s:string" use="required"/>
    </extension>
  </simpleContent>
</complexType>
```

Classname がオブジェクト・クラスを参照する際、同じルールが適用されます。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As %ArrayOfObjects(XMLITEMNAME = "MyXmlItemName", XMLKEYNAME = "MyXmlKeyName");
```

Simple.Object には、MyProp と AnotherProp の 2 つのプロパティが含まれています。Test.DemoObjArray という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="mytypes">
  <complexType name="DemoObjArray">
    <sequence>
     <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNamePairOfMyXmlKeyNameSimpleObject"
 xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNamePairOfMyXmlKeyNameSimpleObject">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:PairOfMyXmlKeyNameSimpleObject" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="PairOfMyXmlKeyNameSimpleObject">
    <complexContent>
      <extension base="s01:SimpleObject" xmlns:s01="mytypes">
        <attribute name="MyXmlKeyName" type="s:string" use="required"/>
      </extension>
    </complexContent>
  </complexType>
  <complexType name="SimpleObject">
    <sequence>
      <element minOccurs="0" name="MyProp" type="s:string"/>
      <element minOccurs="0" name="AnotherProp" type="s:string"/>
    </sequence>
  </complexType>
</schema>
```

7.4.1.3 %ListOfDataTypes

このセクションでは、%ListOfDataTypes として定義されるプロパティが XML 対応クラスに含まれる場合、そのクラスから生成される XML スキーマの一部を示します。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As %ListOfDataTypes(XMLITEMNAME = "MyXmlItemName");
```

Test.DemoList という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="DemoList">
    <sequence>
     <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNameString" xmlns:s01="mytypes"/>

    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNameString">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true" type="s:string"/>

    </sequence>
  </complexType>
</schema>
```

タイプ名のルールの詳細は、"List of Classname" を参照してください。コレクション項目 (この例では PropNameItem) は XSD 文字列型に基づいています。

```
<element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true" type="s:string"/>
```

つまり、コレクション項目は文字列であることが想定されています。"コレクション・クラス使用のオプション" も参照してください。

7.4.1.4 %ArrayOfDataTypes

このセクションでは、%ArrayOfDataTypes として定義されるプロパティが XML 対応クラスに含まれる場合、そのクラスから生成される XML スキーマの一部を示します。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As %ArrayOfDataTypes(XMLITEMNAME = "MyXmlItemName", XMLKEYNAME = "MyXmlKeyName");
```

Test.DemoArray という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="mytypes">
  <complexType name="DemoArray">
    <sequence>
      <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNamePairOfMyXmlKeyNameString"
xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNamePairOfMyXmlKeyNameString">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:PairOfMyXmlKeyNameString" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="PairOfMyXmlKeyNameString">
    <simpleContent>
      <extension base="s:string">
        <attribute name="MyXmlKeyName" type="s:string" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
...
</schema>
```

タイプ名のルールの詳細は、"Array of Classname" を参照してください。コレクション項目 (この例では PairOfMyXmlKeyNameString) は XSD 文字列型に基づいています。

```
<complexType name="PairOfMyXmlKeyNameString">
  <simpleContent>
    <extension base="s:string">
      <attribute name="MyXmlKeyName" type="s:string" use="required"/>
    </extension>
  </simpleContent>
</complexType>
```

つまり、コレクション項目は文字列であることが想定されています。"コレクション・クラス使用のオプション" も参照してください。

7.4.1.5 %ListOfObjects

このセクションでは、%ListOfObjects として定義されるプロパティが XML 対応クラスに含まれる場合、そのクラスから生成される XML スキーマの一部を示します。例えば、以下のようなプロパティ定義を考えてみます。

Class Member

```
Property PropName As list Of %Integer(XMLITEMNAME = "MyXmlItemName");
```

Test.DemoObjList1 という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="mytypes">
  <complexType name="DemoObjList1">
    <sequence>
      <element minOccurs="0" name="PropName" type="s01:ArrayOfMyXmlItemNameRegisteredObject"
xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNameRegisteredObject">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:RegisteredObject" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
...
</schema>
```

タイプ名のルールの詳細は、"List of Classname" を参照してください。コレクション項目のタイプは未定義の RegisteredObject です。

```
<element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:RegisteredObject" xmlns:s01="mytypes"/>
```

したがって、このスキーマは使用できません。"コレクション・クラス使用のオプション" を参照してください。

7.4.1.6 %ArrayOfObjects

このセクションでは、%ArrayOfObjects として定義されるプロパティが XML 対応クラスに含まれる場合、そのクラスから生成される XML スキーマの一部を示します。例えば、以下のようなプロパティ定義を考えてみます。

```
Property PropName As %ArrayOfObjects(XMLITEMNAME = "MyXmlItemName", XMLKEYNAME = "MyXmlKeyName");
```

Test.DemoObjArray1 という名前の XML 対応クラスにこのプロパティが存在する場合、このクラスの XML スキーマには以下が含まれます。

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:s="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" targetNamespace="mytypes">
  <complexType name="DemoObjArray1">
    <sequence>
      <element minOccurs="0" name="PropName"
type="s01:ArrayOfMyXmlItemNamePairOfMyXmlKeyNameRegisteredObject" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="ArrayOfMyXmlItemNamePairOfMyXmlKeyNameRegisteredObject">
    <sequence>
      <element maxOccurs="unbounded" minOccurs="0" name="MyXmlItemName" nillable="true"
type="s01:PairOfMyXmlKeyNameRegisteredObject" xmlns:s01="mytypes"/>
    </sequence>
  </complexType>
  <complexType name="PairOfMyXmlKeyNameRegisteredObject">
    <complexContent>
      <extension base="s01:RegisteredObject" xmlns:s01="mytypes">
        <attribute name="MyXmlKeyName" type="s:string" use="required"/>
      </extension>
    </complexContent>
  </complexType>
...
</schema>
```

タイプ名のルールの詳細は、"List of Classname" を参照してください。コレクション項目のタイプは未定義の `RegisteredObject` に基づいています。

```
  <complexType name="PairOfMyXmlKeyNameRegisteredObject">
    <complexContent>
      <extension base="s01:RegisteredObject" xmlns:s01="mytypes">
        <attribute name="MyXmlKeyName" type="s:string" use="required"/>
      </extension>
    </complexContent>
  </complexType>
```

したがって、このスキーマは使用できません。"コレクション・クラス使用のオプション" を参照してください

7.4.2 コレクション・クラス使用のオプション

XML 対応クラス内のタイプ %ListOfDataTypes または %ArrayOfDataTypes の各プロパティについて、コレクション項目は文字列であることが想定されています。この想定は、お客様のニーズに合っている場合もあれば、そうでない場合もあります。同様に、タイプ %ListOfObjects または %ArrayOfObjects の各プロパティについて、コレクション項目のタイプは `RegisteredObject` であり、InterSystems IRIS にはタイプ `RegisteredObject` の XML プロジェクションが含まれていないため、XML スキーマは使用できません

これらのシナリオでは、以下のいずれかを実行できます。

- 形式 `List of Classname` または `Array of Classname` が含まれるようにプロパティ定義を変更します (Classname は適切なクラスです)。Classname がオブジェクト・クラスである場合、クラスを XML 対応にします。
- コレクション・クラスのカスタムのサブクラス (%ListOfDataTypes、%ArrayOfDataTypes、%ListOfObjects、%ArrayOfObjects) を作成します。サブクラスで、ELEMENTTYPE クラス・パラメータを指定します。以下はその例です。

  Class Definition

  ```
  Class MyApp.MyIntegerCollection Extends %ListOfDataTypes
  {
  Parameter ELEMENTTYPE="%Library.Integer";

  }
  ```

ELEMENTTYPE には、コレクションで使用されるクラスの詳細なパッケージ名およびクラス名を指定します。%ListOfDataTypes または %ArrayOfDataTypes をサブクラスに指定する場合、データ型クラスを指定します。コレクション要素のタイプは、そのクラスの XSDTYPE パラメータにより制御されます。

%ListOfObjects または %ArrayOfObjects をサブクラスに指定する場合、XML 対応クラスを指定します。以下はその例です。

Class Definition

```
Class MyApp.MyObjectCollection Extends %ListOfObjects
{
Parameter ELEMENTTYPE="MyApp.SimpleObject";
}
```

次に、プロパティ定義でカスタムのコレクション・クラスを使用します。以下はその例です。

```
Property MyProp as MyApp.MyIntegerCollection;
```

7.5 その他の XML 対応クラスの XML タイプへの投影

XML 対応クラス、または XML 対応クラスに基づくプロパティの場合、XML タイプは以下のように決定されます。クラスに XMLTYPE パラメータの値がある場合、それがタイプ名として使用されます。それ以外の場合、短いクラス名が XML タイプ名として使用されます。

例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class GXML.PersonWithAddress Extends (%Persistent, %XML.Adaptor)
{
Parameter XMLTYPE = "PersonType";
Property Name As %Name;
Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h");
Property HomeAddress As GXML.Address;
}
```

このクラスのインスタンスの場合、XML タイプは `PersonType` になります。これは、XMLTYPE パラメータから取得されます。

`GXML.Address` クラスに XMLTYPE パラメータが含まれていないとします。この場合、`<HomeAddress>` 要素では、XML タイプは `Address` です。これは短いクラス名です。

7.6 タイプのネームスペースの指定

XML タイプは以下のようにネームスペースに割り当てられます。

1. 対応するクラス定義が XSDTYPE クラス・パラメータを定義する場合、タイプは以下の W3 ネームスペースに入ります。

   ```
   http://www.w3.org/2001/XMLSchema
   ```

データ型クラス内でのみ XSDTYPE クラス・パラメータを指定します。

注釈　データ型クラスは XSDTYPE クラス・パラメータを継承しません。つまり、既存のデータ型クラスをサブクラス化すると、クラスを XSD タイプのいずれかにマッピングする必要がある場合にはこのパラメータを指定する必要があります。

2. クラス定義が XSDTYPE を定義せずに NAMESPACE を定義する場合、タイプは NAMESPACE によって指定されるネームスペースに入ります。
3. それ以外の場合は、タイプはどのネームスペースにも入りません。

   ただし、ネームスペースはスキーマの生成時に指定できます。"クラスからの XML スキーマの生成" を参照してください。

タイプが割り当てられるネームスペースを参照するには、%XML.Schema および %XML.Writer を使用する必要があります。詳細は、"クラスからの XML スキーマの生成" を参照してください。

7.7 QName タイプのネームスペース接頭語の抑制

"XML ツールの使用法" の説明に従って、%XML.Writer で出力を生成する際に、XML タイプ属性を含めることができます。そのためには、ライターの OutputTypeAttribute プロパティを 1 に指定します。

既定では、タイプ属性は QName (修飾名) として書き込まれ、タイプの名前とタイプの所属先ネームスペースの両方を示します。以下はその例です。

```
<TeamA xmlns:s01="http://mynamespace" xsi:type="s01:TeamA">
```

対応する InterSystems IRIS クラス定義を定義して、ネームスペース接頭語が抑制されるようにします。以下はその例です。

```
<TeamB xsi:type="TeamB">
```

例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class STP.TeamA Extends (%RegisteredObject, %XML.Adaptor)
{
Parameter NAMESPACE = "http://mynamespace";
Property Member1 as %String;
Property Member2 as %String;
}
```

STP.TeamB クラスは、記載されていませんが同じ定義があり、SUPPRESSTYPEPREFIX も 1 に指定しています。

どちらのクラスも、3 番目のクラスでプロパティとして使用されます。

Class Definition

```
Class STP.Container Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter NAMESPACE = "http://mynamespace";

Property TeamA As STP.TeamA;

Property TeamB As STP.TeamB;

}
```

STP.Container のインスタンスの出力を生成すると (さらに XML タイプ属性の出力を有効にすると)、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Container xmlns="http://mynamespace"
           xmlns:s="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <TeamA xmlns:s01="http://mynamespace" xsi:type="s01:TeamA">
    <Member1 xsi:type="s:string">Jack O'Neill</Member1>
    <Member2 xsi:type="s:string">Samantha Carter</Member2>
  </TeamA>
  <TeamB xsi:type="TeamB">
    <Member1 xsi:type="s:string">Jasper O'Nelson</Member1>
    <Member2 xsi:type="s:string">Sandra Chartres</Member2>
  </TeamB>
</Container>
```

<TeamA> 要素には xsi:type 属性があり、それは "s01:TeamA" になっています。この要素のネームスペース宣言では、s01 接頭語がネームスペース http://mynamespace を参照することを示します。

ただし、<TeamB> 要素には xsi:type 属性内に接頭語がありません。

注釈　SUPPRESSTYPEPREFIX は、XML タイプが属するネームスペースに影響しません。タイプ接頭語の記述を抑制するだけです。

8

XML スキーマの詳細オプション

このトピックでは、XML スキーマを作成するための詳細オプションについて説明します。

ここに示す XML 例はリテラル形式になります。

このページで説明されているクラスとプロパティのパラメータ
・ XMLTYPECONSTRAINT ・ XMLINCLUDEINLIST ・ XMLINHERITANCE

8.1 サブクラスのタイプの自動作成

クラスの XML プロジェクションを定義すると、そのサブクラスはすべて自動的にそれぞれ別のタイプにマップされます。それらすべてのサブクラスは、基本タイプとしてスーパークラスを使用します。これは、スーパータイプが使用されるどの場所でも、サブタイプのいずれかを代わりに使用できることを意味します。また、サブタイプを使用して、XML スキーマで選択リストまたは置換グループを定義できます。

抽象クラスの XML プロジェクションを定義することもできます。抽象クラスなのでインスタンス化することはできませんが、派生クラスのスキーマでこのクラスは基本タイプとして表示されます。

例を考えてみます。単純な `Person` クラスから始めます。

Class Definition

```
Class UsingSubclasses.Person Extends (%Persistent, %XML.Adaptor)
{
Property Name As %String [Required];
Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h") [Required];
}
```

`Person` クラスに直接基づく 2 つのクラスがあるとします。1 番目は `Patient` クラスです。

Class Definition

```
Class UsingSubclasses.Patient Extends UsingSubclasses.Person
{
Property PatientID As %String [Required];
}
```

次は `Employee` クラスです。

Class Definition

```
Class UsingSubclasses.Employee Extends UsingSubclasses.Person
{
Property EmployeeID As %String [Required];
}
```

最後に、Person をプロパティとして使用するクラスを考えてみます。

Class Definition

```
Class UsingSubclasses.Example1 Extends (%Persistent, %XML.Adaptor)
{
Property Person As UsingSubclasses.Person;
}
```

Example1 クラスのスキーマを生成すると、結果は以下のようになります。

```
<s:complexType name="Example1">
    <s:sequence>
        <s:element name="Person" type="Person" minOccurs="0" />
    </s:sequence>
</s:complexType>
<s:complexType name="Employee">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="EmployeeID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="Person">
    <s:sequence>
        <s:element name="Name" type="s:string" />
        <s:element name="DOB" type="s:date" />
    </s:sequence>
</s:complexType>
<s:complexType name="Patient">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="PatientID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

これから以下の点がわかります。

- Example1 のタイプは Person です。
- Employee タイプおよび Patient タイプがどちらも定義されています。これは既定です（参考までに、これは XMLTYPECONSTRAINT プロパティ・パラメータを "EXPLICIT" に設定することに対応します）。
- XML スキーマの仕様により、前述のスキーマは <Person> 要素が含まれる任意の場所であり、<Employee> 要素または <Patient> 要素のいずれかを含めることができることになります。

8.2 サブタイプの選択リストの作成

XML スキーマの仕様により、複雑なタイプは、特に関連するタイプでは、タイプの選択リストで構成できます。<Person> 要素 1 つではなく、<Person>、<Patient>、または <Employee> の要素をスキーマで許可するとします。こうしたスキーマを定義するには、以下のように Person プロパティの XMLTYPECONSTRAINT プロパティ・パラメータを "CHOICE" に設定します。

Class Definition

```
Class UsingSubclasses.Example2 Extends (%Persistent, %XML.Adaptor)
{
Property Person As UsingSubclasses.Person(XMLTYPECONSTRAINT = "CHOICE");
}
```

既定では、選択リストは Person クラスのすべてのサブクラスで構成されます。Example2 のスキーマは以下のようになります。

```
<s:complexType name="Example2">
    <s:sequence>
        <s:choice minOccurs="0">
            <s:element name="Employee" type="Employee" />
            <s:element name="Patient" type="Patient" />
            <s:element name="Person" type="Person" />
        </s:choice>
    </s:sequence>
</s:complexType>
<s:complexType name="Employee">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="EmployeeID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="Person">
    <s:sequence>
        <s:element name="Name" type="s:string" />
        <s:element name="DOB" type="s:date" />
    </s:sequence>
</s:complexType>
<s:complexType name="Patient">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="PatientID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

前述の例とは対照的に、Example2 のタイプは、Person、Patient、または Employee で構成される選択リストです。後の 3 つのタイプは、前述の例と同様に定義されます。

8.2.1 選択リストでのサブクラスの制限

すべてのサブクラスを選択リストに含める必要がない場合があります。2 つの方法で、サブクラスのリストを制限できます。

- XMLINCLUDEINGROUP クラス・パラメータを使用して、選択リストに含めないように、任意のサブクラスにマークを付けます。
- XMLCHOICELIST プロパティ・パラメータを、選択リストのサブクラスのコンマ区切りのリストに設定します。
- このパラメータは優先されます。つまり、あるサブクラスが XMLCHOICELIST に挙げられている場合は、そのサブクラス自体が XMLINCLUDEINGROUP = 0 とマークされている場合でも、選択リストに含められます。

以下のセクションに例を示します。

8.2.2 明示的リストを含む選択リストの例

XMLCHOICELIST プロパティ・パラメータを、選択リストに含めるサブクラスのコンマ区切りリストに設定するとします。例えば、次のようになります。

Class Definition

```
Class UsingSubclasses.Example2A Extends (%Persistent, %XML.Adaptor)
{
Property Person As UsingSubclasses.Person
(XMLCHOICELIST = "UsingSubclasses.Patient, UsingSubclasses.Employee",
XMLTYPECONSTRAINT = "CHOICE");
}
```

このクラスのスキーマは以下のようになります。

```
<s:complexType name="Example2A">
    <s:sequence>
        <s:choice minOccurs="0">
            <s:element name="Patient" type="Patient" />
            <s:element name="Employee" type="Employee" />
        </s:choice>
    </s:sequence>
</s:complexType>
<s:complexType name="Patient">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="PatientID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="Person">
    <s:sequence>
        <s:element name="Name" type="s:string" />
        <s:element name="DOB" type="s:date" />
    </s:sequence>
</s:complexType>
<s:complexType name="Employee">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="EmployeeID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

4 つのクラスすべて (Example2A、Patient、Person、および Employee) が表示されていますが、選択リストには Person クラスは含まれていないことに注意してください。また、Person、Patient、および Employee タイプは既定の場合と同様に定義されていることにも注意してください。

8.2.3 XMLINCLUDEINGROUP=0 を含む選択リストの例

クラス Person の別のサブクラスを追加し、XMLINCLUDEINGROUP を 0 に設定してそのサブクラスを制限するとします。

Class Definition

```
Class UsingSubclasses.Other Extends UsingSubclasses.Person
{
Parameter XMLINCLUDEINGROUP = 0;
}
```

この場合、このクラスは選択リストにもスキーマにも含まれません。

8.3 サブタイプの置換グループの作成

XML スキーマの仕様では、置換グループの定義も許可されています。置換グループは、選択肢を作成するための代替の方法となります。構文は多少異なります。タイプの明示的な集中リストを作成するのではなく、以下のように、置換対象として考えられるものに注釈を付けます。

```
<s:complexType name="Example3">
    <s:sequence>
        <s:element ref="Person" minOccurs="0" />
    </s:sequence>
</s:complexType>
<s:element name="Person" type="Person"/>
<s:element name="Employee" type="Employee" substitutionGroup="Person"/>
<s:complexType name="Employee">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="EmployeeID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="Person">
    <s:sequence>
        <s:element name="Name" type="s:string" />
        <s:element name="DOB" type="s:date" />
    </s:sequence>
</s:complexType>
<s:element name="Patient" type="Patient" substitutionGroup="Person"/>
<s:complexType name="Patient">
    <s:complexContent>
        <s:extension base="Person">
            <s:sequence>
                <s:element name="PatientID" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
```

このスキーマを作成するには、以下のクラスを使用します。

Class Definition

```
Class UsingSubclasses.Example3 Extends (%Persistent, %XML.Adaptor)
{
Property Person As UsingSubclasses.Person
(XMLTYPECONSTRAINT = "SUBSTITUTIONGROUP");
}
```

8.3.1 置換グループでのサブクラスの制限

任意のプロパティで、XMLTYPECONSTRAINT プロパティ・パラメータを <SUBSTITUTIONGROUP> に設定すると、前述の例のように、グループはそのプロパティのタイプのすべてのサブクラスで自動的に構成されます。XMLINCLUDEINGROUP パラメータを使用して、置換グループに含めないように、任意のサブクラスにマークを付けます。例えば、Person クラスの別のサブクラスを追加するとします。

Class Definition

```
Class UsingSubclasses.Other Extends UsingSubclasses.Person
{

Parameter XMLINCLUDEINGROUP = 0;

Property OtherID As %String [ Required ];
}
```

この場合、このクラスは置換グループに含まれません。そして、このクラスはこの方法で明示的にマークを付けるので、スキーマには含まれません。

8.4 スーパークラスをタイプとして表示する方法

XML スキーマで特定のタイプの階層を表示する必要がある場合、InterSystems IRIS® データ・プラットフォーム・クラス階層がプロジェクションでどう解釈されるかを理解する必要があります。

クラス階層はとりわけ、意味をなすデータ構成を表します。この階層は、対応する XML タイプ定義に可能な限り反映されます。

例えば、以下のクラスがあるとします。

- `Base` という名前のクラス。3 つのパブリック・プロパティ (`Property1`、`Property2`、および `Property3`) を定義します。
- `Addition1` という名前のクラス。`Base` を拡張し、追加のパブリック・プロパティ (`Addition1`) を定義します。
- `Addition2` という名前のクラス。`Addition1` を拡張し、追加のパブリック・プロパティ (`Addition2`) を定義します。

`Addition2` のスキーマには、何が含まれるでしょうか。5 つのプロパティをすべて表示する必要があります。また、これらのクラスはすべてユーザに定義されたものなので、`Addition2` のスキーマは、クラス階層の詳細を示す必要があります。一方、`Base` が InterSystems IRIS クラス・ライブラリのクラスを拡張し、そのライブラリの別のクラスを拡張する場合、それらの詳細はさほど重要ではありません。

同様に、`Addition2` の XML スキーマは、既定では以下のようになります。

```
<s:complexType name="Addition2">
    <s:complexContent>
        <s:extension base="Addition1">
            <s:sequence>
                <s:element name="Addition2" type="s:decimal" minOccurs="0" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="Addition1">
    <s:complexContent>
        <s:extension base="Base">
            <s:sequence>
                <s:element name="Addition1" type="s:string" minOccurs="0" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="Base">
    <s:sequence>
        <s:element name="Property1" type="s:string" minOccurs="0" />
        <s:element name="Property2" type="s:decimal" minOccurs="0" />
        <s:element name="Property3" type="s:date" minOccurs="0" />
    </s:sequence>
</s:complexType>
```

XML タイプ定義は複数の継承をサポートしないため、InterSystems IRIS XML サポートでは、一定の簡素化された想定が行われます。複数のスーパークラスを拡張するクラスの場合、そのクラスのタイプは最初にリストに挙げられたスーパークラスであると想定されます。例は以下のようになります。以下の 3 つのクラス定義を考えてみます。`AddressPart1` にはプロパティが 1 つ含まれます。

Class Definition

```
Class GXML.Writer.ShowMultiple.AddressPart1 Extends %XML.Adaptor
{
Property Street As %String [ Required ];
}
```

AddressPart2 クラスには別のプロパティが含まれます。

Class Definition

```
Class GXML.Writer.ShowMultiple.AddressPart2 Extends %XML.Adaptor
{
Property City As %String [ Required ];
}
```

最後に、Address はこれらのクラスの両方から (AddressPart1 を最初のスーパークラスとして) 継承され、さらに多くのプロパティを追加します。

Class Definition

```
Class GXML.Writer.ShowMultiple.Address Extends
(GXML.Writers.ShowMultiple.AddressPart1,
GXML.Writers.ShowMultiple.AddressPart2)
{
Property State As %String(MAXLEN = 2, PATTERN = "2u") [ Required ];
Property Zip As %String(MAXLEN = 10, PATTERN = "5n.1(1""-""4n)") [ Required ];
}
```

Address の XML スキーマは以下のようになります。

```
<s:complexType name="Address">
    <s:complexContent>
        <s:extension base="AddressPart1">
            <s:sequence>
                <s:element name="City" type="s:string" />
                <s:element name="State" type="s:string" />
                <s:element name="Zip" type="s:string" />
            </s:sequence>
        </s:extension>
    </s:complexContent>
</s:complexType>
<s:complexType name="AddressPart1">
    <s:sequence>
        <s:element name="Street" type="s:string" />
    </s:sequence>
</s:complexType>
```

これから以下の点がわかります。

- 最初にリストに挙げられたスーパークラス、AddressPart1 は、対応する XML タイプによって表されます。この XML タイプには、予期される詳細がすべて含まれます。
- AddressPart1 タイプに含まれるプロパティ以外にも、残りのプロパティすべてが Address タイプに割り当てられます。これは、AddressPart1 クラスのマッピング後、これらのプロパティの表現として唯一考えられるものです。
- AddressPart1 および AddressPart2 はどちらも %XML.Adaptor のサブクラスですが、%XML.Adaptor クラスの構成は公開されません。重視されるのはカスタム・クラスです。

以下の規則は、任意のクラスのスキーマを表示する際のスーパークラスの処理方法を制御します。

- スーパークラスが %XML.Adaptor から継承される場合、このクラスの投影されたプロパティをすべて表す XML タイプによって表されます。短いクラス名がそのプロパティの XML タイプ名として使用されます。クラスで XMLTYPE パラメータの値が指定されている場合、代わりにその値がタイプ名として使用されます。
- スーパークラスが %XML.Adaptor から継承されない場合、XML タイプには表されません。プロパティがある場合、そのプロパティは継承するクラス (表示しているスキーマのクラス) に割り当てられます。
- 任意のクラスが複数のスーパークラスから継承される場合、最初のスーパークラスに対して XML タイプが作成されます (該当する場合。前述の規則を参照)。最初のスーパークラスに属さないプロパティはすべて、前述の例のように、継承するクラスに割り当てられます。

8.5 複数の XML 対応スーパークラスに基づいたクラス

場合によっては、1 つの特定のクラスが複数の XML 対応スーパークラスに基づいていることがあります。そのような場合、対応する XML スキーマは、これらのクラスがリストされる順序であると見なします。例えば、以下に示すクラスについて考えてみます。このクラスは 2 つの XML 対応スーパークラスを継承しています。

```
Class Test.Default Extends (Test.Superclass1, Test.Superclass2)
{
///additional class members ...
}
```

このクラスの XML スキーマは、左端のクラス `Test.Superclass1` から派生した XML タイプを先にリストし、`Test.Superclass2` から派生した XML リストは後にリストします。このクラスに含まれるオブジェクトの XML 出力を生成するときも、この順序と同じになります。

XML スキーマ（および出力）が右から左の順序で決定されるようにするには、XMLINHERITANCE パラメータを `"right"` に設定します。例えば、以下のようになります。

```
Class Test.Default Extends (Test.Superclass1, Test.Superclass2)
{
Parameter XMLINHERITANCE = "right";

///additional class members ...
}
```

9

特殊なトピック

このトピックでは、追加の特殊なトピックについて説明します。

このトピックの XML 例はリテラル形式になります。

このページで説明されているクラスとプロパティのパラメータ
・ XMLNAME ・ XMLSEQUENCE ・ XMLUNSWIZZLE ・ XMLPREFIX ・ XMLIGNOREINVALIDTAG ・ XMLIGNOREINVALIDATTRIBUTE

9.1 要素を閉じる形式の制御

XML で属性のみを含む要素は、以下のいずれかで示すことができます。

```
<tag attribute="value" attribute="value" attribute="value"></tag>
<tag attribute="value" attribute="value" attribute="value"/>
```

InterSystems IRIS® データ・プラットフォームは、これらの形式を等価と認識します。**%XML.Writer** のあるオブジェクトをエクスポートすると、閉じる形式は制御できますが、XML プロジェクション自体は変更されません。"要素を閉じる形式の制御" を参照してください。

9.2 複数の同名のタグを含む XML ドキュメントの処理

XML の特定の要素には、複数の同名の要素を含めることができます。これらの要素は、その順序によって互いに区別されます。例えば、以下は正規の XML ドキュメントです。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <Person>
      <Name>Able, Andrew</Name>
      <DOB>1977-10-06</DOB>
      <Address>
         <Street>6218 Clinton Drive</Street>
         <City>Reston</City>
         <State>TN</State>
         <Zip>87639</Zip>
      </Address>
      <Address>
         <Street>110 High Street</Street>
         <City>Zanesville</City>
         <State>OR</State>
         <Zip>80719</Zip>
      </Address>
   </Person>
</Root>
```

各クラス・プロパティには一意の名前が必要なので、こうしたドキュメントを InterSystems IRIS クラスにマッピングするには多少の注意を要します。

こうしたドキュメントを InterSystems IRIS クラスにマッピングするには、以下の手順を実行します。

- 必要に応じて XMLNAME プロパティ・パラメータを設定して、異なるクラス・プロパティを同じ XML 名にマッピングします。
- XMLSEQUENCE クラス・パラメータを 1 に設定します。この予防措置により、クラス定義で指定されたプロパティの順序に従ってマッピングが行われることが保証されます。
- プロパティが XML ドキュメント内の順序と同じ順序でクラス定義のリストに挙げられていることを確認します。

例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class GXML.TestSequence.Person Extends (%Persistent, %XML.Adaptor)
{

Property Name As %Name [ Required ];
Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h") [ Required ];
Property HomeAddress As GXML.TestSequence.Address(XMLNAME = "Address");
Property WorkAddress As GXML.TestSequence.Address(XMLNAME = "Address");

/// If the XMLSEQUENCE = 1, then the order of the XML elements must match the
/// order of the class properties.  This allows us to deal with XML where the
/// same field appears multiple times and is distinguished by the order.
Parameter XMLSEQUENCE = 1;

}
```

このクラス定義は、前述の XML ドキュメントにマッピングします。

注釈　　XMLSEQUENCE が 1 の場合、XMLIGNOREINVALIDTAG パラメータは無視されます。

9.3 エクスポート後のアンスウィズルの制御

InterSystems IRIS XML ツールを使用して、XML 対応の永続的なオブジェクトをエクスポートすると、通常どおり、必要な情報がすべてメモリに自動的にスウィズルされます。この情報にはオブジェクト値プロパティが含まれています。オブジェクトをエクスポートしたら、いずれのオブジェクト・リストもアンスウィズルされますが、(既定では) 単一のオブジェクト参照はアンスウィズルされません。大きなオブジェクトの場合は、<STORE> エラーが発生する可能性があります。

このシナリオで、いずれの単一オブジェクト参照もアンスウィズルされるようにするには、XML 対応のクラスで XMLUNSWIZZLE パラメータを以下のように設定します。

```
Parameter XMLUNSWIZZLE = 1;
```

このパラメータの既定値は 0 です。

9.4 エクスポートでの InterSystems IRIS ID の投影

InterSystems IRIS オブジェクトを (別のオブジェクトのプロパティとしてではなく) 最上位レベルで投影する際、その内部 ID、OID、およびグローバルに一意な ID はオブジェクト・プロパティとして使用できないので、これらの ID は投影されません。ただし、場合によっては、オブジェクト ID を一意の識別子として使用する必要がある場合があります。そこで、例えば、ストアド・オブジェクトを更新する前に、入力 (変更された) オブジェクトを対応するストアド・オブジェクトと照合できます。

InterSystems IRIS XML サポートでは、InterSystems IRIS オブジェクト識別子の XML ドキュメントへの投影に使用できるヘルパー・クラス、**%XML.Id** (内部 ID 用)、**%XML.Oid** (OID 用)、および **%XML.GUID** (グローバルに一意な ID 用) が用意されています。

これらのクラスを使用するには、特殊なプロパティを、エクスポートしようとする ID を含めることを目的とする XML 対応クラスに追加します。このプロパティのタイプは、**%XML.Id**、**%XML.Oid**、または **%XML.GUID** である必要があります。クラスの SQL プロジェクションに含まれないように、このプロパティを投影して、このプロパティを Transient としてマークする必要があります。

XML へのエクスポートの際は、XML 対応クラスのオブジェクトをメモリに入れます。オブジェクトがメモリに格納されると、追加した特殊プロパティが要求 ID を InterSystems IRIS 内部ストレージから取得し、その値を (エクスポートできるように) 含めます。

例えば、以下のクラスを考えてみます。

Class Definition

```
Class MyApp4.Obj.Person4 Extends (%Persistent,%Populate,%XML.Adaptor)
{

Property IdForExport As %XML.Id
(XMLNAME="IRISID", XMLPROJECTION="ELEMENT") [Private, Transient];

Property Name As %Name;

Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h");

}
```

このクラスでは、特殊なプロパティは `IdForExport` です。このプロパティは、`IRISID` という XML 要素名で投影されます。

このクラスの出力例は以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <Person>
      <IRISID>1</IRISID>
      <Name>Marks,Jules F.</Name>
      <DOB>1989-04-02</DOB>
   </Person>
   <Person>
      <IRISID>2</IRISID>
      <Name>Palmer,Angelo O.</Name>
      <DOB>1937-11-15</DOB>
   </Person>
...
```

9.5 エクスポート時のネームスペース接頭語の制御

オブジェクトの XML 出力を生成する場合、システムは適宜ネームスペース接頭語を生成しますが、必要に応じてその接頭語を指定できます。そのためには、XML 対応オブジェクトのクラス定義で以下のパラメータを設定します。

XMLPREFIX

このクラスのネームスペースと関連付けるための接頭語を指定します。

詳細は、"オブジェクトからの XML 出力の記述" を参照してください。

9.6 インポート時の予期しない要素および属性の処理

ソース XML ドキュメントには予期しない要素および属性が含まれている場合があるため、XML 対応クラスには、このようなドキュメントをインポートする際の対応方法を指定する 2 つのパラメータが用意されています。例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class GXML.TestImportParms.Person Extends (%Persistent,%XML.Adaptor)
{

Property Name As %Name [ Required ];
Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h") [ Required ];
}
```

以下の XML ドキュメントについても考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <Person employeeID="450">
      <Name>Dillard, Daniel</Name>
      <DOB>1962-09-18</DOB>
      <UserID>fr0078</UserID>
      <Address>
         <Street>810 Main Street</Street>
         <City>Reston</City>
         <State>NJ</State>
         <Zip>02641</Zip>
      </Address>
   </Person>
</Root>
```

`employeeID` 属性および `<Address>` 要素は、このクラスのプロパティには対応しないので、予期されません。

予期しない属性および要素の処理方法を指定するには、XML 対応クラスの以下のパラメータを使用します。

XMLIGNOREINVALIDATTRIBUTE

予期しない属性の処理方法を制御します。このパラメータが 1 (既定) の場合、そうした属性は無視されます。0 の場合、エラーとして扱われ、インポートは失敗します。

XMLIGNOREINVALIDTAG

予期しない要素の処理方法を制御します。このパラメータが 1 の場合、そうした要素は無視されます。0 の場合 (既定)、エラーとして扱われ、インポートは失敗します。

これらのパラメータは、インポートのみに影響します。

注釈　`xmlns` 属性、配列キーの `name` 属性、およびスキーマのインスタンス (`xsi`) の属性は常に無視されます。

また、XMLSEQUENCE が 1 の場合、XMLIGNOREINVALIDTAG パラメータは無視されます。"複数の同名のタグを含む XML ドキュメントの処理" を参照してください。

A

XML プロジェクション・パラメータの概要

ここでは、InterSystems IRIS® データ・プラットフォームの XML プロジェクション・オプションについて概説します。特に明記のない限り、クラス・パラメータは XML 対応クラスで使用可能であり、プロパティ・パラメータはこれらのクラスのプロパティで使用可能です。

トピック	パラメータ
XML プロジェクションの有効化。"InterSystems IRIS オブジェクトの XML への投影" を参照してください。	XMLENABLED クラス・パラメータ
要素や属性へのプロパティのマッピング。"オブジェクトの XML への投影" を参照してください。	· XMLPROJECTION プロパティ・パラメータ ("NONE"、"ATTRIBUTE"、"XMLATTRIBUTE"、"CONTENT"、"ELEMENT"、または "WRAPPED") · XMLSUMMARY クラス・パラメータ · XMLDEFAULTREFERENCE クラス・パラメータ ("SUMMARY"、"COMPLETE"、"ID"、"OID"、または "GUID") · XMLREFERENCE プロパティ・パラメータ ("SUMMARY"、"COMPLETE"、"ID"、"OID"、または "GUID")
XML 要素名と属性名。"XML の要素名と属性名の制御" を参照してください。	· XMLNAME クラス・パラメータ · XMLNAME プロパティ・パラメータ · XMLITEMNAME プロパティ・パラメータ · XMLKEYNAME プロパティ・パラメータ 既定値は、XML タイプ名に基づいています。
XML タイプ。"XML スキーマへの投影の制御" を参照してください。	· XMLTYPE クラス・パラメータ · XMLTYPE プロパティ・パラメータ · XSDTYPE クラス・パラメータ

トピック	パラメータ
ネームスペース。"要素と属性に対するネームスペースの指定" を参照してください。	· NAMESPACE クラス・パラメータ · ELEMENTQUALIFIED クラス・パラメータ (0 または 1) このパラメータはエクスポート時にオーバーライドできます。 · ELEMENTQUALIFIED プロパティ・パラメータ (0 または 1) このパラメータはエクスポート時にオーバーライドできます。 · ATTRIBUTEQUALIFIED クラス・パラメータ (0 または 1) このパラメータはエクスポート時にオーバーライドできます。 · XMLREF プロパティ・パラメータ (0 または 1) · REFNAMESPACE プロパティ・パラメータ · XMLPREFIX クラス・パラメータ
空文字列および NULL。"空文字列および NULL 値の処理" を参照してください。	· XMLUSEEMPTYELEMENT クラス・パラメータ (0 または 1) · XMLIGNORENULL クラス・パラメータ (0、1、"INPUTONLY"、または "RUNTIME") · XMLNIL クラス・パラメータ (0 または 1) XMLNIL プロパティ・パラメータ (0 または 1) XMLIGNORENULL が "RUNTIME" の場合は、XMLNIL をエクスポート時またはインポート時にオーバーライドできます。 · XMLNILNOOBJECT クラス・パラメータ (0 または 1) · XMLNILNOOBJECT プロパティ・パラメータ (0 または 1)
XML 特殊文字のエスケープ。"XML 特殊文字の処理" を参照してください。	· CONTENT パラメータ ("STRING"、"ESCAPE"、"ESCAPE-C14N"、または "MIXED") · ESCAPE パラメータ ("XML" または "HTML")
タイム・ゾーン。"UTC タイム・ゾーン・インジケータの処理" を参照してください。	XMLTIMEZONE プロパティ・パラメータ ("UTC" または "IGNORE")
XML タイプの詳細情報 (制限事項を含む)。"XML スキーマへの投影の制御" および "XML スキーマの詳細オプション" を参照してください。	· XMLTYPECONSTRAINT プロパティ・パラメータ ("EXPLICIT"、"CHOICE"、または "SUBSTITUTIONGROUP") · XMLINCLUDEINGROUP クラス・パラメータ (0 または 1) · XMLCHOICELIST プロパティ・パラメータ · XMLINHERITANCE クラス・パラメータ ("left" または "right") · 多数の InterSystems IRIS データ型プロパティ・パラメータ

トピック	パラメータ
入力、出力、または入出力用のプロパティの使用。"投影されるプロパティの可用性の制御" を参照してください。	XMLIO プロパティ・パラメータ ("INOUT"、"IN"、"OUT"、または "CALC")
使用可能な XML ドキュメント形式の制御。"XML ドキュメントのフォーマット・オプションの指定" を参照してください。	XMLFORMAT クラス・パラメータ ("LITERAL"、"ENCODED"、または NULL (両方の形式の場合))
同じ名前の複数の要素。"複数の同名の要素を含む XML ドキュメントの処理" を参照してください。	XMLSEQUENCE クラス・パラメータ (0 または 1)
ストリーム・プロパティ。"ストリーム・プロパティの行末の制御" を参照してください。	XMLSTREAMMODE プロパティ・パラメータ ("BLOCK" または "LINE")
予期しない要素および属性。"インポート時の予期しない要素および属性の処理" を参照してください。	· XMLIGNOREINVALIDTAG クラス・パラメータ (0 または 1) · XMLIGNOREINVALIDATTRIBUTE クラス・パラメータ (0 または 1)
ネームスペース接頭語。"エクスポート時のネームスペース接頭語の制御" を参照してください。	XMLPREFIX クラス・パラメータ
スキーマにおける pattern 制限の指定。	XMLPATTERN プロパティ・パラメータ

その他のパラメータ : XMLELEMENTREF プロパティ・パラメータ。このプロパティ・パラメータは廃止されています (XMLREF に置き換えられました) が、永久にサポートされます。