XML ツールの使用法

Version 2024.1
2024-06-03

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

XML ツールの使用法
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

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

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

目次

テーブル一覧

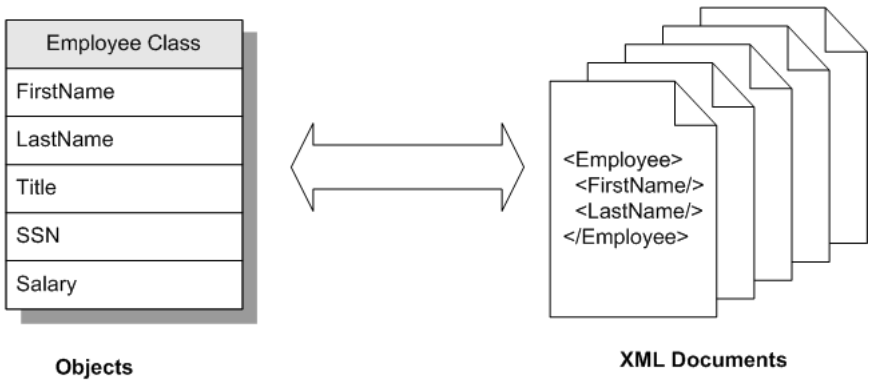
1

InterSystems XML ツールの概要

InterSystems IRIS® データ・プラットフォームでは、オブジェクトを XML ドキュメントの直接表現として使用でき、XML ドキュメントの直接表現をオブジェクトとして使用できます。InterSystems IRIS には、ネイティブのオブジェクト・データベースが含まれているため、このようなオブジェクトを直接データベース内で使用できます。さらに、InterSystems IRIS には、任意の XML ドキュメントと DOM (ドキュメント・オブジェクト・モデル) を操作するためのツールが用意されています。ドキュメントが InterSystems IRIS のいずれのクラスにも関連していない場合でも同じです。

1.1 XML でのオブジェクト・データの表現

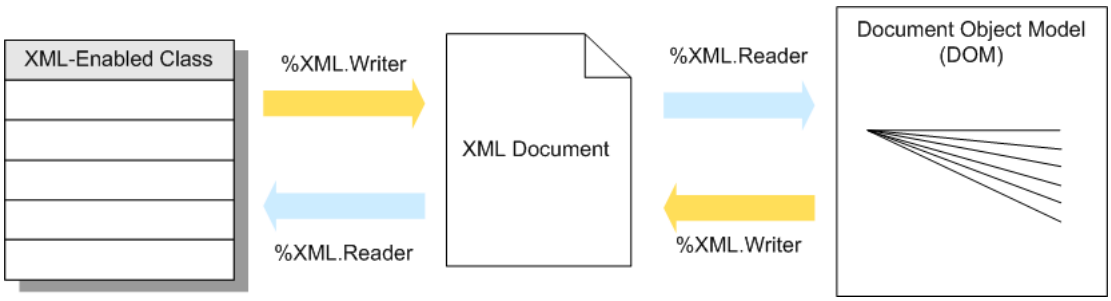
XML ツールの一部は、主として XML 対応のクラスと併用することを目的としています。クラスを XML 対応にするには、そのスーパークラス・リストに %XML.Adaptor を追加します。%XML.Adaptor クラスを使用すると、そのクラスのインスタンスを XML ドキュメントとして表すことができます。クラス・パラメータとプロパティ・パラメータを追加して、プロジェクションを微調整します。"オブジェクトの XML への投影" を参照してください。



XML 対応のクラスでは、データは以下のすべての形式で使用できます。

- クラス・インスタンスに含まれています。クラスによっては、データをディスクに保存することもでき、ディスク上では、他の永続クラスとまったく同じように使用できます。
- XML ドキュメントに含まれています。これは、ファイルやストリームなどのドキュメントの場合もあります。
- DOM に含まれています。

以下の図は、これらの形式の間でデータを変換するために使用するツールの概要です。



XML 対応クラスのコンテキストで、**%XML.Writer** クラスは、XML ドキュメントとしてのデータのエクスポートを可能にします。出力先は通常、ファイルまたはストリームです。

%XML.Reader を使用すると、クラスのインスタンスに適切な XML ドキュメントをインポートできます。ソースは通常、ファイルまたはストリームです。このクラスを使用するには、クラス名と XML ドキュメントに含まれている要素との間の相関関係を指定します。指定された要素は、対応するクラスで予想される構造になっている必要があります。次に、ノードごとにドキュメントを読み取ります。読み取ると、そのクラスのメモリ内インスタンスが作成され、そこには XML ドキュメント内で見つかったデータが含まれます。

XML ドキュメントを操作する場合には、DOM も役に立ちます。**%XML.Reader** クラスを使用すると、XML ドキュメントを読み取り、それを表す DOM を作成できます。この表現では、DOM は一連のノードであり、必要に応じてノード間を移動します。具体的には、**%XML.Document** のインスタンスを作成します。これはドキュメント自体を表し、ノードを含みます。その後、**%XML.Node** を使用し、ノードを検査して操作します。必要に応じて、**%XML.Writer** を使用して XML ドキュメントを再度記述します。

InterSystems IRIS XML ツールには、XML ドキュメントと DOM にあるデータにアクセスしたり、この両方を変更したりする方法が数多く用意されています。

1.2 任意の XML の作成

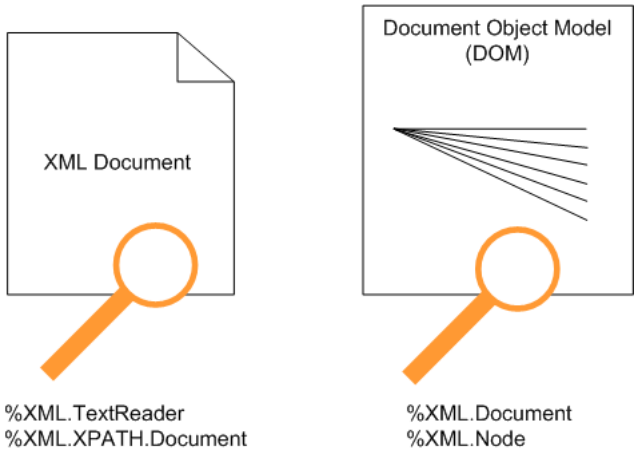
任意の XML (つまり、いずれの InterSystems IRIS クラスにもマップしない XML) の作成と操作にも InterSystems IRIS XML ツールを使用できます。任意の XML ドキュメントを作成するには、**%XML.Writer** を使用します。このクラスには、要素の追加、属性の追加、ネームスペース宣言の追加などを行うメソッドが用意されています。

任意の DOM を作成するには、**%XML.Document** を使用します。このクラスには、1 つの空のノードと共に DOM を返すクラス・メソッドが用意されています。次に、必要に応じて、そのクラスのインスタンス・メソッドを使用してノードを追加します。

または、**%XML.Reader** を使用して任意の XML ドキュメントを読み取り、そのドキュメントから DOM を作成します。

1.3 データへのアクセス

InterSystems IRIS XML ツールには、XML 形式のデータにアクセスする方法がいくつか用意されています。次の図に概要を示します。



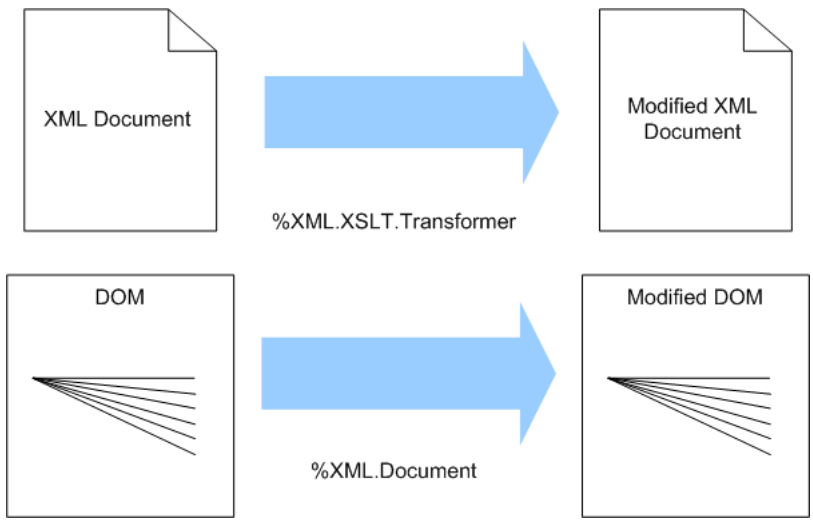
適格な XML ドキュメントでは、以下のクラスを使用して、そのドキュメント内のデータを操作できます。

- **%XML.TextReader** – これは、ノードごとにドキュメントを読み取って解析する場合に使用できます。"%XML.TextReader の使用" を参照してください。
- **%XML.XPATH.Document** – これは、ドキュメント内の具体的なノードを参照する XPATH 式を使用することによって、データを取得する場合に使用できます。"XPath 式の評価" を参照してください。

InterSystems IRIS では、DOM は **%XML.Document** のインスタンスです。このインスタンスはドキュメント自体を表し、ノードを含みます。DOM から値を取得するには、このクラスのプロパティおよびメソッドを使用します。ノードの検査と操作には、**%XML.Node** を使用します。詳細は、"XML ドキュメントの DOM 表現" を参照してください。

1.4 XML の変更

InterSystems IRIS XML ツールには、XML 形式のデータを変更する方法も用意されています。次の図に概要を示します。



XML ドキュメントでは、**%XML.XSLT.Transformer** のクラス・メソッドを使用して、XSLT 変換を実行し、変更後のドキュメントを取得できます。"XSLT 変換の実行" を参照してください。

DOM では、**%XML.Document** のメソッドを使用して DOM を変更できます。例えば、要素または属性を追加するか、または削除することができます。

1.5 SAX パーサ

InterSystems IRIS XML ツールでは、InterSystems IRIS SAX (Simple API for XML) パーサが使用されます。これは、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。SAX は、完全な XML 検証とドキュメントの解析を提供する解析エンジンです。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。InterSystems IRIS の組み込み XML サポートを使用するか、または InterSystems IRIS 内で独自のカスタム SAX インタフェース・クラスを提供することで、このパーサを使用して XML ドキュメントを処理できるようになります。

特殊なアプリケーションの場合は、カスタムのエンティティ・リゾルバとコンテンツ・ハンドラを作成できます。業界標準の XML DTD またはスキーマ検証を使用して、受信 XML を検証したり、解析対象の XML 項目を指定することもできます。"SAX パーサの使用法のカスタマイズ" を参照してください。

1.6 追加の XML ツール

InterSystems IRIS XML サポートには、以下の追加のツールが含まれています。

- XML スキーマ・ウィザードは、XML スキーマのドキュメントを読み取り、そのスキーマで定義されたタイプに対応する XML 対応クラスのセットを生成します。クラスを収めるパッケージと、クラス定義の詳細を制御するさまざまなオプションを指定します。
- %XML.Schema クラスを使用すると、XML 対応クラスのセットから XML スキーマを生成できます。
- %XML.Namespaces クラスを使用すると、XML ネームスペースと、それらのネームスペース内のクラスを、InterSystems IRIS ネームスペースに対して検証することができます。
- %XML.Security.EncryptedData などのクラスを使用すると、XML ドキュメントの暗号化、および暗号化されたドキュメントの解読を行うことができます。
- %XML.Security.Signature などのクラスを使用すると、XML ドキュメントのデジタル署名、およびデジタル・シグニチャの検証を行うことができます。

2

XML ツールを使用する場合の検討事項

任意の種類の XML ツールを操作する場合には、以下のように、一般的に検討すべき事項が少なくとも 3 つあります。

- どのような XML ドキュメントにも文字エンコードがあります。
- XML ドキュメントを (リテラル、または SOAP エンコードの) クラスにマップするには、さまざまな方法があります。
- SAX パーサの既定の動作を理解しておく必要があります。

2.1 入出力の文字エンコード

XML ドキュメントをエクスポートするときには、使用する文字エンコードを指定できます。指定しなかった場合は、エクスポート先に応じて、InterSystems IRIS® データ・プラットフォームによってエンコードが選択されます。

- 出力先がファイルまたはバイナリ・ストリームの場合は、"UTF-8" が既定です。
- 出力先が文字列または文字ストリームの場合は、"UTF-16" が既定です。

InterSystems IRIS によって読み取られた任意の XML ドキュメントでは、ドキュメントの XML 宣言にそのファイルの文字エンコードを明記する必要があり、明記しておけば、ドキュメントは宣言どおりにエンコードされるようになります。以下はその例です。

```
<?xml version="1.0" encoding="UTF-16"?>
```

ただし、文字エンコードがドキュメントで宣言されていない場合は、以下のように想定されます。

- ドキュメントがファイルまたはバイナリ・ストリームの場合は、文字セットが "UTF-8" と想定されます。
- ドキュメントが文字列または文字ストリームの場合は、文字セットが "UTF-16" と想定されます。

文字セットおよび変換テーブルの詳細は、″変換テーブル″ を参照してください。

2.2 ドキュメント形式の選択

XML ドキュメントを操作するときには、ドキュメントを InterSystems IRIS のクラスにマップするときに使用する形式を認識しておく必要があります。同様に、XML ドキュメントを作成するときには、ドキュメントを記述するときに使用するドキュメント形式を指定します。XML ドキュメント形式は、以下のとおりです。

- リテラルとは、ドキュメントがオブジェクト・インスタンスのリテラル・コピーであることを示します。ほとんどの場合、作業対象が SOAP の場合でもリテラル形式を使用します。

  別途明記されている場合を除き、このドキュメントに挙げている例ではリテラル形式を使用しています。

- エンコードとは、SOAP 1.1 規格または SOAP 1.2 規格で説明されている方法でエンコードされていることを示します。これらの規格へのリンクは、"XML 標準" を参照してください。

  SOAP 1.1 と SOAP 1.2 では、詳細が少し異なります。

以下のサブセクションに、これらのドキュメント形式の違いを示します。

2.2.1 リテラル形式

以下の例では、XML ドキュメントをリテラル形式で表示します。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <Person>
      <Name>Klingman,Julie G.</Name>
      <DOB>1946-07-21</DOB>
      <GroupID>W897</GroupID>
      <Address>
         <City>Bensonhurst</City>
         <Zip>60302</Zip>
      </Address>
      <Doctors>
         <DoctorClass>
            <Name>Jung,Kirsten K.</Name>
         </DoctorClass>
         <DoctorClass>
            <Name>Xiang,Charles R.</Name>
         </DoctorClass>
         <DoctorClass>
            <Name>Frith,Terry R.</Name>
         </DoctorClass>
      </Doctors>
   </Person>
</Root>
```

2.2.2 エンコード形式

一方、以下の例では同じデータをエンコード形式で表示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Root 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">
...
   <DoctorClass id="id2" xsi:type="DoctorClass">
      <Name>Jung,Kirsten K.</Name>
   </DoctorClass>
...
   <DoctorClass id="id3" xsi:type="DoctorClass">
      <Name>Quixote,Umberto D.</Name>
   </DoctorClass>
...
   <DoctorClass id="id8" xsi:type="DoctorClass">
      <Name>Chadwick,Mark L.</Name>
   </DoctorClass>
...
   <Person>
      <Name>Klingman,Julie G.</Name>
      <DOB>1946-07-21</DOB>
      <GroupID>W897</GroupID>
      <Address href="#id17" />
      <Doctors SOAP-ENC:arrayType="DoctorClass[3]">
         <DoctorClass href="#id8" />
         <DoctorClass href="#id2" />
         <DoctorClass href="#id3" />
      </Doctors>
```

```
   </Person>
   <AddressClass id="id17" xsi:type="s_AddressClass">
      <City>Bensonhurst</City>
      <Zip>60302</Zip>
   </AddressClass>
...
</Root>
```

エンコードされたバージョンでは、以下の違いに注意してください。

- 出力のルート要素に、SOAP エンコードのネームスペースおよび他の標準ネームスペースの宣言が含まれます。
- このドキュメントには、person、address、および doctor の各要素がすべて同じレベルで含まれます。address 要素および doctor 要素は、この 2 つの要素を参照する person 要素が使用する一意の ID でリスト表示されます。各オブジェクト値プロパティはこの方法で処理されます。
- 最上位レベルの address および doctor 要素の名前は、それらの要素を参照するプロパティと同じ名前ではなく、それぞれのクラスの名前と同じになります。
- エンコード形式には、属性は含まれません。`GroupID` プロパティは、`Person` クラスの属性としてマッピングされます。リテラル形式では、このプロパティは属性として投影されます。ただし、エンコードされたバージョンでは、このプロパティは要素として投影されます。
- コレクションに対する処理は異なります。例えば、リスト要素には属性 `ENC:arrayType` があります。
- 各要素は `xsi:type` 属性の値を持ちます。

注釈　SOAP 1.2 の場合、エンコード・バージョンは若干異なります。バージョンを簡単に区別するには、SOAP エンコードのネームスペースの宣言を確認します。

- SOAP 1.1 の場合、SOAP エンコードのネームスペースは `"http://schemas.xmlsoap.org/soap/encoding/"` です。
- SOAP 1.2 の場合、SOAP エンコードのネームスペースは `"http://schemas.xmlsoap.org/wsdl/soap12/"` です。

2.3 パーサの動作

InterSystems IRIS SAX パーサは、InterSystems IRIS により XML ドキュメントが読み取られるたびに使用されるので、その既定の動作を知っておくと役に立ちます。パーサは、以下のようなタスクを行います。

- XML ドキュメントが適格な文書であるかどうかを検証します。
- 指定されたスキーマまたは DTD を使用して、ドキュメントを検証しようとします。

  ここでは、1 つのスキーマには、他のスキーマを参照する <import> 要素および <include> 要素を含めることができる、ということを覚えておくと役に立ちます。以下はその例です。

  ```
  <xsd:import namespace="target-namespace-of-the-importing-schema"
                    schemaLocation="uri-of-the-schema"/>

  <xsd:include schemaLocation="uri-of-the-schema"/>
  ```

  これらの他のスキーマをパーサで使用できる場合以外は、検証は失敗します。特に WSDL ドキュメントの場合は、すべてのスキーマをダウンロードして、修正された場所を使用するように主スキーマを編集することが必要になる場合もあります。
- すべての外部エンティティを含め、すべてのエンティティを解決しようとします (この作業は他の XML パーサでも行います)。場所によっては、このプロセスには時間がかかる場合もあります。特に、Xerces では一部の URL の解決

にネットワーク・アクセサが使用され、実装ではブロックする I/O が使用されます。結果的に、タイムアウトは発生せず、ネットワーク・フェッチがエラー状態になって停止する可能性があります (現実にはまず発生しません)。

また、Xerces では https をサポートしていないので、https に位置するエンティティの解析はできません。

必要に応じて、カスタムのエンティティ・リゾルバを作成したり、エンティティの解析を無効にすることができます。

3

XML ドキュメントの読み取りと検証

InterSystems IRIS® データ・プラットフォームを使用すれば、XML ドキュメントをいくつかの方法で読み取って使用できます。このページでは、XML ドキュメントの概要を示し、XML ドキュメントの検証方法を説明します。

3.1 基本的な手法

InterSystems IRIS では、3 つの基本的な方法で XML ドキュメントを読み取って解析できます。

- どの XML ドキュメントにも **%XML.TextReader** を使用できます (別のトピックで説明しています)。この API を使用すると、ドキュメントを読み取ることで **%XML.TextReader** のインスタンスが作成され、それを使用してドキュメントをノード単位で検査できます。
- どの XML ドキュメントにも **%XML.Reader** を使用して、XML ドキュメント・オブジェクト・モデル (DOM) としてアクセスできます。
- XML ドキュメントが XML 対応のクラス定義に適切にマッピングされていれば、**%XML.Reader** クラスを使用して、そのクラスのインスタンスにドキュメントをインポートできます (別のトピックで説明しています)。

%XML.TextReader と **%XML.Reader** の両方で、InterSystems IRIS の SAX (Simple API for XML) パーサが使用されます。

3.2 ドキュメントが整形式であることの確認

XML ドキュメントを読み取るときは、同時に検証も実行することが一般的です。つまり、SAX パーサはドキュメントが整形式であることを確認し、必要に応じて他のタスクを実行すると共に、宣言済みのスキーマまたは DTD との照合によってドキュメントを検証します。

XML ドキュメントが整形式であるかどうかの確認のみが必要な場合は、以下のように **%XML.TextReader** を使用します。

1. 以下のメソッドのいずれかの最初の引数を使用して、ドキュメント・ソースを指定します。

メソッド	最初の引数
ParseFile()	完全なパスを含むファイル名ファイル名およびパスは ASCII 文字のみを含む必要があります。
ParseStream()	ストリーム
ParseString()	文字列
ParseURL()	URL

2. 解析メソッドから返されたステータスを確認します。ステータスにエラー・コードがある場合、そのエラー・コードに問題の場所が示されています。

   ドキュメントに複数のエラーが存在することがありますが、ドキュメントをこれ以上読み取ることができなくなった時点でパーサは終了します。

例：

```
USER>set file="C:\0work\XMLdemo\inputfile2.xml"

USER>set status=##class(%XML.TextReader).ParseFile(file)

USER>write status=1
1
USER>set file="C:\0work\XMLdemo\inputfile2a.xml"

USER>set status=##class(%XML.TextReader).ParseFile(file)

USER>write status=1
0
USER>d $system.OBJ.DisplayError(status)

ERROR #6301: SAX XML Parser Error: expected end of tag 'xlistitem' while processing
C:\0work\XMLdemo\inputfile2a.xml at line 423 offset 3
```

3.3 ドキュメントがスキーマまたは DTD に従っていることの確認

必要に応じ、%XML.TextReader を使用して、宣言済みのスキーマまたは DTD との照合によってドキュメントを検証することもできます。そのためには、前のセクションで説明したように ParseFile()、ParseStream()、ParseString()、または ParseURL() を呼び出しますが、この場合は 2 番目の引数 TextReader も指定します。この引数は出力として返され、%XML.TextReader のインスタンスです。この引数を使用してドキュメントを反復処理し、エラーと警告を検出できます。

例：

Class Member

```
ClassMethod ValidateFile(file As %String = "C:\0work\XMLdemo\inputfile2.xml",schema as %String="")
{
    write !!,"Validating "_file_"..."

    if (schema="") {
        //in this case, use the schema that the file refers to
        Set status=##class(%XML.TextReader).ParseFile(file,.tReader,,flags)
    } else {
        //use an override schema
        Set status=##class(%XML.TextReader).ParseFile(file,.tReader,,flags,,schema)
    }
    if $$$ISERR(status) {
        do $system.OBJ.DisplayError(status)
    }
    if '$ISOBJECT(tReader) {
        write !, ">>> Cannot read this file, because it is not valid..."
        quit
    }
```

```
    set errcount=0
    set warningcount=0
    while (tReader.Read()) {
        if (tReader.NodeType="error") {
            set errcount=errcount+1
            Write !, ">>> *ERROR* ",tReader.Value
        } elseif (tReader.NodeType="warning") {
            set warningcount=warningcount+1
            Write !, ">>> *WARNING* ",tReader.Value
        }
    }
    if (errcount=0) && (warningcount=0) {
        write !, ">>> No warnings or errors"
    }
}
```

以下の追加のメソッドを使用することで、ディレクトリにある各ファイルを再帰的にスキャンできます。

Class Member

```
ClassMethod ValidateFilesInDir(dirtoprocess As %String = "C:\0work\XMLdemo",schema as %String="")
{
    set stmt = ##class(%SQL.Statement).%New()
    set status = stmt.%PrepareClassQuery("%File","FileSet")
    if $$$ISERR(status) {
        do $system.OBJ.DisplayError(status)
        quit
    }

    set rset = stmt.%Execute(dirtoprocess,"*.xml",,1)
    while rset.%Next() {
        set filetoprocess=rset.%Get("Name")
        set type=rset.%Get("Type")
        if (type="F") {
            do ..ValidateFile(filetoprocess,schema)
        } elseif (type="D") {
            set dirname=rset.%Get("Name")
            do ..ValidateFilesInDir(dirname)
        }
    }
}
```

"%XML.TextReader の使用" も参照してください。

4

%XML.TextReader の使用

%XML.TextReader クラスを使用すると、InterSystems IRIS® データ・プラットフォーム・オブジェクトに直接マップされているかどうかに関係なく、簡潔で容易な方法で任意の XML ドキュメントを読み取ることができます。具体的には、このクラスを使用すると、適格な形式の XML ドキュメントをナビゲートし、その中の情報 (要素、属性、コメント、ネームスペース URI など) を表示できます。このクラスは、DTD、もしくは XML スキーマを基にした、ドキュメントの完全な検証も提供します。ただし、%XML.Reader と異なり、%XML.TextReader には DOM を返す方法が用意されていません。DOM が必要な場合は、"オブジェクトへの XML のインポート" を参照してください。

注釈　使用するどの XML ドキュメントの XML 宣言にも、そのドキュメントの文字エンコードを明記する必要があります。これにより、ドキュメントが宣言どおりにエンコードされます。文字エンコードが宣言されていない場合は、"入出力の文字エンコード" で説明されている既定値が使用されます。これらの既定値が正しくない場合は、XML 宣言を修正して、実際に使用されている文字セットを指定するようにします。

4.1 任意の XML の読み取り

InterSystems IRIS オブジェクト・クラスとのリレーションシップが必ずしもあるとは限らない任意の XML ドキュメントを読み取るには、%XML.TextReader クラスのメソッドを呼び出します。これにより、ドキュメントを開き、テキスト・リーダ・オブジェクトとして一時的な格納場所にロードします。テキスト・リーダ・オブジェクトには、ナビゲート可能なノードのツリーが含まれ、そのそれぞれにソース・ドキュメントについての情報が含まれています。そのため、記述したメソッドでドキュメントをナビゲートし、ドキュメントに関する情報を検出できます。このオブジェクトのプロパティは、ドキュメント内の現在の場所に応じて、そのドキュメントに関する情報を提示します。検証エラーがある場合、それらのエラーはツリー内のノードとしても表示できます。

4.1.1 全体構造

メソッドは以下の操作の一部またはすべてを実行する必要があります。

1. 以下のメソッドのいずれかの最初の引数を使用して、ドキュメント・ソースを指定します。

メソッド	最初の引数
ParseFile()	完全なパスを含むファイル名ファイル名およびパスは ASCII 文字のみを含む必要があります。
ParseStream()	ストリーム
ParseString()	文字列
ParseURL()	URL

どのような場合でも、ソース・ドキュメントは、適格な XML ドキュメント、つまり、XML 構文の基本的な規約に従ったドキュメントである必要があります。これらのメソッドはそれぞれ、結果が成功だったかどうかを示すステータス (**$$$OK** または失敗コード) を返します。通常のメカニズムでステータスをテストできます。特に、$System.Status.DisplayError(status) を使用して、エラー・メッセージのテキストを表示できます。

メソッドは **$$$OK** を返す場合、これらのメソッドのそれぞれに対し、参照 (2 番目の引数) によって、XML ドキュメント内の情報を含むテキスト・リーダ・オブジェクトを返します。

引数をさらに追加すると、エンティティの解析、検証、検出された項目などを制御できます。"解析メソッドの引数リスト" を参照してください。

2. 解析メソッドで返されたステータスを確認し、必要に応じて実行を中止します。

   解析メソッドが **$$$OK** を返した場合、テキスト・リーダ・オブジェクトはソース XML ドキュメントに対応しています。このオブジェクトはナビゲートできます。

   ドキュメントには多くの場合、`"element"`、`"endelement"`、`"startprefixmapping"` などのノードが含まれています。"ノード・タイプ" に、ノード・タイプの一覧が記載されています。

   重要　どのような検証エラーが発生した場合でも、ドキュメントには `"error"` ノードまたは `"warning"` ノードが含まれています。そのようなノードがないか、コードで確認する必要があります。"検証の実行" を参照してください。

3. 以下のインスタンス・メソッドのいずれかを使用して、ドキュメントの読み取りを開始します。
   - ドキュメントの最初のノードへ移動するには、Read() を使用します。
   - 特定のタイプの最初の要素へ移動するには、ReadStartElement() を使用します。
   - `"chars"` の最初のノードへ移動するには、MoveToContent() を使用します。

   "ドキュメントのナビゲート" を参照してください。

4. このノードで関係するプロパティの値があれば、それを取得します。利用可能なプロパティには、Name、Value、Depth などがあります。"ノード・プロパティ" を参照してください。

5. 必要に応じてドキュメントのナビゲートおよびプロパティ値の取得を続けます。

   現在のノードが要素の場合、MoveToAttributeIndex() または MoveToAttributeName() メソッドを使用して、要素の属性にフォーカスを移動できます。要素に戻るには、該当する場合は MoveToElement() を使用します。

6. 必要に応じて、Rewind() メソッドを使用して、ドキュメントの最初 (最初のノードの前) に戻ります。これは、ソース内を後退できる唯一のメソッドです。

メソッドの実行後、テキスト・リーダは消滅し、関連する一時格納場所もすべてクリーンアップされます。

4.1.2 例 1

ここでは、任意の XML ファイルを読み取り、各ノードのシーケンス番号、タイプ、名前、および値を表示する単純なメソッドを示します。

Class Member

```
ClassMethod WriteNodes(myfile As %String)
{
    set status=##class(%XML.TextReader).ParseFile(myfile,.textreader)
    //check status
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}
    //iterate through document, node by node
    while textreader.Read()
    {
        Write !, "Node ", textreader.seq, " is a(n) "
        Write textreader.NodeType," "
        If textreader.Name'=""
        {
            Write "named: ", textreader.Name
            }
            Else
            {
                Write "and has no name"
                }
        Write !, "    path: ",textreader.Path
        If textreader.Value'=""
        {
            Write !, "    value: ", textreader.Value
            }
        }
}
```

この例は、以下を実行します。

1. ParseFile() クラス・メソッドを呼び出します。このメソッドはソース・ファイルを読み取り、テキスト・リーダ・オブジェクトを生成して、変数 doc 内でそのオブジェクトを参照によって返します。
2. ParseFile() が成功した場合、このメソッドは Read() メソッドを実行して、ドキュメント内で次のノードをそれぞれ検索します。
3. 各ノードについて、このメソッドは、そのノードのシーケンス番号、ノード・タイプ、ノード名 (存在する場合)、ノード・パス、およびノード値 (存在する場合) が含まれた出力行を記述します。現在のデバイスに出力されます。

以下の例のソース・ドキュメントを考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="mystyles.css"?>
<Root>
   <s01:Person xmlns:s01="http://www.root.org">
      <Name attr="xyz">Willeke,Clint B.</Name>
      <DOB>1925-10-01</DOB>
   </s01:Person>
</Root>
```

このソース・ドキュメントについては、前述のメソッドで以下の出力が生成されます。

```
Node 1 is a(n) processinginstruction named: xml-stylesheet
    path:
    value: type="text/css" href="mystyles.css"
Node 2 is a(n) element named: Root
    path: /Root
Node 3 is a(n) startprefixmapping named: s01
    path: /Root
    value: s01 http://www.root.org
Node 4 is a(n) element named: s01:Person
    path: /Root/s01:Person
Node 5 is a(n) element named: Name
    path: /Root/s01:Person/Name
Node 6 is a(n) chars and has no name
```

```
    path: /Root/s01:Person/Name
    value: Willeke,Clint B.
Node 7 is a(n) endelement named: Name
    path: /Root/s01:Person/Name
Node 8 is a(n) element named: DOB
    path: /Root/s01:Person/DOB
Node 9 is a(n) chars and has no name
    path: /Root/s01:Person/DOB
    value: 1925-10-01
Node 10 is a(n) endelement named: DOB
    path: /Root/s01:Person/DOB
Node 11 is a(n) endelement named: s01:Person
    path: /Root/s01:Person
Node 12 is a(n) endprefixmapping named: s01
    path: /Root
    value: s01
Node 13 is a(n) endelement named: Root
    path: /Root
```

コメントが無視されていることに注意してください。既定では、%XML.TextReader クラスはコメントを無視します。この変更の詳細は、"解析メソッドの引数リスト" で説明します。

4.1.3 例 2

以下の例では、XML ファイルを読み取り、その中の各要素をリスト表示します。

Class Member

```
ClassMethod ShowElements(myfile As %String)
{
    set status = ##class(%XML.TextReader).ParseFile(myfile,.textreader)
    //check status
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}
    //iterate through document, node by node
    while textreader.Read()
    {
        if (textreader.NodeType = "element")
        {
            write textreader.Name,!
            }
        }
}
```

このメソッドは、NodeType プロパティを使用して、各ノードのタイプをチェックします。そのノードが要素の場合、メソッドはその名前を現在のデバイスに出力します。前述の XML ソース・ドキュメントについては、このメソッドで以下の出力が生成されます。

```
Root
s01:Person
Name
DOB
```

4.2 ノード・タイプ

ドキュメントのノードはそれぞれ、以下のタイプのいずれかになります。

テーブル 4-1: テキスト・リーダ・ドキュメントのノード・タイプ

タイプ	説明
"attribute"	XML 属性

タイプ	説明
"chars"	文字一式 (要素のコンテンツなど) %XML.TextReader クラスは、他のノード・タイプ ("CDATA"、"EntityReference"、および "EndEntity") を認識しますが、それらを "chars" に自動的に変換します。
"comment"	XML コメント
"element"	XML 要素の先頭
"endelement"	XML 要素の終了
"endprefixmapping"	ネームスペースが宣言されているコンテキストの終了
"entity"	XML エンティティ
"error"	パーサで検出された検証エラー。"検証の実行" を参照してください。
"ignorablewhitespace"	混在するコンテンツ・モデルのマークアップ間にある空白
"processinginstruction"	XML 処理命令
"startprefixmapping"	ネームスペースを含むまたは含まない可能性がある、XML ネームスペース宣言
"warning"	パーサで検出された検証の警告。"検証の実行" を参照してください。

1 つの XML 要素が複数のノードで構成されていることに注意してください。例えば、以下の XML フラグメントを考えてみます。

XML

```
<Person>
   <Name>Willeke,Clint B.</Name>
   <DOB>1925-10-01</DOB>
</Person>
```

SAX パーサは、この XML を以下のノードのセットとして表示します。

テーブル 4-2: ドキュメント・ノードの例

ノード番号	ノードのタイプ	ノード名 (存在する場合)	ノードの値 (存在する場合)
1	element	Person	
2	element	Name	
3	chars		Willeke,Clint B.
4	endelement	Name	
5	element	DOB	
6	chars		1925-10-01
7	endelement	DOB	
8	endelement	Person	

例えば <DOB> 要素は、element ノード、chars ノード、および endelement ノードの 3 つのノードであると見なされることに注意してください。また、この要素のコンテンツは、chars ノードの値としてのみ使用できることにも注意してください。

4.3 ノード・プロパティ

%XML.TextReader クラスは XML ドキュメントを解析し、ドキュメントのコンポーネントに対応するノードのセットで構成されるテキスト・リーダ・オブジェクトを作成します。ノード・タイプについては、"ドキュメント・ノード" を参照してください。

別のノードにフォーカスを変更すると、そのテキスト・リーダ・オブジェクトのプロパティが更新され、現在調べているノードについての情報が含められます。ここでは、%XML.TextReader クラスのすべてのプロパティについて説明します。

AttributeCount

現在のノードが要素または属性の場合、このプロパティは、要素の属性の番号を示します。任意の要素内で最初の属性の番号は 1 となります。

その他のタイプのノードでは、このプロパティは 0 になります。

Depth

ドキュメント内での現在のノードの深さを示します。ルート要素の深さは 1 です。ルート要素外の項目の深さは 0 です。属性は、それが属する要素と同じ深さにあることに注意してください。同様に、エラーまたは警告は、そのエラーまたは警告を引き起こした項目と同じ深さになります。

EOF

リーダがソース・ドキュメントの末尾に到達した場合は True、それ以外の場合は False です。

HasAttributes

現在のノードが要素の場合、その要素に属性があれば、このプロパティは True です (属性がなければ False です)。現在のノードが属性の場合、このプロパティは True です。

その他のタイプのノードでは、このプロパティは False になります。

HasValue

現在のノードが値を持つタイプのノードの場合 (その値が Null であっても)、True です。それ以外の場合、このプロパティは False です。具体的には、このプロパティは以下のタイプのノードでは True となります。

- attribute
- chars
- comment
- entity
- ignorablewhitespace
- processinginstruction
- startprefixmapping

エラーおよび警告のタイプのノードについては、それらのノード・タイプに値があっても、HasValue は False となることに注意してください。

IsEmptyElement

現在のノードが要素であり、空白である場合には True です。それ以外の場合、このプロパティは False です。

LocalName

attribute、element、または endelement のタイプのノードの場合、これは現在の要素または属性の名前からネームスペースの接頭語を除いたものとなります。その他すべてのタイプのノードでは、このプロパティは Null になります。

Name

ノードのタイプに応じた、現在のノードの完全修飾名です。以下のテーブルに詳細を示します。

テーブル 4-3: タイプごとのノード名

ノード・タイプ	名前と例

attribute	属性の名前。例えば、次のような属性の場合、
	`groupID="GX078"`
	Name は次のようになります。
	`groupID`

element
または
endelement
	要素の名前。例えば、次のような要素の場合、
	`<s01:Person groupID="GX078">...</s01:Person>`
	Name は次のようになります。
	`s01:Person`

entity	エンティティの名前
startprefixmapping または endprefixmapping	接頭語 (存在する場合)。例えば、次のようなネームスペース宣言の場合、 xmlns:s01="http://www.root.org" Name は次のようになります。 `s01` 別の例では、次のようなネームスペース宣言の場合、 xmlns="http://www.root.org" Name は Null になります。

processinginstruction	処理命令のターゲット。例えば、次のような処理命令の場合、
	`<?xml-stylesheet type="text/css" href="mystyles.css"?>`
	Name は次のようになります。
	`xml-stylesheet`

その他すべてのタイプ	null

NamespaceUri

attribute、element、または endelement のタイプのノードの場合、これは属性または要素が属するネームスペース (存在する場合) となります。その他すべてのタイプのノードでは、このプロパティは Null になります。

NodeType

現在のノードのタイプ。"ドキュメント・ノード" を参照してください。

Path

要素のパス。例えば、以下の XML ドキュメントを考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="mystyles.css"?>
<s01:Root xmlns:s01="http://www.root.org" xmlns="www.default.org">
   <Person>
      <Name>Willeke,Clint B.</Name>
      <DOB>1925-10-01</DOB>
      <GroupID>U3577</GroupID>
      <Address xmlns="www.address.org">
         <City>Newton</City>
         <Zip>56762</Zip>
      </Address>
   </Person>
</s01:Root>
```

City 要素の Path プロパティは /s01:Root/Person/Address/City です。他の要素も同様に処理されます。

ReadState

テキスト・リーダ・オブジェクトの全体の状態を、以下のいずれかで示します。

- "Initial" は、Read() メソッドがまだ呼び出されていないことを示します。
- "Interactive" は、Read() メソッドが少なくとも 1 回呼び出されたことを示します。
- "EndOfFile" は、ファイルの末尾に達したことを示します。

値

ノードのタイプに応じた、現在のノードの値 (存在する場合) です。以下のテーブルに詳細を示します。

テーブル 4-4: タイプごとのノードの値

ノード・タイプ	値と例
attribute	属性の値例えば、次のような属性の場合、 groupID="GX078" Value は次のようになります。 GX078
chars	テキスト・ノードのコンテンツ例えば、次のような要素の場合、 <DOB>1925-10-01</DOB> chars ノードでは、Value は次のようになります。 1925-10-01
comment	コメントのコンテンツ例えば、次のようなコメントの場合、 <!--Comment here--> Value は次のようになります。 Comment here
entity	エンティティの定義

ノード・タイプ	値と例
error	エラー・メッセージ。例は、"検証の実行" を参照してください。
ignorablewhitespace	空白のコンテンツ
processinginstruction	ターゲット以外の処理命令全体のコンテンツ例えば、次のような処理命令の場合、 <?xml-stylesheet type="text/css" href="mystyles.css"?> Value は次のようになります。 type="text/css" href="mystyles.css"?
startprefixmapping	接頭語の後にスペース文字が 1 つ、その後に URI が続きます。例えば、次のようなネームスペース宣言の場合、 xmlns:s01="http://www.root.org" Value は次のようになります。 s01 http://www.root.org
warning	警告メッセージ。例は、"検証の実行" を参照してください。
その他すべてのタイプ (element を含む)	null

seq

ドキュメント内のこのノードのシーケンス番号。最初のノードの番号は 1 です。属性には、それが属する要素と同じシーケンス番号が割り当てられていることに注意してください。

4.4 解析メソッドの引数リスト

ドキュメント・ソースを解析するには、テキスト・リーダの ParseFile()、ParseStream()、ParseString()、または ParseURL() メソッドを使用します。どのような場合でも、ソース・ドキュメントは、適格な XML ドキュメント、つまり、XML 構文の基本的な規約に従ったドキュメントである必要があります。これらのメソッドでは、最初の 2 つの引数のみが必要となります。参考までに、これらのメソッドの引数を以下に順番に示します。

1. Filename、Stream、String、または URL ー ドキュメント・ソース。

   ParseFile() では、Filename 引数には ASCII 文字のみを含む必要があります。

2. TextReader ー テキスト・リーダ・オブジェクト。メソッドが $$$OK を返す場合に、出力パラメータとして返されます。

3. Resolver ー ソースの解析時に使用されるエンティティ・リゾルバ。詳細は、"SAX パーサの使用法のカスタマイズ" の "カスタム・エンティティの解析実行" を参照してください。

4. Flags ー SAX パーサによる検証と処理を制御するフラグまたはフラグの組み合わせ。詳細は、"SAX パーサの使用法のカスタマイズ" の "パーサ・フラグの設定" を参照してください。

5. Mask ー XML ソース内の目的の項目を指定するためのマスク。詳細は、"SAX パーサの使用法のカスタマイズ" の "イベント・マスクの指定" を参照してください。

   Tipヒント %XML.TextReader の解析メソッドでは、既定のマスクは $$$SAXCONTENTEVENTS です。これはコメントを無視することに注意してください。可能性のあるタイプのノードをすべて解析するには、この引数に $$$SAXALLEVENTS を使用します。これらのマクロは、%occSAX.inc インクルード・ファイルで定義します。

6. SchemaSpec ー ドキュメント・ソースの検証の基準となるスキーマ仕様。この引数は、ネームスペース/URL のペアをコンマで区切って指定したリストを含む文字列です。

   ```
   "namespace URL,namespace URL"
   ```

   ここで、namespace はスキーマに使用する XML ネームスペースで、URL はスキーマ・ドキュメントの位置を表す URL です。ネームスペースと URL の値の間は、1 つの空白文字で区切られています。

7. KeepWhiteSpace ー 空白を保持するかどうかを指定するオプション。

8. pHttpRequest ー (ParseURL() メソッドのみ) %Net.HttpRequest のインスタンスとしての、Web サーバへの要求。既定では、システムは %Net.HttpRequest の新規インスタンスを作成してそれを使用しますが、代わりに、%Net.HttpRequest の別のインスタンスで要求を行うことができます。これは、既存の %Net.HttpRequest があり、プロキシおよびその他のプロパティが既に設定されている場合に役立ちます。このオプションは、http のタイプの (file や ftp などではない) URL にのみ適用されます。

   %Net.HttpRequest の詳細は、"インターネット・ユーティリティの使用法" を参照してください。または、%Net.HttpRequest のクラス・ドキュメントを参照してください。

4.5 ドキュメントのナビゲート

ドキュメントをナビゲートするには、テキスト・リーダの Read()、ReadStartElement()、MoveToAttributeIndex()、MoveToAttributeName()、MoveToElement()、MoveToContent()、および Rewind() メソッドを使用します。

4.5.1 次のノードへの移動

ドキュメント内の次のノードに進むには、Read() メソッドを使用します。Read() メソッドは、前方のノードが存在しなくなるまで (つまり、ドキュメントの末尾に達するまで)、True の値を返します。前述の例では、このメソッドを以下のようなループで使用しました。

```
 While (textreader.Read()) {
...
 }
```

4.5.2 特定の要素の最初の出現箇所への移動

ドキュメント内の特定の要素の、最初の出現箇所に移動できます。この操作には、ReadStartElement() メソッドを使用します。このメソッドは、要素が検出されなくなるまで True の値を返します。要素が検出されない場合、メソッドはファイルの末尾に達しています。

ReadStartElement() メソッドは、要素名および (オプションで) ネームスペース URI の 2 つの引数をとります。%XML.TextReader クラスはネームスペースの接頭語を処理しないことに注意してください。これにより、ReadStartElement() メソッドでは以下の 2 つの要素の名前が異なると見なします。

```
<Person>Smith,Ellen W. xmlns="http://www.person.org"</Person>
<s01:Person>Smith,Ellen W. xmlns:s01="http://www.person.org"</s01:Person>
```

4.5.3 属性への移動

要素に移動すると、その要素に属性がある場合は、以下の 2 つの方法のいずれかでその属性に移動できます。

- MoveToAttributeIndex() メソッドを使用して、インデックス (要素内の属性の通常の位置) により特定の属性に移動します。このメソッドの引数は、属性のインデックス番号のみです。AttributeCount プロパティを使用して、指定された要素内の属性の数を知ることができます。すべてのプロパティのリストについては、"ノード・プロパティ" を参照してください。
- MoveToAttributeName() メソッドを使用して、名前により特定の属性に移動します。このメソッドは、属性名および (オプションで) ネームスペース URI の 2 つの引数をとります。%XML.TextReader クラスはネームスペースの接頭語を処理しないことに注意してください。属性に接頭語がある場合、その接頭語は属性名の一部と見なされます。

現在の要素に対する属性を終了したら、Read() などの検索メソッドを実行することで、ドキュメント内の次の要素に移動できます。また、MoveToElement() メソッドを実行して、現在の属性を含む要素に戻ることもできます。

例えば、以下のコードは、インデックス番号によって、現在のノードに対するすべての属性をリストにします。

ObjectScript

```
If (textreader.NodeType = "element") {
    // list attributes for this node
    For a = 1:1:textreader.AttributeCount {
        Do textreader.MoveToAttributeIndex(a)
        Write textreader.LocalName," = ",textreader.Value,!
    }
}
```

以下のコードは、現在のノードに対する color 属性の値を検索します。

ObjectScript

```
If (textreader.NodeType = "element") {
    // find color attribute for this node
    If (textreader.MoveToAttributeName("color")) {
        Write "color = ",textreader.Value,!
    }
}
```

4.5.4 コンテンツを含む次のノードへの移動

MoveToContent() メソッドはコンテンツの検出に役立ちます。具体的には以下のとおりです。

- ノードのタイプが "chars" 以外の場合、このメソッドは "chars" タイプの次のノードに進みます。
- ノードのタイプが "chars" の場合、このメソッドはファイル内で先に進みません。

4.5.5 巻き戻し

ここで説明するメソッドは、Rewind() メソッドを除き、すべてドキュメント内を前進します。Rewind() メソッドは、ドキュメントの最初に移動し、すべてのプロパティをリセットします。

4.6 検証の実行

既定では、ソース・ドキュメントは指定された任意の DTD またはスキーマ・ドキュメントに対して検証されます。ドキュメントに DTD セクションが含まれる場合、そのドキュメントはその DTD に対して検証されます。代わりに、スキーマ・ドキュメントとの照合で検証するには、"解析メソッドの引数リスト" の説明にあるように、ParseFile()、ParseStream()、ParseString()、または ParseURL() の引数リストで目的のスキーマを指定します。

ほとんどのタイプの検証の問題は致命的ではなく、エラーまたは警告のいずれかを発生させます。具体的には、タイプが "error" または "warning" のノードが、ドキュメント・ツリーのエラーが発生した場所に自動的に追加されます。こうしたノードには、他のタイプのノードと同様に移動し、調査できます。

例えば、以下の XML ドキュメントを考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Root [
  <!ELEMENT Root (Person)>
  <!ELEMENT Person (#PCDATA)>
]>
<Root>
   <Person>Smith,Joe C.</Person>
</Root>
```

この場合、検証エラーは予期されません。このトピックで前述したメソッド例 WriteNodes() を思い出してください。そのメソッドを使用してこのドキュメントを読み取る場合、出力は以下のようになります。

```
Node 1 is a(n) element named: Root
    and has no value
Node 2 is a(n) ignorablewhitespace and has no name
    with value:

Node 3 is a(n) element named: Person
    and has no value
Node 4 is a(n) chars and has no name
    with value: Smith,Joe C.
Node 5 is a(n) endelement named: Person
    and has no value
Node 6 is a(n) ignorablewhitespace and has no name
    with value:

Node 7 is a(n) endelement named: Root
    and has no value
```

一方、ファイルは以下のようになるとします。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Root [
  <!ELEMENT Root (Person)>
  <!ELEMENT Person (#PCDATA)>
]>
<Root>
   <Employee>Smith,Joe C.</Employee>
</Root>
```

この場合、<Employee> 要素が DTD セクションで宣言されていないので、エラーが予想されます。ここで、メソッド例 WriteNodes() を使用してこのドキュメントを読み取る場合、出力は以下のようになります。

```
Node 1 is a(n) element named: Root
    and has no value
Node 2 is a(n) ignorablewhitespace and has no name
    with value:

Node 3 is a(n) error and has no name
    with value: Unknown element 'Employee'
while processing c:/TextReader/docwdtd2.txt at line 7 offset 14
Node 4 is a(n) element named: Employee
    and has no value
Node 5 is a(n) chars and has no name
    with value: Smith,Joe C.
Node 6 is a(n) endelement named: Employee
    and has no value
Node 7 is a(n) ignorablewhitespace and has no name
    with value:

Node 8 is a(n) error and has no name
    with value: Element 'Employee' is not valid for content model: '(Person)'
```

```
while processing c:/TextReader/docwdtd2.txt at line 8 offset 8
Node 9 is a(n) endelement named: Root
    and has no value
```

"SAX パーサの使用法のカスタマイズ" の "パーサ・フラグの設定" も参照してください。

4.7 例 : ネームスペースのレポート

次のサンプルのメソッドでは、任意の XML ファイルを読み取り、それぞれの要素および属性が属するネームスペースを示します。

Class Member

```
ClassMethod ShowNamespacesInFile(filename As %String)
{
  Set status = ##class(%XML.TextReader).ParseFile(filename,.textreader)

  //check status
  If $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

  //iterate through document, node by node
  While textreader.Read()
  {
    If (textreader.NodeType = "element")
    {
       Write !,"The element ",textreader.LocalName
       Write " is in the namespace ",textreader.NamespaceUri
       }
    If (textreader.NodeType = "attribute")
    {
       Write !,"The attribute ",textreader.LocalName
       Write " is in the namespace ",textreader.NamespaceUri
       }
     }
}
```

ターミナルで使用すると、このメソッドは以下のような出力を生成します。

```
The element Person is in the namespace www://www.person.com
The element Name is in the namespace www://www.person.com
```

次のバリエーションでは、XML 対応オブジェクトを受け取り、それをストリームに書き込んで、そのストリームを使用して同じタイプのレポートを生成しています。

Class Member

```
ClassMethod ShowNamespacesInObject(obj)
{
  set writer=##class(%XML.Writer).%New()

  set str=##class(%GlobalCharacterStream).%New()
  set status=writer.OutputToStream(str)
  if $$$ISERR(status) {do $System.Status.DisplayError(status) quit ""}

  //write to the stream
  set status=writer.RootObject(obj)
  if $$$ISERR(status) {do $System.Status.DisplayError(status) quit }

  Set status = ##class(%XML.TextReader).ParseStream(str,.textreader)

  //check status
  If $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

  //iterate through document, node by node
  While textreader.Read()
  {
    If (textreader.NodeType = "element")
    {
       Write !,"The element ",textreader.LocalName
       Write " is in the namespace ",textreader.NamespaceUri
```

```
      }
   If (textreader.NodeType = "attribute")
   {
      Write !,"The attribute ",textreader.LocalName
      Write " is in the namespace ",textreader.NamespaceUri
      }
    }
}
```

5
オブジェクトへの XML のインポート

%XML.Reader クラスを使用すると、InterSystems IRIS® データ・プラットフォーム・オブジェクトに XML ドキュメントをインポートできます。

注釈　使用するどの XML ドキュメントの XML 宣言にも、そのドキュメントの文字エンコードを明記する必要があります。これにより、ドキュメントが宣言どおりにエンコードされます。文字エンコードが宣言されていない場合は、"入出力の文字エンコード" で説明されている既定値が使用されます。これらの既定値が正しくない場合は、XML 宣言を修正して、実際に使用されている文字セットを指定するようにします。

任意の XML ドキュメントを読み取って DOM (ドキュメント・オブジェクト・モデル) を返すには、%XML.Reader を使用することもできます。"XML ドキュメントの DOM 表現" を参照してください。

5.1 概要

InterSystems IRIS には、XML ドキュメントを読み取り、そのドキュメントの要素に対応する 1 つまたは複数の XML 対応 InterSystems IRIS オブジェクトを作成するツールがあります。基本的な要件は以下のとおりです。

- そのオブジェクトのクラス定義は、%XML.Adaptor を拡張する必要があります。いくつかの例外はありますが、そのオブジェクトが参照するクラスも、%XML.Adaptor の拡張とします。"オブジェクトの XML への投影" を参照してください。

  Tipヒント　対応する XML スキーマを使用できる場合は、それを使用してクラス (およびサポートするクラス) を生成できます。"XML スキーマからのクラスの生成" を参照してください。

- XML ドキュメントをインポートするには、%XML.Reader のインスタンスを作成してから、そのインスタンスのメソッドを呼び出します。これらのメソッドでは、XML ソース・ドキュメントを指定し、XML 要素を XML 対応クラスに関連付け、ソースからオブジェクトに要素を読み込みます。

%XML.Reader クラスは、%XML.Adaptor によって提供されるメソッドと連携して、以下の処理を行います。

- InterSystems IRIS SAX インタフェースを使用して、受信 XML ドキュメントを解析および検証します。この検証では、DTD 検証または XML スキーマ検証のいずれかを実行できます。
- XML 対応のオブジェクトが XML ドキュメントに含まれる要素と相互に関連するかどうかを判定し、そのドキュメントを読み取ったときにこれらのオブジェクトのメモリ内インスタンスを生成します。

%XML.Reader クラスによって生成されたオブジェクト・インスタンスは、データベースには格納されません。これは、メモリ内のオブジェクトです。データベースにオブジェクトを格納する場合は、%Save() メソッドを呼び出すか (永続オブジェクト)、関連するプロパティ値を永続オブジェクトにコピーし、それを保存します。また、アプリケーションでは、新規のデータ

を挿入するタイミングと、既存のデータを更新するタイミングを決定する必要があります。%XML.Reader では、これらを区別できません。

次のターミナル・セッションは簡単な例を示しています。この例では、XML ファイルを新しいオブジェクトに読み込んで、そのオブジェクトを調べてから保存します。

Terminal

```
GXML>Set reader = ##class(%XML.Reader).%New()

GXML>Set file="c:\sample-input.xml"

GXML>Set status = reader.OpenFile(file)

GXML>Write status
1
GXML>Do reader.Correlate("Person","GXML.Person")

GXML>Do reader.Next(.object,.status)

GXML>Write status
1
GXML>Write object.Name
Worthington,Jeff R.
GXML>Write object.Doctors.Count()
2
GXML>Do object.%Save()
```

この例では、次のサンプル XML ファイルを使用しています。

XML

```
<Person GroupID="90455">
  <Name>Worthington,Jeff R.</Name>
  <DOB>1976-11-03</DOB>
  <Address>
    <City>Elm City</City>
    <Zip>27820</Zip>
  </Address>
  <Doctors>
    <Doctor>
      <Name>Best,Nora A.</Name>
    </Doctor>
    <Doctor>
      <Name>Weaver,Dennis T.</Name>
    </Doctor>
  </Doctors>
</Person>
```

5.2 オブジェクトのインポート

5.2.1 メソッドの全体的な構造

メソッドは、以下の操作の一部またはすべてをこの順序で実行する必要があります。

1. %XML.Reader クラスのインスタンスを作成します。
2. 必要に応じて、このインスタンスの Format プロパティを指定して、インポートするファイルの形式を指定します。"リーダ・プロパティ" を参照してください。

   既定では、XML ファイルはリテラル形式と想定されます。ファイルが SOAP でエンコードされた形式の場合は、そのように明記して、ファイルを正しく読み取れるようにする必要があります。
3. 必要に応じて、このインスタンスのその他のプロパティを設定します。"リーダ・プロパティ" を参照してください。
4. ソースを開きます。そのためには、%XML.Reader の次のメソッドのいずれかを使用します。

- OpenFile() ー ファイルを開きます。
- OpenStream() ー ストリームを開きます。
- OpenString() ー 文字列を開きます。
- OpenURL() ー URL を開きます。

  URL で HTTPS を使用する場合、"HTTPS URL のドキュメントへのアクセス" を参照してください。

いずれの場合も、オプションでメソッドに 2 番目の引数を指定して、Format プロパティの値をオーバーライドすることができます。

5. このファイルの 1 つ以上の XML 要素の名前と、対応する構造を持つ InterSystems IRIS XML 対応クラスとを相互に関連付けます。共有するには、以下の 2 つの方法があります。

   - Correlate() メソッドを使用します。このメソッドには、以下のシグニチャがあります。

     ```
     method Correlate(element As %String,
                      class As %String,
                      namespace As %String)
     ```

     ここで、element は XML 要素名、class は InterSystems IRIS クラス名 (パッケージ付き)、namespace はオプションのネームスペース URI を示します。

     namespace 引数を使用する場合、指定されたネームスペースの中の指定された要素名のみが適合します。namespace 引数を "" と指定すると、Next() メソッドで指定された既定のネームスペースが適合します。namespace 引数を使用しない場合、要素名のみを使用して適合を判断します。

     Tipヒント このトピックの例では関連付けを 1 つしか示していませんが、Correlate() メソッドを繰り返し呼び出して複数の要素を関連付けることができます。

   - CorrelateRoot() メソッドを使用します。このメソッドには、以下のシグニチャがあります。

     ```
     method CorrelateRoot(class As %String)
     ```

     ここで、class は InterSystems IRIS クラス名 (パッケージ付き) です。このメソッドでは、XML ドキュメントのルート要素を指定したクラスに関連付けることを指定します。

6. 次のように、クラス・インスタンスをインスタンス化します。

   - Correlate() を使用した場合、ファイルの関連付けられた要素を、一度に 1 つずつループします。ループ内で、Next() メソッドを使用します。このメソッドには、以下のシグニチャがあります。

     ```
     method Next(ByRef oref As %ObjectHandle,
                 ByRef sc As %Status,
                 namespace As %String = "") as %Integer
     ```

     ここで、oref はメソッドで作成されたオブジェクト、sc はステータス、namespace はファイルの既定のネームスペースを示します。

   - CorrelateRoot() を使用した場合、Next() メソッドを一度呼び出します。これにより、関連付けられたクラスがインスタンス化されます。

   ファイルの最後に達すると、Next() メソッドは 0 を返します。その後 Next() を再度呼び出すと、ファイルの先頭から再度オブジェクトをループします (指定した相互関係は、そのまま有効です)。

5.2.2 エラーのチェック

前述のセクションで説明したメソッドのほとんどではステータスが返されます。メソッドを実行するたびに返されるステータスを確認し、必要に応じて実行を中止します。

5.2.3 基本的なインポートの例

次のような test.xml と呼ばれる XML ファイルがあるとします。

XML

```
<Root>
   <Person>
      <Name>Elvis Presley</Name>
   </Person>
   <Person>
      <Name>Johnny Carson</Name>
   </Person>
</Root>
```

まず、Person のオブジェクト表現である、XML 対応のクラス MyApp.Person を定義します。

Class Definition

```
Class MyApp.Person Extends (%Persistent,%XML.Adaptor)
{
Parameter XMLNAME = "Person";

Property Name As %String;
}
```

このファイルを MyAppPerson クラスのインスタンスにインポートするには、次のメソッドを記述します。

Class Member

```
ClassMethod Import()
{
    // Create an instance of %XML.Reader
    Set reader = ##class(%XML.Reader).%New()

    // Begin processing of the file
    Set status = reader.OpenFile("c:\test.xml")
    If $$$ISERR(status) {do $System.Status.DisplayError(status)}

    // Associate a class name with the XML element name
    Do reader.Correlate("Person","MyApp.Person")

    // Read objects from xml file
    While (reader.Next(.object,.status)) {
        Write object.Name,!
    }

    // If error found during processing, show it
    If $$$ISERR(status) {do $System.Status.DisplayError(status)}

}
```

このメソッドは、以下のタスクを実行します。

- InterSystems IRIS SAX インタフェースを使用して、入力ファイルを解析します。DTD またはスキーマが指定されている場合、DTD またはスキーマに対するドキュメントの認証も行います。
- Correlate() メソッドを使用してクラス MyApp.MyPerson を XML 要素 <Person> に関連付けます。<Person> の各子要素は、MyPerson のプロパティになります。
- 入力ファイルから、<Person> 要素がすべてなくなるまで、1 つずつ読み取ります。
- 最後に、エラーによりループが終了すると、そのエラーが現在の出力デバイスに表示されます。

上記の説明のとおり、この例ではオブジェクトがデータベースに格納されません。MyPerson は永続オブジェクトなので、以下の行を While ループに追加することで、オブジェクトをデータベースに格納できます。

ObjectScript

```
Set savestatus = object.%Save()
If $$$ISERR(savestatus) {do $System.Status.DisplayError(savestatus)}
```

5.2.4 HTTPS URL のドキュメントへのアクセス

OpenURL() メソッドでは、SSL/TLS を必要とする URL にドキュメントがある場合、以下の手順を実行します。

1. 管理ポータルを使用して、必要な接続の詳細を格納する SSL/TLS 構成を作成します。詳細は、インターシステムズの "TLS ガイド" を参照してください。

   これは、1 回限りの手順です。

2. %XML.Reader を使用する場合は、リーダ・インスタンスの SSLConfiguration プロパティを設定します。値については、前の手順で作成した SSL/TLS 構成の名前を指定します。

または、%XML.Reader を使用する場合は、以下の手順も実行します。

1. %Net.HttpRequest のインスタンスを作成します。
2. そのインスタンスの SSLConfiguration プロパティを、管理ポータルで作成した SSL/TLS 構成の名前に設定します。
3. %Net.HttpRequest のそのインスタンスを、OpenURL() の 3 つ目の引数として使用します。

以下はその例です。

ObjectScript

```
set reader=##class(%XML.Reader).%New()
set httprequest=##class(%Net.HttpRequest).%New()
set httprequest.SSLConfiguration="mysslconfigname"
set status=reader.OpenURL("https://myurl",,httprequest)
```

5.2.4.1 サーバに認証が必要な場合のドキュメントへのアクセス

サーバに認証が必要な場合、%Net.HttpRequest のインスタンスを作成し、そのインスタンスの Username プロパティおよび Password プロパティを設定します。また、前述のとおりに SSL を使用します (つまり、SSLConfiguration プロパティも設定します)。次に、前述の例で示しているとおりに、%Net.HttpRequest のインスタンスを OpenURL() の 3 つ目の引数として使用します。

%Net.HttpRequest および認証の詳細は、"HTTP 要求の送信" にある "ログイン資格情報の提供" を参照してください。

5.3 必要な要素および属性のチェック

既定では、Next() メソッドは、Required とマークされたプロパティに対応する要素と属性の有無をチェックしません。該当する要素と属性の有無をリーダがチェックするようにするには、Next() を呼び出す前に、リーダの CheckRequired プロパティを 1 に設定します。互換性の理由から、このプロパティの既定値は 0 です。

CheckRequired を 1 に設定して Next() を呼び出すときに、インポートされた XML の必須の要素または属性が失われている場合、Next() メソッドは、sc 引数をエラー・コードに設定します。以下はその例です。

```
SAMPLES>set next= reader.Next(.object,.status)

SAMPLES>w next
0
SAMPLES>d $system.Status.DisplayError(status)

ERROR #6318: Property required in XML document: ReqProp
```

5.4 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.Adaptor クラスには、そのようなドキュメントをインポートする際の動作を指定するパラメータが用意されています。詳細は、"特殊なトピック" を参照してください。

5.5 空の要素および属性のインポート方法の制御

オブジェクトを XML 対応にする場合、NULL 値および空文字列を XML に投影する方法を指定します ("オブジェクトの XML への投影" を参照)。

その指定方法の 1 つは、XML 対応クラスで XMLIGNORENULL を "RUNTIME" (大文字/小文字の区別なし) に設定することです。この場合、%XML.Reader を使用して XML ファイルを読み取り、InterSystems IRIS オブジェクトを作成すると、InterSystems IRIS は次のようにリーダの IgnoreNull プロパティの値を使用して、空の要素または属性の処理方法を決定します。

- リーダの IgnoreNull プロパティが 0 (既定) の場合、要素または属性が空のとき、対応するプロパティには $char(0) が設定されます。
- リーダの IgnoreNull プロパティが 1 の場合、要素または属性が空のとき、対応するプロパティは設定されず、"" となります。

リーダの IgnoreNull プロパティが有効になるためには、XML 対応クラスで XMLIGNORENULL が "RUNTIME" に設定されている必要があります。XMLIGNORENULL に指定可能な他の値は、0 (既定値)、1、および "INPUTONLY" です。"オブジェクトの XML への投影" を参照してください。

5.5.1 例 : IgnoreNull が 0 (既定)

まず、以下のクラスについて考えてみます。

Class Definition

```
Class EmptyStrings.Import Extends (%Persistent, %XML.Adaptor)
{

Parameter XMLNAME="Test";

///Reader will set IgnoreNull property
Parameter XMLIGNORENULL = "RUNTIME";

Property PropertyA As %String;

Property PropertyB As %String;

Property PropertyC As %String;

Property PropertyD As %String(XMLPROJECTION = "ATTRIBUTE");

Property PropertyE As %String(XMLPROJECTION = "ATTRIBUTE");
}
```

以下の XML ファイルについても考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Test PropertyD="">
  <PropertyA></PropertyA>
  <PropertyB xsi:nil="true"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</Test>
```

%XML.Reader のインスタンスを作成した場合、それを使用してこのファイルを前のクラスにインポートすると、次のような結果が得られます。

- `PropertyA` と `PropertyD` は `$char(0)` になります。
- その他のすべてのプロパティは `""` になります。

例えば、各プロパティを検証しその値を調べて出力を記述するルーチンを記述する場合、次のようになります。

```
PropertyA is $char(0)
PropertyB is null
PropertyC is null
PropertyD is $char(0)
PropertyE is null
```

5.5.2 例 : IgnoreNull が 1

前の例のバリエーションとして、リーダの IgnoreNull プロパティを 1 に設定します。この場合、すべてのプロパティは `""` になります。

例えば、各プロパティを検証しその値を調べて出力を記述するルーチンを記述する場合、次のようになります。

```
PropertyA is null
PropertyB is null
PropertyC is null
PropertyD is null
PropertyE is null
```

5.6 その他の便利なメソッド

場合によっては、%XML.Reader の以下の追加メソッドを使用する必要がある場合があります。

- Rewind() メソッドは、XML ソース・ドキュメントの最初から読み取りを再開する必要がある場合に使用します。このメソッドは、すべての相互関係をクリアします。
- Close() メソッドは、インポート・ハンドラを明示的に閉じる、または削除する必要がある場合に使用します。インポート・ハンドラは自動的に削除されます。このメソッドは、下位互換性のために含まれています。

5.7 リーダ・プロパティ

%XML.Reader の以下のプロパティを設定して、メソッドの全体的な動作を制御できます。

- ドキュメントを解析するときに %XML.Reader のインスタンスがプロセス・プライベート・グローバルを使用するかどうかを指定するには、UsePPGHandler プロパティを使用します。このプロパティが True の場合、インスタンスはプロセス・プライベート・グローバルを使用します。このプロパティが False の場合、インスタンスはメモリを使用します。このプロパティが設定されていない場合 (または空文字列と等しい場合)、インスタンスは既定 (通常はメモリ) を使用します。

  Format プロパティは、XML ドキュメントの全体の形式の指定に使用します。以下の値のいずれかを指定します。

  - 既定値であり、このトピックのほとんどの例で使用されている "literal"。
  - SOAP 1.1 規格で説明されている方法でエンコードされている "encoded"。
  - SOAP 1.2 規格で説明されている方法でエンコードされている "encoded12"。

  OpenFile()、OpenStream()、OpenString()、OpenURL() の各メソッド内ではこの Format プロパティをオーバーライドできることに注意してください。

  Correlate() と Next() を使用していない場合、このプロパティには何の効果もなくなります。
- Summary プロパティは、リーダに XML 対応オブジェクトの要約フィールドのみのインポートを強制するために使用します。"オブジェクトの XML への投影" で説明したように、オブジェクトの概要は XMLSUMMARY クラス・パラメータによって指定され、コンマ区切りのプロパティのリストとして指定されます。
- IgnoreSAXWarnings プロパティを使用して、リーダがSAX パーサにより発行される警告を報告すべきかを指定します。

%XML.Reader は、参照中のドキュメントを検証するのに使用できるプロパティも提供します。

- Document プロパティには、読み取り対象の解析されたドキュメント全体を表現する %XML.Document のインスタンスが含まれています。%XML.Document の詳細は、クラス・リファレンスを参照してください。
- Node プロパティは、XML ドキュメントの現在のノードを表す文字列です。0 はドキュメント、つまり、ルート要素の親を表すことに注意してください。

EntityResolver、SAXFlags、および SAXSchemaSpec の各プロパティの詳細は、"SAX パーサの使用法のカスタマイズ" を参照してください。SAXMask プロパティが無視されることに注意してください。

5.8 相互に関連付けられたオブジェクトをリーダが処理する方法の再定義

%XML.Reader のインスタンスが XML 対応クラスと相互に関連付けられた XML 要素を検出すると、リーダはそのクラスの XMLNew() メソッドを呼び出し、次にこのメソッドは既定では %New() を呼び出します。つまり、リーダが相互に関連付けられた要素を検出すると、リーダは相互に関連付けられたクラスの新しいオブジェクトを作成します。この新しいオブジェクトには、XML ドキュメントから読み取ったデータが格納されます。

この動作をカスタマイズするには、XML 対応クラス (または場合によっては独自のカスタム XML アダプタ) 内の XMLNew() を再定義します。例えば、このメソッドは代わりにそのクラスの既存のインスタンスを開くことができます。次にこの既存のインスタンスは、XML ドキュメントから読み取ったデータを受け取ります。

以下の例では、XML ドキュメントから読み取った新しいデータを使用して既存のインスタンスを更新するように XMLNew() を変更する方法を示しています。

どちらの例でも、簡易化のために、XML ドキュメント内のノードには、そのクラスのエクステント内の ID と比較できる ID が含まれていると想定しています。当然ながら、他の方法で XML ドキュメントを既存のオブジェクトと比較することもできます。

5.8.1 %XML.Reader が XMLNew() を呼び出すとき

参考までに、%XML.Reader は XMLNew() メソッドを以下の 2 つの場合に自動的に呼び出します。

- (外部ドキュメントの) XML 要素を XML 対応クラスに関連付けた後で %XML.Reader の Next() メソッドを呼び出すと、%XML.Reader は XMLNew() を呼び出します。Next() メソッドはドキュメントから次の要素を取得し、XMLNew() を呼び出して適切なオブジェクトのインスタンスを作成した後で、要素をオブジェクトにインポートします。
- 同様に、%XML.Reader は関連付けられた XML 要素の任意のオブジェクト値プロパティの XMLNew() を呼び出します。

5.8.2 例 1 : XML 対応クラス内の XMLNew() の変更

まず、以下の XML ファイルについて考えてみましょう。

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <Person>
    <IRISID>4</IRISID>
    <Name>Quine,Maria K.</Name>
    <DOB>1964-11-14</DOB>
    <Address>
      <City>Hialeah</City>
      <Zip>94999</Zip>
    </Address>
    <Doctors>
      <Doctor>
        <Name>Vanzetti,Debra B.</Name>
      </Doctor>
    </Doctors>
  </Person>
...
```

このファイルは、次の InterSystems IRIS クラスにマッピングします (一部のみ表示)。

Class Definition

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

```
Parameter XMLNAME="Person";

/// make sure this is the same as the XMLNAME of the property
/// in this class that is of type %XML.Id
Parameter NAMEOFEXPORTID As %String = "IRISID";

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

Property Name As %Name;

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

Property Address As GXML.Address;

Property Doctors As list Of GXML.Doctor;

}
```

このクラスでは、IdForExport プロパティの目的は、このクラスのオブジェクトがエクスポートされるときに InterSystems IRIS 内部 ID を要素 (IRISID) に投影することです (この例では、これによりインポート用の適切なファイルを簡単に生成できます。お使いのクラスにはこのようなプロパティが含まれている必要はありません)。

NAMEOFEXPORTID パラメータを使用して、このクラスのオブジェクトをエクスポートする際に InterSystems IRIS ID 用に使用する要素を指定します。これが含まれている理由は、このクラスに追加したカスタマイズされた XMLNew() メソッドの利便性のためだけです。このメソッドの定義は以下のとおりです。

Class Member

```
ClassMethod XMLNew(doc As %XML.Document, node As %Integer,
contOref As %RegisteredObject = "") As GXML.PersonWithXMLNew
{
    Set id=""
    Set tmpnode=doc.GetNode(node)
    Do tmpnode.MoveToFirstChild()
    Do {
        //compare data node to the string given by the NAMEOFEXPORTID parameter
        //which indicates the XMLNAME of the ids for this object
        If tmpnode.NodeData=..#NAMEOFEXPORTID {
            //get the text from this node; this corresponds to an id in the database
            Do tmpnode.GetText(.id)}
        } While tmpnode.MoveToNextSibling()

    //if there is no id in the given node, create a new object
    If id="" {
        Write !, "Creating a new object..."
        Quit ..%New()}

    //open the given object
    Set result=..%OpenId(id)

    //if the id doesn't correspond to an existing object, create a new object
    If result=$$$NULLOREF {
        Write !, "Creating a new object..."
        Quit ..%New()}

    Write !, "Updating an existing object..."

    Quit result
}
```

%XML.Reader は、XML ドキュメントを読み取ってノードを GXML.PersonWithXMLNew と相互に関連付ける際に、このメソッドを呼び出します。このメソッドは、このクラス内の NAMEOFEXPORTID パラメータの値 (IRISID) を参照します。次にこのメソッドは、IRISID という要素を持つドキュメント内のノードを調べて、その値を取得します。

この ID がこのクラスの既存オブジェクトに対応している場合は、このメソッドはそのインスタンスを開きます。そうでない場合は、このメソッドはこのクラスの新しいインスタンスを開きます。どちらの場合でも、開かれたインスタンスは XML ドキュメントで指定されたプロパティを受け取ります。

最後に、次のユーティリティ・クラスには、XML ファイルを開いて %XML.Reader を呼び出すメソッドが含まれています。

Class Definition

```
Class GXML.DemoXMLNew Extends %RegisteredObject
{

ClassMethod ReadFile(filename As %String = "c:\temp\sample.xml")
{
    Set reader=##class(%XML.Reader).%New()
    Set sc=reader.OpenFile(filename)
    If $$$ISERR(sc) {Do $system.OBJ.DisplayError(sc) Quit  }

    Do reader.Correlate("Person","GXML.PersonWithXMLNew")

    //loop through elements in file
    While reader.Next(.person,.sc) {
        Write !,person.Name,!
        Set sc=person.%Save()
        If $$$ISERR(sc) {Do $system.OBJ.DisplayError(sc) Quit  }
    }

    Quit
}

}
```

上記のメソッドを実行すると、ファイル内の <Person> 要素ごとに以下のいずれかの操作が実行されます。

- 既存のオブジェクトが開かれて、ファイル内の詳細情報によって更新されてから保存されます。
- ファイル内の詳細情報が格納された新しいオブジェクトが作成されます。

以下はその例です。

```
GXML>d ##class(GXML.DemoXMLNew).ReadFile()

Updating an existing object...
Zampitello,Howard I.

Creating a new object...
Smyth,Linda D.

Creating a new object...
Vanzetti,Howard I.
```

5.8.3 例 2：カスタム XML アダプタ内の XMLNew() の変更

2 つ目の例では、1 つ目の例と同じ操作を実行するカスタム XML アダプタを作成します。このアダプタ・クラスは以下のとおりです。

Class Definition

```
Class GXML.AdaptorWithXMLNew Extends %XML.Adaptor
{

/// make sure this is the same as the XMLNAME of the property in this class
/// that is of type %XML.Id
Parameter NAMEOFEXPORTID As %String = "IRISID";

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

ClassMethod XMLNew(document As %XML.Document, node As %Integer,
containerOref As %RegisteredObject = "") As %RegisteredObject
[CodeMode=objectgenerator,GenerateAfter=%XMLGenerate,ServerOnly=1]
{
 If %compiledclass.Name'="GXML.AdaptorWithXMLNew" {
  Do %code.WriteLine(" Set id="""")")
  Do %code.WriteLine(" Set tmpnode=document.GetNode(node)")
  Do %code.WriteLine(" Do tmpnode.MoveToFirstChild()")
  Do %code.WriteLine(" Do {")
  Do %code.WriteLine("     If tmpnode.NodeData=..#NAMEOFEXPORTID ")
  Do %code.WriteLine("       {Do tmpnode.GetText(.id)}")
  Do %code.WriteLine(" } While tmpnode.MoveToNextSibling() ")
  Do %code.WriteLine(" If id="""" {")
```

```
  Do %code.WriteLine("  Write !,""Creating new object...""")
  Do %code.WriteLine("  Quit ##class(""_%class.Name_").%New()}")
  Do %code.WriteLine(" set result=##class(""_%class.Name_").%OpenId(id)")
  Do %code.WriteLine(" If result=$$$NULLOREF {")
  Do %code.WriteLine("     Write !,""Creating new object...""")
  Do %code.WriteLine("     Quit ##class(""_%class.Name_").%New() }")
  Do %code.WriteLine(" Write !,""Updating existing object ...""")
  Do %code.WriteLine(" Quit result")

  }

 QUIT $$$OK
}

}
```

IdForExport プロパティと NAMEOFEXPORTID パラメータによって、サブクラスのオブジェクトがエクスポートされるときに InterSystems IRIS 内部 ID を要素に投影する方法の規約が決定されます。その趣旨は、サブクラス内の IdForExport を再定義する場合は、それに応じて NAMEOFEXPORTID を再定義するというものです。

このクラスでは、XMLNew() メソッドはメソッド・ジェネレータです。このクラス (または任意のサブクラス) がコンパイルされると、InterSystems IRIS はここで示しているコードをこのメソッドの本体に書き込みます。"クラスの定義と使用" の "メソッド・ジェネレータ" を参照してください。

以下のクラスは、カスタム・アダプタを拡張します。

Class Definition

```
Class GXML.PersonWithXMLNew2
Extends (%Persistent, %Populate, GXML.AdaptorWithXMLNew)
{

Parameter XMLNAME = "Person";

Property Name As %Name;

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

Property Address As GXML.Address;

Property Doctors As list Of GXML.Doctor;

}
```

前出のサンプル ReadFile メソッドを実行すると、ファイル内の <Person> 要素ごとに、このメソッドは新しいレコードを作成して保存するか、既存のレコードを開いて更新します。

5.9 その他の例

ここでは別の例をいくつか挙げます。

5.9.1 柔軟なリーダ・クラス

以下の例は、ファイル名、ディレクトリ、クラス、および要素を入力引数として受け入れる、より柔軟なリーダ・メソッドを示します。このメソッドは、読み取る各オブジェクトを保存します。

Class Definition

```
Class Readers.BasicReader Extends %RegisteredObject
{

ClassMethod Read(mydir, myfile, class, element)
{
   set reader=##class(%XML.Reader).%New()
   if $extract(mydir,$length(mydir))'="/" {set mydir=mydir_"/"}
   set file=mydir_myfile
```

```
   set status=reader.OpenFile(file)
   if $$$ISERR(status) {do $System.Status.DisplayError(status)}

   do reader.Correlate(element,class)

   while reader.Next(.object,.status)
   {
      if $$$ISERR(status) {do $System.Status.DisplayError(status)}
      set status=object.%Save()
      if $$$ISERR(status) {do $System.Status.DisplayError(status)}
      }
}

}
```

Person オブジェクトで読み取る場合、対応する Address オブジェクトでも自動的に読み取りが行われることに注意してください (Person オブジェクトを保存すると、対応する Address オブジェクトも自動的に保存されます)。これはかなり粗野な方法なので、データの一括ロードにのみ適しています。この方法では、既存のデータとの比較や既存のデータの更新は行われません。

このメソッドを使用するには、読み取る XML ドキュメントとプロジェクションが一致する XML 対応クラスが必要です。MyApp.PersonWithAddress および MyApp.Address という名前の XML 対応クラスがあるとします。また、以下の XML ドキュメントがあるとします。

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>
   </Person>
</Root>
```

このファイルのオブジェクトを読み取り、それをディスクに保存するには、以下のような操作を行います。

ObjectScript

```
 set dir="C:\XMLread-these"
 set file="PersonData.txt"
 set cls="MyApp.PersonWithAddress"
 set element="Person"
 do ##class(Readers.BasicReader).Read(dir,file,cls,element)
```

5.9.2 文字列の読み取り

以下のメソッドは、XML 文字列、クラス、および要素を入力引数として受け入れます。このメソッドは、読み取る各オブジェクトを保存します。

Class Definition

```
Class Readers.BasicReader Extends %RegisteredObject
{

ClassMethod ReadString(string, class, element)
{
   set reader=##class(%XML.Reader).%New()
   set status=reader.OpenString(string)
   if $$$ISERR(status) {do $System.Status.DisplayError(status)}

   do reader.Correlate(element,class)

   while reader.Next(.object,.status)
   {
      if $$$ISERR(status) {do $System.Status.DisplayError(status)}
      set status=object.%Save()
      if $$$ISERR(status) {do $System.Status.DisplayError(status)}
      }
}

}
```

このメソッドを使用するには、以下のような操作を行います。

ObjectScript

```
 set cls="MyApp.Person"
 set element="Person"
 do ##class(Readers.BasicReader).ReadString(string,cls,element)
```

6

XML ドキュメントの DOM 表現

%XML.Document クラスと %XML.Node クラスを使用して、任意の XML ドキュメントを DOM (ドキュメント・オブジェクト・モデル) として表現できます。その後、このオブジェクトをナビゲートして変更できます。新しい DOM を作成してオブジェクトに追加することもできます。

注釈　使用する任意の XML ドキュメントの XML 宣言にはそのドキュメントの文字エンコードを明記する必要があり、明記しておけば、ドキュメントは宣言どおりにエンコードされるようになります。文字エンコードが宣言されていない場合、InterSystems IRIS® データ・プラットフォームでは、″入出力の文字エンコード″ で説明されている既定値が使用されます。これらの既定値が正しくない場合は、XML 宣言を修正して、実際に使用されている文字セットを指定するようにします。

6.1 DOM として XML ドキュメントを開く

既存の XML ドキュメントを DOM として使用するために開くには、以下の手順を実行します。

1. %XML.Reader クラスのインスタンスを作成します。
2. 必要に応じて、このインスタンスの Format プロパティを指定して、インポートするファイルの形式を指定します。

   既定では、XML ファイルはリテラル形式と想定されます。ファイルが SOAP でエンコードされた形式の場合は、そのように明記して、ファイルを正しく読み取れるようにする必要があります。

   ″オブジェクトへの XML のインポート″ の ″リーダ・プロパティ″ を参照してください。

   Correlate() と Next() を使用していない場合、このプロパティには何の効果もなくなります。
3. ソースを開きます。そのためには、%XML.Reader の次のメソッドのいずれかを使用します。
   - OpenFile() ー ファイルを開きます。
   - OpenStream() ー ストリームを開きます。
   - OpenString() ー 文字列を開きます。
   - OpenURL() ー URL を開きます。

   いずれの場合も、オプションでメソッドに 2 番目の引数を指定して、Format プロパティの値をオーバーライドすることができます。
4. DOM である Document プロパティにアクセスします。このプロパティは %XML.Document のインスタンスであり、ドキュメント全体についての情報を見つけるのに使用できるメソッドを提供します。例えば、CountNamespace() は DOM で使用されるネームスペースの総数を返します。

または、XML ドキュメントを格納したストリームがある場合は、%XML.Document の GetDocumentFromStream() メソッドを呼び出します。これにより %XML.Document のインスタンスが返されます。

6.1.1 例 1：ファイルの DOM への変換

例えば、以下のメソッドで XML ファイルを読み取り、そのドキュメントを表す %XML.Document のインスタンスを返します。

```
ClassMethod GetXMLDocFromFile(file) As %XML.Document
{
    set reader=##class(%XML.Reader).%New()

    set status=reader.OpenFile(file)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}

    set document=reader.Document
    quit document
}
```

6.1.2 例 2：オブジェクトの DOM への変換

以下のメソッドで OREF を受け入れ、そのオブジェクトを表す %XML.Document のインスタンスを返します。このメソッドでは、OREF は XML 対応のクラスのインスタンスと想定されます。

Class Member

```
ClassMethod GetXMLDoc(object) As %XML.Document
{
    //make sure this is an instance of an XML-enabled class
    if '$IsObject(object){
        write "Argument is not an object"
        quit $$$NULLOREF
        }
    set classname=$CLASSNAME(object)
    set isxml=$CLASSMETHOD(classname,"%Extends","%XML.Adaptor")
    if 'isxml {
        write "Argument is not an instance of an XML-enabled class"
        quit $$$NULLOREF
        }

    //step 1 - write object as XML to a stream
    set writer=##class(%XML.Writer).%New()
    set stream=##class(%GlobalCharacterStream).%New()
    set status=writer.OutputToStream(stream)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}
    set status=writer.RootObject(object)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}

    //step 2 - extract the %XML.Document from the stream
    set status=##class(%XML.Document).GetDocumentFromStream(stream,.document)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}

   quit document
}
```

6.2 DOM のネームスペースの取得

InterSystems IRIS で XML ドキュメントを読み取り、DOM を作成するとき、ドキュメント内で使用されるすべてのネームスペースが識別され、それぞれにインデックス番号が割り当てられます。

%XML.Document クラスのインスタンスは、ドキュメント内のネームスペースについての情報を見つけるために使用できる以下のメソッドを提供します。

CountNamespace()

ドキュメント内のネームスペースの数を返します。

FindNamespace()

指定されたネームスペースに対応するインデックスを返します。

GetNamespace()

指定されたインデックスの XML ネームスペース URI を返します。

以下の例のメソッドは、ドキュメント内で使用されているネームスペースを示すレポートを表示します。

Class Member

```
ClassMethod ShowNamespaces(doc As %XML.Document)
{
    Set count=doc.CountNamespace()
    Write !, "Number of namespaces in document: "_count
    For i=1:1:count {
        Write !, "Namespace "_i_" is "_doc.GetNamespace(i)
}

}
```

"現在のノードについての情報の取得" も参照してください。

6.3 DOM のノードのナビゲート

ドキュメントのノードにアクセスするには、以下の 2 つの手法を使用できます。

- %XML.Document のインスタンスの GetNode() メソッドを使用します。このメソッドは整数を受け入れます。この整数はノード番号を示すもので、1 から始まっています。
- %XML.Document のインスタンスの GetDocumentElement() メソッドを呼び出します。

  このメソッドは、%XML.Node のインスタンスを返します。これにより、ルート・ノードについての情報にアクセスし、他のノードに移動するのに使用するプロパティとメソッドが提供されます。以下のサブセクションには、%XML.Node の操作に関する詳細が記述されています。

6.3.1 子ノードまたは兄弟ノードへの移動

子ノードまたは兄弟ノードに移動するには、%XML.Node のインスタンスの以下のメソッドを使用します。

- MoveToFirstChild()
- MoveToLastChild()
- MoveToNextSibling()
- MoveToPreviousSibling()

これらの各メソッドは、(メソッド名で示されるように) 別のノードへの移動を試みます。移動が可能な場合、メソッドは True を返します。可能でない場合、メソッドは False を返し、フォーカスはメソッドが呼び出される前と同じ状態です。

これらの各メソッドには、オプションの引数 skipWhitespace があります。この引数が True の場合、メソッドは空白をすべて無視します。skipWhitespace の既定値は False です。

6.3.2 親ノードへの移動

現在のノードの親ノードに移動するには、%XML.Node のインスタンスの MoveToParent() メソッドを使用します。

このメソッドはオプションの引数 restrictDocumentNode をとります。この引数が True の場合、メソッドはドキュメント・ノード (ルート) に移動しません。restrictDocumentNode の既定値は False です。

6.3.3 特定のノードへの移動

特定のノードに移動するために、%XML.Node のインスタンスの NodeId プロパティを設定できます。以下はその例です。

ObjectScript

```
set saveNode = node.NodeId
//..... lots of processing
//...
// restore position
set node.NodeId=saveNode
```

6.3.4 id 属性の使用

場合によっては、XML ドキュメントには id という名前の属性が含まれることがあります。これはドキュメント内の異なるノードを識別するために使用されます。以下はその例です。

XML

```
<?xml version="1.0"?>
<team>
<member id="alpha">Jack O'Neill</member>
<member id="beta">Samantha Carter</member>
<member id="gamma">Daniel Jackson</member>
</team>
```

この例のようにドキュメントが id という属性を使用している場合、この属性を使用してそのノードに移動できます。このためには、そのノードの位置にある %XML.Node のインスタンスを返す、ドキュメントの GetNodeById() メソッドを使用します。(他のほとんどの検索メソッドとは異なり、このメソッドは %XML.Node ではなく %XML.Document から利用できることに注意してください。)

6.4 DOM ノード・タイプ

%XML.Document クラスと %XML.Node クラスは、以下の DOM ノード・タイプを認識します。

- 要素 ($$$xmlELEMENTNODE)

  これらのマクロは、%xml.DOM.inc インクルード・ファイルで定義します。
- テキスト ($$$xmlTEXTNODE)
- 空白 ($$$xmlWHITESPACENODE)

その他のタイプの DOM ノードは無視されます。

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"?>
<team>
<member id="alpha">Jack O'Neill</member>
<member id="beta">Samantha Carter</member>
<member id="gamma">Daniel Jackson</member>
</team>
```

DOM として表示する際、このドキュメントには以下のノードが含まれます。

テーブル 6-1: ドキュメント・ノードの例

NodeID	NodeType	LocalName	メモ
0,29	$$$xmlELEMENTNODE	team	
1,29	$$$xmlWHITESPACENODE		このノードは、<team> ノードの子です。
1,23	$$$xmlELEMENTNODE	member	このノードは、<team> ノードの子です。
2,45	$$$xmlTEXTNODE	Jack O'Neill	このノードは、最初の <member> ノードの子です。
1,37	$$$xmlWHITESPACENODE		このノードは、<team> ノードの子です。
1,41	$$$xmlELEMENTNODE	member	このノードは、<team> ノードの子です。
3,45	$$$xmlTEXTNODE	Samantha Carter	このノードは、2 番目の <member> ノードの子です。
1,45	$$$xmlWHITESPACENODE		このノードは、<team> ノードの子です。
1,49	$$$xmlELEMENTNODE	member	このノードは、<team> ノードの子です。
4,45	$$$xmlTEXTNODE	Daniel Jackson	このノードは、3 番目の <member> ノードの子です。
1,53	$$$xmlWHITESPACENODE		このノードは、<team> ノードの子です。

ノード・タイプやローカル名などの詳細へのアクセスについては、次のセクションを参照してください。

6.5 現在のノードについての情報の取得

%XML.Node の以下の文字列プロパティは、現在のノードについての情報を提供します。いずれの場合にも、現在のノードがないとエラーがスローされます。

LocalName

現在の要素ノードのローカル名。別のタイプのノードでこのプロパティにアクセスしようとすると、エラーがスローされます。

Namespace

現在の要素ノードのネームスペース URI。別のタイプのノードでこのプロパティにアクセスしようとすると、エラーがスローされます。

NamespaceIndex

現在の要素ノードのネームスペースのインデックス。

InterSystems IRIS で XML ドキュメントを読み取り、DOM を作成するとき、ドキュメント内で使用されるすべてのネームスペースが識別され、それぞれにインデックス番号が割り当てられます。

別のタイプのノードでこのプロパティにアクセスしようとすると、エラーがスローされます。

Nil

この要素ノードに対して xsi:nil または xsi:null が True または 1 の場合は True になります。それ以外の場合、このプロパティは False になります。

NodeData

文字ノードの値。

NodeId

現在のノードの ID。別のノードに移動するために、このプロパティを設定できます。

NodeType

前のセクションで説明している、現在のノードのタイプ。

QName

要素ノードの Qname。接頭語がドキュメントで有効な場合に、XML としての出力のみに使用されます。

以下のメソッドは、現在のノードについての追加情報を提供します。

GetText()

```
method GetText(ByRef text) as %Boolean
```

要素ノードのテキスト・コンテンツを取得します。テキストが返される場合、このメソッドは True を返します。この場合、実際のテキストは参照渡しの最初の引数に追加されます。

HasChildNodes()

```
method HasChildNodes(skipWhitespace As %Boolean = 0) as %Boolean
```

現在のノードに子ノードがある場合に True を返します。それ以外の場合は False を返します。

GetNumberAttributes()

```
method GetNumberAttributes() as %Integer
```

現在の要素の属性の番号を返します。

6.5.1 例

以下の例のメソッドは、現在のノードについての情報を提供するレポートを記述します。

Class Member

```
ClassMethod ShowNode(node as %XML.Node)
{
    Write !,"LocalName="_node.LocalName
    If node.NodeType=$$$xmlELEMENTNODE  {
        Write !,"Namespace="_node.Namespace
    }
    If node.NodeType=$$$xmlELEMENTNODE {
        Write !,"NamespaceIndex="_node.NamespaceIndex
     }
    Write !,"Nil="_node.Nil
    Write !,"NodeData="_node.NodeData
    Write !,"NodeId="_node.NodeId
    Write !,"NodeType="_node.NodeType
    Write !,"QName="_node.QName
    Write !,"HasChildNodes returns "_node.HasChildNodes()
    Write !,"GetNumberAttributes returns "_node.GetNumberAttributes()
```

```
    Set status=node.GetText(.text)
    If status {
        Write !, "Text of the node is "_text
        } else {
            Write !, "GetText does not return text"
        }
}
```

出力例は以下のようになります。

```
LocalName=staff
Namespace=
NamespaceIndex=
Nil=0
NodeData=staff
NodeId=1
NodeType=e
QName=staff
HasChildNodes returns 1
GetNumberAttributes returns 5
GetText does not return text
```

6.6 属性の検証のための基本的なメソッド

%XML.Node の以下のメソッドを使用して、現在のノードの属性を検証できます。"属性の検証のためのその他のメソッド"も参照してください。

- AttributeDefined() ー 現在の要素が指定した名前の属性を持つ場合、ゼロ以外 (True) を返します。
- FirstAttributeName() ー 現在の要素の最初の属性の属性名を返します。
- GetAttributeValue() ー 指定した属性の値を返します。要素に属性がない場合、メソッドは Null を返します。
- GetNumberAttributes() ー 現在の要素の属性の番号を返します。
- LastAttributeName() ー 現在の要素の最後の属性の属性名を返します。
- NextAttributeName() ー 属性名を指定すると、指定した属性が有効かどうかにかかわらず、このメソッドは照合順の次の属性の名前を返します。
- PreviousAttributeName() ー 属性名を指定すると、指定した属性が有効かどうかにかかわらず、このメソッドは照合順の前の属性の名前を返します。

以下の例は、指定したノードの属性を参照し、簡単なレポートを記述します。

Class Member

```
ClassMethod ShowAttributes(node as %XML.Node)
{
    Set count=node.GetNumberAttributes()
    Write !, "Number of attributes: ", count
    Set first=node.FirstAttributeName()
    Write !, "First attribute is: ", first
    Write !, "   Its value is: ",node.GetAttributeValue(first)
    Set next=node.NextAttributeName(first)

    For i=1:1:count-2 {
        Write !, "Next attribute is: ", next
        Write !, "   Its value is: ",node.GetAttributeValue(next)
        Set next=node.NextAttributeName(next)
        }
    Set last=node.LastAttributeName()
    Write !, "Last attribute is: ", last
    Write !, "   Its value is: ",node.GetAttributeValue(last)
}
```

以下の XML ドキュメントの例について考えてみます。

XML

```
<?xml version="1.0"?>
<staff attr1="first" attr2="second" attr3="third" attr4="fourth" attr5="fifth">
  <doc>
    <name>David Marston</name>
  </doc>
</staff>
```

このドキュメントの最初のノードをメソッド例に渡すと、以下の出力が表示されます。

```
Number of attributes: 5
First attribute is: attr1
   Its value is: first
Next attribute is: attr2
   Its value is: second
Next attribute is: attr3
   Its value is: third
Next attribute is: attr4
   Its value is: fourth
Last attribute is: attr5
   Its value is: fifth
```

6.7 属性の検証のためのその他のメソッド

このセクションでは、属性の名前、値、ネームスペース、QName、および値ネームスペースを取得するのに使用できるメソッドについて説明します。これらのメソッドは以下のとおりに分類されます。

- 属性名のみを使用するメソッド
- 属性名とネームスペースを使用するメソッド

注釈　XML 規格では、1 つの要素には、それぞれ別のネームスペースにある同じ名前を持つ、複数の属性を含むことができます。ただし、InterSystems IRIS XML では、これはサポートされていません。

"属性の検証のための基本的なメソッド" も参照してください。

6.7.1 属性名のみを使用するメソッド

属性についての情報を取得するには以下のメソッドを使用します。

GetAttribute()

```
method GetAttribute(attributeName As %String,
                    ByRef namespace As %String,
                    ByRef value As %String,
                    ByRef valueNamespace As %String)
```

指定した属性のデータを返します。このメソッドは、参照により以下の値を返します。

- namespace は、属性の QName からのネームスペース URI です。
- value は属性値です。
- valueNamespace は、その値が属するネームスペース URI です。例えば、以下の属性を考えてみます。

  ```
  xsi:type="s:string"
  ```

  この属性の値は `string` であり、この値は、別の場所で宣言され、接頭語を `s` とするネームスペースの中にあります。このドキュメントの既述部分に、以下のネームスペース宣言があったとします。

  ```
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  ```

この場合、valueNamespace は "http://www.w3.org/2001/XMLSchema" となります。

GetAttributeNamespace()

```
method GetAttributeNamespace(attributeName As %String) as %String
```

現在の要素の attributeName という名前の属性の QName から、ネームスペース URI を返します。

GetAttributeQName()

```
method GetAttributeQName(attributeName As %String) as %String
```

指定した属性の QName を返します。

GetAttributeValue()

```
method GetAttributeValue(attributeName As %String) as %String
```

指定した属性の値を返します。

GetAttributeValueNamespace()

```
method GetAttributeValueNamespace(attributeName As %String) as %String
```

指定した属性の値のネームスペースを返します。

6.7.2 属性名とネームスペースを使用するメソッド

名前とネームスペースの両方を使用して属性についての情報を取得するには、以下のメソッドを使用します。

GetAttributeNS()

```
method GetAttributeNS(attributeName As %String,
                      namespace As %String,
                      ByRef value As %String,
                      ByRef valueNamespace As %String)
```

指定した属性のデータを返します。attributeName および namespace で目的の属性を指定します。このメソッドは、参照により以下のデータを返します。

- value は属性値です。
- valueNamespace は、その値が属するネームスペース URI です。例えば、以下の属性を考えてみます。

  ```
  xsi:type="s:string"
  ```

  この属性の値は string であり、この値は、別の場所で宣言され、接頭語を s とするネームスペースの中にあります。このドキュメントの既述部分に、以下のネームスペース宣言があったとします。

  ```
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  ```

  この場合、valueNamespace は "http://www.w3.org/2001/XMLSchema" となります。

GetAttributeQNameNS()

```
method GetAttributeQNameNS(attributeName As %String,
                           namespace As %String)
                           as %String
```

指定した属性の QName を返します。attributeName および namespace で目的の属性を指定します。

GetAttributeValueNS()

```
method GetAttributeValueNS(attributeName As %String,
                           namespace As %String)
                           as %String
```

指定した属性の値を返します。attributeName および namespace で目的の属性を指定します。

GetAttributeValueNamespaceNS

```
method GetAttributeValueNamespaceNS(attributeName As %String,
                                    namespace As %String)
                                    as %String
```

指定した属性の値のネームスペースを返します。attributeName および namespace で目的の属性を指定します。

6.8 DOM の作成または編集

DOM を作成するか既存の DOM を変更するには、%XML.Document の以下のメソッドを使用します。

CreateDocument()

```
classmethod CreateDocument(localName As %String,
                           namespace As %String)
                           as %XML.Document
```

ルート要素のみから構成される %XML.Document の新しいインスタンスを返します。

AppendCharacter()

```
method AppendCharacter(text As %String)
```

新しい文字データ・ノードをこの要素ノードの子のリストに追加します。現在のノードのポインタは変更されません。このノードは追加された子の親のままです。

AppendElement()

```
method AppendElement(localName As %String,
                     namespace As %String,
                     text As %String)
```

新しい要素ノードをこのノードの子のリストに追加します。テキストの引数が指定されると、文字データが新しい要素の子として追加されます。現在のノードのポインタは変更されません。このノードは追加された子の親のままです。

AppendNode()

```
method AppendNode(sourceNode As %XML.Node) as %Status
```

このノードの子として、指定したノードのコピーを追加します。コピーするノードは、任意のドキュメントをコピー元にできます。現在のノードのポインタは変更されません。このノードは追加された子の親のままです。

AppendTree()

```
method AppendTree(sourceNode As %XML.Node) as %Status
```

このノードの子として、指定したノードの (すべての子を含む) コピーを追加します。コピーするツリーは任意のドキュメントをコピー元にできますが、ソース・ノードの下位ノードをこのノードにすることはできません。現在のノードのポインタは変更されません。このノードは追加された子の親のままです。

InsertNamespace()

```
method InsertNamespace(namespace As %String)
```

指定されたネームスペース URI をドキュメントに追加します。

InsertCharacter()

```
method InsertCharacter(text as %String, ByRef child As %String) as %String
```

このノードの子として、新しい文字データ・ノードを挿入します。新しい文字データは、指定した子ノードの直前に挿入されます。child 引数は、子ノードのノード ID です。これは参照によって渡されるため、挿入後に更新される可能性があります。挿入されたノードの NodeId が返されます。現在のノードのポインタは変更されません。

InsertNode()

```
method InsertNode(sourceNode As %XML.Node, ByRef child As %String) as %String
```

このノードの子として、指定したノードのコピーを挿入します。コピーするノードは、任意のドキュメントをコピー元にできます。新規ノードは、指定した子ノードの直前に挿入します。child 引数は、子ノードのノード ID です。これは参照によって渡されるため、挿入後に更新される可能性があります。挿入されたノードの NodeId が返されます。現在のノードのポインタは変更されません。

InsertTree()

```
method InsertTree(sourceNode As %XML.Node, ByRef child As %String) as %String
```

このノードの子として、指定したノードの (その子を含む) コピーを挿入します。コピーするツリーは任意のドキュメントをコピー元にできますが、ソース・ノードの下位ノードをこのノードにすることはできません。新規ノードは、指定した子ノードの直前に挿入します。child 引数は、子ノードのノード ID です。これは参照によって渡されるため、挿入後に更新される可能性があります。挿入されたノードの NodeId が返されます。現在のノードのポインタは変更されません。

Remove()

```
method Remove()
```

現在のノードを削除し、その親を現在のノードにします。

RemoveAttribute()

```
method RemoveAttribute(attributeName As %String)
```

指定された属性を削除します。

RemoveAttributeNS()

```
method RemoveAttributeNS(attributeName As %String,
                         namespace As %String)
```

指定した属性を削除します。attributeName および namespace で目的の属性を指定します。

ReplaceNode()

```
method ReplaceNode(sourceNode As %XML.Node) as %Status
```

ノードを、指定したノードのコピーで置き換えます。コピーするノードは、任意のドキュメントをコピー元にできます。現在のノードのポインタは変更されません。

ReplaceTree()

```
method ReplaceTree(sourceNode As %XML.Node) as %Status
```

ノードを、指定したノードの (そのすべての子を含む) コピーで置き換えます。コピーするツリーは任意のドキュメントをコピー元にできますが、ソース・ノードの下位ノードにすることはできません。現在のノードのポインタは変更されません。

SetAttribute()

```
method SetAttribute(attributeName As %String,
                    namespace As %String = "",
                    value As %String = "",
                    valueNamespace As %String = "")
```

現在の要素の属性のデータを設定します。以下はその説明です。

- attributeName は、属性の名前です。
- namespace は、この要素の attributeName という名前の属性の QName からのネームスペース URI です。
- value は属性値です。
- 属性値が "prefix:value" の形式である場合、valueNamespace は接頭語に相当するネームスペース URI です。

6.8.1 DOM の作成または編集の例

以下の例は、DOM の作成と編集のそれぞれのメソッドの使用法を示しています。例は互いのメソッドに基づいて作成されており、サンプルの出力 XML とペアになっています。

```
// CreateDocument
Set newdom = ##class(%XML.Document).CreateDocument("team", "t:http://example.com/example")
Set writer = ##class(%XML.Writer).%New()
do writer.Document(newdom)
// output:
//<?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example"/>

// AppendElement
set node = newdom.GetDocumentElement()
do node.AppendElement("member", "t:http://example.com/example", "Jack O'Neill")
do node.AppendElement("member", "t:http://example.com/example", "Samantha Carter")
do node.AppendElement("member", "t:http://example.com/example", "Daniel Jackson")
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    <member> Jack O'Neill</member>
//    <member> Samantha Carter</member>
//    <member> Daniel Jackson</member>
// </team>

// Append Character
set node = newdom.GetDocumentElement()
do node.MoveToFirstChild()
do node.AppendElement("status", "t:http://example.com/example")
do node.MoveToLastChild()
```

```
do node.AppendCharacter("Playable")
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter</member>
//    <member> Daniel Jackson</member>
// </team>

// AppendNode
do node.MoveToParent() // step back to parent member node
do node.MoveToNextSibling() // move to next member
set statusnode = newdom.GetDocumentElement() // make a new node
do statusnode.MoveToFirstChild() // get first member
do statusnode.MoveToLastChild() // get that status node
do node.AppendNode(statusnode) // give the next member the status node (note: but not its children!)
do node.MoveToLastChild()
do node.AppendCharacter("Playable")
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
//    <member> Daniel Jackson</member>
// </team>

// AppendTree
do node.MoveToParent()
do node.MoveToNextSibling()
do node.AppendTree(statusnode)
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
//    <member> Daniel Jackson
//      <status> Playable </status>
//    </member>
// </team>

// InsertNamespace
set nscount = newdom.CountNamespace()
w nscount // output: 2
do newdom.InsertNamespace("y:http://example2.com/example2")
set nscount = newdom.CountNamespace()
w nscount // output: 3

// InsertCharacter
set rootnode = newdom.GetDocumentElement()
set childnode = newdom.GetDocumentElement()
do childnode.MoveToFirstChild()
set childnodeid = childnode.NodeId
do rootnode.InsertCharacter("Team Go Getters", .childnodeid)
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    Team Go Getters
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
//    <member> Daniel Jackson
//      <status> Playable </status>
//    </member>
// </team>

// InsertNode
do rootnode.InsertNode(childnode,.childnodeid) // empty member node
do writer.Document(newdom)
```

```
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    Team Go Getters
//    <member/>
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
//    <member> Daniel Jackson
//      <status> Playable </status>
//    </member>
// </team>

//InsertTree
Set newdom2 = ##class(%XML.Document).CreateDocument("team2", "t2:http://example2.com/example2") // make
 a second dom
set root2 = newdom2.GetDocumentElement()
set firstchild2 = newdom2.GetDocumentElement()
set firstchild2id = firstchild2.NodeId
set lastchild = newdom.GetDocumentElement() // get the last member of the first dom
do lastchild.MoveToLastChild()
do root2.InsertTree(lastchild, .firstchild2id)
do writer.Document(newdom2)
// output:
//<?xml version="1.0" encoding="UTF-8"?>
// <member xmlns="t:http://example.com/example">Daniel Jackson
//   <status>Playable</status>
// </member>

// Remove
set node = newdom.GetDocumentElement()
do node.MoveToFirstChild()
do node.MoveToNextSibling()
do node.Remove() // removing the empty member node
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    Team Go Getters
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
//    <member> Daniel Jackson
//      <status> Playable </status>
//    </member>
// </team>

// SetAttribute
set node = newdom.GetDocumentElement()
do node.MoveToLastChild()
do node.SetAttribute("gender", , "man",)
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    Team Go Getters
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
//    <member gender="man"> Daniel Jackson
//      <status> Playable </status>
//    </member>
// </team>

// RemoveAttribute
do node.RemoveAttribute("gender")
do writer.Document(newdom)
// output:
// <?xml version="1.0" encoding="UTF-8"?>
// <team xmlns="t:http://example.com/example">
//    Team Go Getters
//    <member> Jack O'Neill
//      <status> Playable </status>
//    </member>
//    <member> Samantha Carter
//      <status> Playable </status>
//    </member>
```

```
//    <member> Daniel Jackson
//      <status> Playable </status>
//    </member>
// </team>

// RemoveAttributeNS
set node = newdom.GetDocumentElement()
do node.MoveToLastChild()
do node.SetAttribute("gender", "t:http://example.com/example", "man", "http://example.com/example")
do node.RemoveAttributeNS("gender", "t:http://example.com/example")
// before:
// <--snip-->
// <member gender="s01:man" xmlns:s01="http://example.com/example"> Daniel Jackson
//      <status> Playable </status>
// </member>

// after:
// <--snip-->
// <member> Daniel Jackson
//      <status> Playable </status>
// </member>

// ReplaceNode
do firstchild2.ReplaceNode(node) // replace the member element in the second DOM with a member node
do writer.Document(newdom2)
// output:
//<?xml version="1.0" encoding="UTF-8"?>
// <member xmlns="t:http://example.com/example"/>

// ReplaceTree
do firstchild2.ReplaceTree(node) // Replace the empty member node with a populated tree
do writer.Document(newdom2)
// output:
//<?xml version="1.0" encoding="UTF-8"?>
// <member xmlns="t:http://example.com/example">Daniel Jackson
//   <status>Playable</status>
// </member>
```

6.9 DOM からの XML 出力の記述

DOM または DOM のノードをシリアル化し、XML 出力を生成することができます。このためには、%XML.Writer の以下のメソッドを使用します。

Document()

```
method Document(document As %XML.Document) as %Status
```

%XML.Document のインスタンスを指定する場合、このメソッドは現在指定されている出力先にドキュメントを記述します。

DocumentNode()

```
method DocumentNode(document As %XML.Node) as %Status
```

%XML.Node のインスタンスを指定する場合、このメソッドは現在指定されている出力先にノードを記述します。

Tree()

```
method Tree(node As %XML.Node) as %Status
```

%XML.Node のインスタンスを指定する場合、このメソッドは現在指定されている出力先にノードとその下位ツリーを記述します。

出力先の指定と %XML.Writer のプロパティの設定の詳細は、"オブジェクトからの XML 出力の記述" を参照してください。

7

オブジェクトからの XML 出力の記述：基礎

ここでは、InterSystems IRIS® データ・プラットフォーム・オブジェクトからの XML 出力、任意のデータの XML 出力、またはその組み合わせを記述する方法についての基本を説明します。詳細については、トピック "オブジェクトからの XML 出力の記述：詳細" を参照してください。

"DOM からの XML 出力の記述" では、代替手法についても説明しています。

7.1 概要

XML を記述するには、以下のようにします。

- 特定のオブジェクトの出力が必要な場合は、そのオブジェクトのクラス定義を XML 対応にする必要があります。いくつかの例外はありますが、そのオブジェクトが参照するクラスも、%XML.Adaptor の拡張とします。
- %XML.Writer のインスタンスを作成してから、そのインスタンスのメソッドを呼び出します。コードによって目的の XML 出力の全体構造を指定するライター・メソッドを指定します。この全体構造では、文字のエンコード、オブジェクトの記述順序、処理命令も対象とするかどうかなどを指定します。

次のターミナル・セッションは、サンプルの XML 対応クラスを参照する簡単な例を示しています。

Terminal

```
GXML>Set p=##class(GXML.Person).%OpenId(1)

GXML>Set w=##class(%XML.Writer).%New()

GXML>Set w.Indent=1

GXML>Set status=w.RootObject(p)
<?xml version="1.0" encoding="UTF-8"?>
<Person GroupID="R9685">
  <Name>Zimmerman,Jeff R.</Name>
  <DOB>1961-10-03</DOB>
  <Address>
    <City>Islip</City>
    <Zip>15020</Zip>
  </Address>
  <Doctors>
    <Doctor>
      <Name>Sorenson,Chad A.</Name>
    </Doctor>
    <Doctor>
      <Name>Sorenson,Chad A.</Name>
    </Doctor>
    <Doctor>
      <Name>Uberoth,Roger C.</Name>
    </Doctor>
  </Doctors>
</Person>
```

7.2 全体構造

コードは、以下の操作の一部またはすべてをこの順序で実行する必要があります。

1. 無効である可能性のあるオブジェクトを使用する場合は、そのオブジェクトの %ValidateObject() メソッドを呼び出し、返されるステータスをチェックします。オブジェクトが有効ではない場合、XML も有効とはなりません。

   %XML.Writer ではエクスポートの前にオブジェクトの検証はしません。これは、オブジェクトの作成完了後にそれが未検証である場合、(例えば、必須プロパティが失われてるために) オブジェクト (XML) が無効となる可能性があることを意味します。

2. %XML.Writer クラスのインスタンスを作成し、オプションでそのプロパティを設定します。

   特に以下のプロパティを設定する必要がある場合があります。

   - Indent ー 出力にインデントと行のラッピングを適用するかどうかを制御します。Indent を 1 に設定するとインデントと行のラッピングが適用され、Indent を 0 に設定すると出力は長い単一の行となります。既定値は 0 です。"インデント・オプションに関する詳細" を参照してください。
   - IndentChars ー インデントに使用する文字を指定します。既定では、2 つのスペースで構成される文字列です。Indent を 0 に設定すると、このプロパティには何の効果もなくなります。
   - Charset ー 使用する文字セットを指定します。"出力の文字セットの指定" を参照してください。

   ドキュメントを読みやすくするために、ここで挙げる例では Indent を 1 に設定します。

   また、"プロローグに影響するプロパティ" および "ライターのその他のプロパティ" も参照してください。

3. 出力先を指定します。

   既定では、現在のデバイスに出力されます。出力先を指定するには、ドキュメントの出力を開始する前に、以下のメソッドのいずれかを呼び出します。

   - OutputToDevice() ー 出力先を現在のデバイスに設定します。
   - OutputToFile() ー 出力先を指定されたファイルに設定します。パスの指定は、絶対パスでも相対パスでもかまいません。存在するディレクトリ・パスを指定するようにします。
   - OutputToString() ー 出力先を文字列に設定します。その後で、他のメソッドを使用してこの文字列を取得できます。
   - OutputToStream() ー 出力先を指定されたストリームに設定します。

4. ドキュメントを開始します。これには StartDocument() メソッドを使用できます。StartDocument() でドキュメントを開始していない場合は、Write()、WriteDocType()、RootElement()、WriteComment()、および WriteProcessingInstruction() の各メソッドを使用すると、暗黙的にドキュメントを開始できます。

5. 必要に応じて、ドキュメントのプロローグの行を記述します。使用できるメソッドは以下のとおりです。

   - WriteDocType() ー DOCTYPE 宣言を記述します。"ドキュメント・タイプの定義の生成" を参照してください。
   - WriteProcessingInstruction() ー プロセス指示を記述します。"処理命令の記述" を参照してください。

6. 必要に応じて、既定のネームスペースを指定します。XML ネームスペースが定義されていないクラスにはこれが使用されます。"既定のネームスペースの指定" を参照してください。

   必要に応じて、ルート要素にネームスペース宣言を追加します。このためには、ルート要素を開始するにいくつかのユーティリティ・メソッドを呼び出すことができます。"ネームスペース宣言の追加" を参照してください。

7. ドキュメントのルート要素の記述を開始します。その詳細は、そのルート要素を InterSystems IRIS オブジェクトに対応させるかどうかで異なります。以下の 2 つの可能性があります。

- ルート要素は InterSystems IRIS オブジェクトに直接対応している可能性があります。多くの場合、これに該当するのは、ある 1 つのオブジェクトのみの出力を生成する場合です。

  この場合は、RootObject() メソッドを使用し、指定の XML 対応オブジェクトをルート要素として記述します。

- ルート要素を要素セットの単なるラッパにでき、これらの要素は InterSystems IRIS オブジェクトです。

  この場合は、RootElement() メソッドを使用して、指定した名前を持つルートレベル要素を挿入します。

また、"ルート要素の記述" も参照してください。

8. RootElement() メソッドを使用した場合は、1 つ以上の要素の出力をルート要素の中に生成するメソッドを呼び出します。ルート要素の中には、自由な順序と論理であらゆる要素を記述できます。個々の要素を記述する方法にはいくつか種類があり、これらを組み合わせることもできます。

   - Object() メソッドを使用して、XML 対応オブジェクトを記述できます。この要素の名前を指定します。または、オブジェクトで定義されている既定の名前を使用することもできます。
   - Element() メソッドを使用すると、要素の開始タグを指定した名前で記述できます。それに続いて、WriteAttribute()、WriteChars()、WriteCData()、およびその他のメソッドを使用して、要素のコンテンツ、属性、子要素を記述できます。この子要素には、他の Element() または Object() を使用できます。要素の終了を示すには、EndElement() メソッドを使用します。
   - %XML.Element クラスを使用して、要素を手動で構築できます。

   詳細は、"要素の手動作成" を参照してください。

9. RootElement() メソッドを使用した場合、EndRootElement() メソッドを呼び出します。このメソッドを使用すると、ドキュメントのルート要素が閉じ、インデントを使用している場合は、ドキュメントの構成に応じてインデントが減少します。
10. StartDocument() メソッドを使用して開始したドキュメントでは、EndDocument() メソッドを呼び出してドキュメントを閉じます。
11. 出力先を文字列とした場合は、GetXMLString() メソッドを使用するとその文字列を取得できます。

この他にも作成方法は考えられますが、特定の状況でなければ呼び出せないメソッドがある点は注意を要します。具体的には、あるドキュメントの記述を開始すると、そのドキュメントを終了しない限り、他のドキュメントを開始できません。ドキュメントを開始したまま、他のドキュメントを開始しようとすると、ライター・メソッドから次のステータスが返されます。

```
#6275: Cannot output a new XML document or change %XML.Writer properties
until the current document is completed.
```

StartDocument() メソッドは、ドキュメントを明示的に開始します。Write()、WriteDocType()、RootElement()、WriteComment()、および WriteProcessingInstruction() など、他のメソッドはドキュメントを暗黙的に開始します。

注釈　ここで説明したメソッドは、具体的な構成単位を XML ドキュメントに記述できるように設計されていますが、ドキュメントに対するさらに高度な制御が必要なこともあります。%XML.Writer クラスには別のメソッドとして Write() があります。このメソッドを使用すると、出力の中のどの位置にでも任意の文字列を記述できます。

また、Reset() メソッドを使用して、ライター・プロパティを再初期化できます。これは、XML ドキュメントを生成し、ライター・インスタンスを新しく作成せずに別の XML ドキュメントを生成しようとする場合に役立ちます。

7.3 エラーのチェック

%XML.Writer のメソッドのほとんどはステータスを返します。メソッドを実行するたびに返されるステータスを確認し、必要に応じて実行を中止します。

7.4 コメント行の挿入

コメント行を挿入するには WriteComment() メソッドを使用します。このメソッドは、ドキュメント内の任意の場所で使用できます。XML ドキュメントの記述を開始していない場合は、このメソッドを実行すると暗黙的にドキュメントが開始されます。

7.5 例

以下の例は、指定された XML 対応オブジェクトの XML 出力を生成します。

Class Member

```
ClassMethod Write(obj) As %Status
{
    set writer=##class(%XML.Writer).%New()
    set writer.Indent=1

    //these steps are not really needed because
    //this is the default destination
    set status=writer.OutputToDevice()
    if $$$ISERR(status) {
        do $System.Status.DisplayError(status)
        quit $$$ERROR($$$GeneralError, "Output destination not valid")
    }

    set status=writer.RootObject(obj)
    if $$$ISERR(status) {
        do $System.Status.DisplayError(status)
        quit $$$ERROR($$$GeneralError, "Error writing root object")
    }

    quit status
}
```

この例は OutputToDevice() メソッドを使用して、出力を既定の出力先である現在のデバイスに転送することに注意してください。

このメソッドを以下のように実行するとします。

ObjectScript

```
 set obj=##class(GXML.Person4).%OpenId(1)
 do ##class(MyWriters.BasicWriter).Write(obj)
```

出力は当然、使用されるクラスに依存しますが、以下のようになる場合があります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Person4>
  <Name>Tesla,Alexandra L.</Name>
  <DOB>1983-11-16</DOB>
  <GroupID>Y9910</GroupID>
  <Address>
    <City>Albany</City>
    <Zip>24450</Zip>
  </Address>
  <Doctors>
    <Doc>
      <Name>Schulte,Frances T.</Name>
    </Doc>
    <Doc>
      <Name>Smith,Albert M.</Name>
    </Doc>
  </Doctors>
</Person4>
```

7.6 インデント・オプションに関する詳細

ライターの Indent プロパティを使用して、出力結果に追加の改行が含まれるようにして、より読みやすいものにすることができます。このフォーマットに対する正式な仕様はありません。ここでは、Indent が 1 に等しい場合に %XML.Writer によって使用されるルールについて説明します。

- 空白文字のみを含む要素があれば、空の要素に変換されます。
- 各要素は、別々の行に配置されます。
- ある要素が、先行する要素の子である場合には、その要素は親要素を基準にしてインデントされます。このインデント設定は、2 つのスペースを既定値とする IndentChars プロパティによって決定されます。

8

オブジェクトからの XML 出力の記述：詳細

ここでは、InterSystems IRIS® データ・プラットフォーム・オブジェクトからの XML 出力、任意のデータの XML 出力、またはその組み合わせを記述する際に実行する特定のタスクについて詳しく説明します。

8.1 文字セットの指定

出力ドキュメントに使用する文字セットを指定するには、ライター・インスタンスの Charset プロパティを設定します。オプションには、"UTF-8"、"UTF-16"、および InterSystems IRIS でサポートされるその他の文字セットが含まれます。

既定のエンコードの詳細は、"入出力の文字エンコード" を参照してください。

8.2 プロローグの記述

XML ファイルのプロローグ (ルート要素の前のセクション) には、ドキュメント・タイプの定義、処理命令、およびコメントを含めることができます。このセクションでは、以下の項目について説明します。

- プロローグに影響するプロパティ
- ドキュメント・タイプの定義を生成する方法 (必要な場合)
- 処理命令を生成する方法

8.2.1 プロローグに影響するプロパティ

ライター・インスタンスで、以下のプロパティがプロローグに影響します。

Charset

XML 宣言の文字セット宣言、および (それに対応して) 出力に使用する文字セットのエンコードを制御します。"出力の文字セットの指定" を参照してください。

NoXmlDeclaration

出力に XML 宣言を記述するかどうかを制御します。ほとんどの場合、既定値は 0 で、XML 宣言が記述されます。文字セットを指定していない場合に文字列または文字ストリームを出力先に指定すると、既定値は 1 で、宣言は記述されません。

8.2.2 ドキュメント・タイプの定義の生成

ルート要素の前にドキュメント・タイプの定義を記述して、ドキュメントで使用するスキーマを定義できます。ドキュメント・タイプの定義を生成するには、WriteDocType() メソッドを使用します。このメソッドには、必須の引数が 1 つ、オプションの引数が 3 つあります。このドキュメントで目的とする範囲では、ドキュメント・タイプの定義に記述する内容として以下のものが考えられます。

```
<!DOCTYPE doc_type_name external_subset [internal_subset]>
```

ここに示すように、ドキュメント・タイプには名前があり、その名前は、XML の規則に従ってルート要素の名前にする必要があります。この定義では、外部サブセットまたは内部サブセット、あるいはその両方を指定できます。

External_subset セクションは、他の場所にある DTD ファイルを指定します。このセクションの構造は以下のいずれかです。

```
PUBLIC public_literal_identifier
PUBLIC public_literal_identifier system_literal_identifier
SYSTEM system_literal_identifier
```

ここでは、public_literal_identifier および system_literal_identifier は DTD の URI を指定する文字列です。DTD では、パブリック識別子とシステム識別子の両方を指定できます。以下は、パブリック識別子とシステム識別子の両方を使用した外部サブセットを持つドキュメント・タイプの定義の例です。

```
<!DOCTYPE hatches <!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">>
```

internal_subset セクションはエンティティ宣言のセットです。以下は、定義の内部セットのみを持つドキュメント・タイプの定義の例です。

```
<!DOCTYPE teams [ <!ENTITY baseball team (name,city,player*)>
                  !ENTITY name (#PCDATA)>
                  !ENTITY city (#PCDATA)>
                  !ENTITY player (#PCDATA)>
  ] >
```

WriteDocType() メソッドは以下の 4 つの引数をとります。

- 最初の引数は、XML ドキュメントの中で使用できるように、ドキュメント・タイプの名前を指定します。この引数は必須で、有効な XML 識別子を指定する必要があります。また、このドキュメント内のルート・レベル要素の名前として、これと同じ名前を使用する必要があります。
- 2 番目と 3 番目の引数はオプションで、以下のように定義の外部部分を指定します。

  テーブル 8-1: WriteDocType の引数

2 番目の引数	3 番目の引数	結果として得られる外部部分
“publicURI”	null	PUBLIC “publicURI”
“publicURI”	“systemURI”	PUBLIC “publicURI” “systemURI”
null	“systemURI”	SYSTEM “systemURI”

- 4 番目の引数もオプションで、定義の内部部分を指定します。この引数に null 以外の値を指定すると、その値は角括弧 ([]) で囲まれ、定義末尾の適切な場所に置かれます。これ以外の文字は追加されません。

8.2.3 処理命令の記述

XML に処理命令を記述するには、WriteProcessingInstruction() メソッドを使用します。このメソッドは以下の 2 つの引数をとります。

1. 処理命令の名前 (ターゲット)
2. 文字列で記述した処理命令

このメソッドを使用すると、以下のような内容が XML に記述されます。

```
<?name instructions?>
```

例えば、以下の処理命令を記述するとします。

```
<?xml-stylesheet type="text/css" href="mystyles.css"?>
```

これは、以下のように WriteProcessingInstruction() メソッドを呼び出すことで実現できます。

ObjectScript

```
 set instructions="type=""text/css"" href=""mystyles.css"""
 set status=writer.WriteProcessingInstruction("xml-stylesheet", instructions)
```

8.3 既定のネームスペースの指定

ライター・インスタンスでは、既定のネームスペースを指定できます。既定のネームスペースは、NAMESPACE パラメータが設定されていないクラスにのみ適用されます。選択肢はいくつかあります。

- 既定のネームスペースを指定できます。4 つの主な出力メソッド (RootObject()、RootElement()、Object()、または Element()) すべてがネームスペースを引数として受け入れます。NAMESPACE パラメータがクラス定義で設定されない場合のみ、該当する要素がネームスペースに割り当てられます。
- ライター・インスタンスの全体の既定ネームスペースを指定できます。そのためには、ライター・インスタンスの DefaultNamespace プロパティの値を指定します。

8.3.1 例

以下のクラスを考えてみます。

```
Class Writers.BasicDemoPerson Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter XMLNAME="Person";

Property Name As %Name;

Property DOB As %Date;

}
```

既定では、このクラスのオブジェクトを単純にエクスポートする場合は、出力は以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Person>
  <Name>Persephone MacMillan</Name>
  <DOB>1976-02-20</DOB>
</Person>
```

対照的に、DefaultNamespace をライター・インスタンスの "http://www.person.org" に設定してオブジェクトをエクスポートする場合は、出力は以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Person xmlns="www.person.org">
  <Name>Persephone MacMillan</Name>
  <DOB>1976-02-20</DOB>
</Person>
```

この場合、既定のネームスペースが <Person> 要素で使用されます。この要素は、それ以外ではネームスペースに割り当てられません。

8.4 ネームスペース宣言の追加

8.4.1 既定の動作

%XML.Writer クラスは自動的にネームスペース宣言を挿入し、ネームスペース接頭語を生成し、その接頭語を適切な場所に適用します。例えば、以下のクラス定義について考えてみます。

Class Definition

```
Class ResearchWriters.PersonNS Extends (%Persistent, %XML.Adaptor)
{

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

Parameter XMLNAME = "Person";

Property Name As %Name;

Property DOB As %Date;

}
```

このクラスのオブジェクトを複数エクスポートする場合は、以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <Person xmlns="http://www.person.com">
    <Name>Persephone MacMillan</Name>
    <DOB>1975-09-15</DOB>
  </Person>
  <Person xmlns="http://www.person.com">
    <Name>Joseph White</Name>
    <DOB>2003-01-31</DOB>
  </Person>
...
```

ネームスペース宣言は、各 <Person> 要素に自動的に追加されます。ドキュメントのルートにのみ追加する方が好ましいと思われるかもしれません。

8.4.2 宣言の手動追加

XML 出力にネームスペースを導入するタイミングを制御できます。次のメソッドはすべて、その直後に記述される要素に影響を及ぼします。ただし、その要素の後に記述される要素には影響しません。使用上の便宜のために、いくつかのメソッドでは標準の W3 ネームスペースを追加します。

一般的には、これらのメソッドを使用してネームスペース宣言をドキュメントのルート要素に追加します。つまり、RootObject() または RootElement() を呼び出す前に、これらのメソッドを 1 つ以上呼び出します。

注釈　これらのメソッドはいずれも、ネームスペースに要素を割り当てず、これらのネームスペースが既定のネームスペースとして追加されることはありません。特定の要素を生成する際は、使用するネームスペースを示します。"ルート要素の記述" および "XML 要素の生成" を参照してください。

AddNamespace()

```
method AddNamespace(namespace As %String,
                    prefix As %String,
                    schemaLocation As %String) as %Status
```

指定のネームスペースを追加します。ここで、namespace は追加するネームスペース、prefix はこのネームスペースのオプションの接頭語、schemaLocation は対応するスキーマの場所を示すオプションの URI です。

接頭語を指定しない場合、接頭語が自動的に生成されます (`s01`、`s02` などの形式)。

以下の例は、このメソッドの効果を示しています。まず、`Person` クラスが (NAMESPACE クラス・パラメータを使用して) ネームスペースに割り当てられているとします。先に AddNamespace() メソッドを呼び出さずにこのクラスのインスタンスの出力を生成すると、以下のような結果が得られます。

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <Person xmlns="http://www.person.org">
    <Name>Love,Bart Y.</Name>
...
```

別の方法として、ルート要素を記述する前に、以下のように AddNamespace() メソッドを呼び出すとします。

ObjectScript

```
 set status=writer.AddNamespace("http:///www.person.org","p")
```

その後でルート要素を生成すると、以下の結果が得られます。

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:p="http:///www.person.org">
  <Person xmlns="http://www.person.org">
...
```

または次のように、関連付けられたスキーマの場所を示す 3 番目の引数を指定して AddNamespace() メソッドを呼び出します。

ObjectScript

```
 set
status=writer.AddNamespace("http:///www.person.org","p","http://www.MyCompany.com/schemas/person.xsd")
```

その後で Root 要素を生成すると、以下の結果が得られます。

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:p="http:///www.person.org"
xsi:schemaLocation="http:///www.person.org http://www.MyCompany.com/schemas/person.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Person xmlns="http://www.person.org">
...
```

AddInstanceNamespace()

```
method AddInstanceNamespace(prefix As %String) as %Status
```

W3 スキーマのインスタンスのネームスペースを追加します。prefix はこのネームスペースで使用するオプションの接頭語です。既定の接頭語は xsi です。

以下はその例です。

```
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
```

AddSchemaNamespace()

```
method AddSchemaNamespace(prefix As %String) as %Status
```

W3 スキーマのネームスペースを追加します。prefix はこのネームスペースで使用するオプションの接頭語です。既定の接頭語は s です。

以下はその例です。

```
<Root xmlns:s="http://www.w3.org/2001/XMLSchema">
...
```

AddSOAPNamespace()

```
method AddSOAPNamespace(soapPrefix As %String,
                        schemaPrefix As %String,
                        xsiPrefix As %String) as %Status
```

W3 SOAP エンコードのネームスペース、SOAP スキーマのネームスペース、および SOAP スキーマのインスタンスのネームスペースを追加します。このメソッドは、各ネームスペースの接頭語を示すオプションの引数を 1 つずつ、計 3 つの引数をとります。各ネームスペースの既定の引数は、それぞれ SOAP-ENC、s、および xsi です。

以下はその例です。

```
<Root 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">
...
```

"SOAP でエンコードされた XML の生成" も参照してください。

AddSOAP12Namespace()

```
method AddSOAP12Namespace(soapPrefix As %String,
                          schemaPrefix As %String,
                          xsiPrefix As %String) as %Status
```

W3 SOAP 1.2 エンコードのネームスペース、SOAP スキーマのネームスペース、および SOAP スキーマのインスタンスのネームスペースを追加します。

以下はその例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-encoding"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
```

これらのメソッドを 1 つ以上使用できます。複数のメソッドを使用すると、それによって影響される要素には、指定したすべてのネームスペースの宣言が記述されます。

使用例は、"クラスからの XML スキーマの生成" の "より複雑なスキーマの例" を参照してください。

8.5 ルート要素の記述

XML ドキュメントにはルート要素を 1 つのみ記述する必要があります。この要素を作成するには、以下の 2 つの方法があります。

- ルート要素は InterSystems IRIS の 1 つの XML 対応オブジェクトに直接対応している可能性があります。

  この場合は、RootObject() メソッドを使用し、指定の XML 対応オブジェクトをルート要素として記述します。出力には、そのオブジェクトのすべてのオブジェクト参照が記述されます。ルート要素にはそのオブジェクトの構造が取り込まれ、そこに追加の要素を挿入することはできません。この要素の名前を指定することも、XML 対応オブジェクトで定義されている既定の名前を使用することもできます。

  前述の例では、この方法を使用しました。

- ルート要素を他の要素セット (おそらく XML 対応のオブジェクトのセット) の単なるラッパとすることも可能です。

  この場合は、RootElement() メソッドを使用して、指定した名前を持つルートレベル要素を挿入します。インデントされているドキュメントでこのメソッドを使用すると、これに続く操作でインデント・レベルが上がります。

  次に、1 つ以上の要素の出力をルート要素の中に生成する別のメソッドを呼び出します。ルートの中では、任意の順序または論理で必要な要素を追加できます。詳細は、"要素の手動作成" を参照してください。この後、EndRootElement() メソッドを呼び出してルート要素を閉じます。

  使用例は、次のセクションを参照してください。

どちらの場合でも、ルート要素に使用するネームスペースを指定できます。これは、XML 対応クラスに NAMESPACE パラメータの値がない場合にのみ適用されます。"ネームスペースの割り当ての概要" を参照してください。

ドキュメントにドキュメント・タイプの定義が含まれる場合、その DTD の名前はルート要素の名前と同じにする必要があることに注意してください。

8.6 XML 要素の生成

RootElement() を使用してドキュメントのルート要素を開始した場合、そのルート要素の中で各要素を生成する作業はユーザ側で実行する必要があります。これには以下の 3 つの方法があります。

- InterSystems IRIS オブジェクトからの要素の生成
- Element() メソッドを使用した手動の要素の作成
- %XML.Element を使用した手動の要素の作成

エクスポートされたオブジェクトのネームスペースへの割り当ての詳細は、"ネームスペース使用の制御"を参照してください。

8.6.1 要素としてのオブジェクトの生成

InterSystems IRIS オブジェクトから要素として出力を生成できます。この場合は、Object() メソッドを使用して XML 対応オブジェクトを記述します。出力には、そのオブジェクトのすべてのオブジェクト参照が記述されます。この要素の名前を指定します。または、オブジェクトで定義されている既定の名前を使用することもできます。

Object() メソッドは、RootElement() メソッドを実行してから EndRootElement() メソッドを実行するまでの間でのみ使用できます。

8.6.1.1 例

この例では、指定された XML 対応クラスのすべての保存済みインスタンスの出力を生成します。

Class Member

```
ClassMethod WriteAll(cls As %String = "", directory As %String = "c:\")
{
    if '##class(%Dictionary.CompiledClass).%ExistsId(cls) {
        Write !, "Class does not exist or is not compiled"
        Quit
    }
    Set check=$classmethod(cls,"%Extends","%XML.Adaptor")
    If 'check {
        Write !, "Class does not extend %XML.Adaptor"
        Quit
    }

    set filename=directory_cls_".xml"
    set writer=##class(%XML.Writer).%New()
    set writer.Indent=1
    set status=writer.OutputToFile(filename)
    if $$$ISERR(status) { do $System.Status.DisplayError(status) quit }

    set status=writer.RootElement("SampleOutput")
    if $$$ISERR(status) { do $System.Status.DisplayError(status) quit }

    //Get IDs of objects in the extent for the given class
    set stmt=##class(%SQL.Statement).%New()
    set status = stmt.%PrepareClassQuery(cls,"Extent")
    if $$$ISERR(status) {write "%Prepare failed:" do $SYSTEM.Status.DisplayError(status) quit}

    set rset=stmt.%Execute()
   if (rset.%SQLCODE '= 0) {write "%Execute failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
 quit}

    while (rset.%Next()) {
        //for each ID, write that object
        set objid=rset.%Get("ID")
        set obj=$CLASSMETHOD(cls,"%OpenId",objid)
        set status=writer.Object(obj)
        if $$$ISERR(status) { do $System.Status.DisplayError(status) quit }
    }
    if (rset.%SQLCODE < 0) {write "%Next failed:", !, "SQLCODE ", rset.%SQLCODE, ": ", rset.%Message
quit}

    do writer.EndRootElement()
    do writer.EndDocument()
}
```

このメソッドを実行して得られる出力には、指定されたクラスのすべての保存済みオブジェクトが、ルート要素の中で入れ子になった形式で記述されています。Sample.Person については、出力は次のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<SampleOutput>
  <Person>
    <Name>Tillem,Robert Y.</Name>
    <SSN>967-54-9687</SSN>
    <DOB>1961-11-27</DOB>
    <Home>
      <Street>3355 First Court</Street>
```

```
      <City>Reston</City>
      <State>WY</State>
      <Zip>11090</Zip>
    </Home>
    <Office>
      <Street>4922 Main Drive</Street>
      <City>Newton</City>
      <State>NM</State>
      <Zip>98073</Zip>
    </Office>
    <FavoriteColors>
      <FavoriteColorsItem>Red</FavoriteColorsItem>
    </FavoriteColors>
    <Age>47</Age>
  </Person>
  <Person>
    <Name>Waters,Ed X.</Name>
    <SSN>361-66-2801</SSN>
    <DOB>1957-05-29</DOB>
    <Home>
      <Street>5947 Madison Drive</Street>
...
</SampleOutput>
```

8.6.2 要素の手動作成

手動で XML 要素を作成できます。この場合は、Element() メソッドを使用して、要素の開始タグを指定した名前で記述します。それに続いて、要素のコンテンツ、属性、子要素を記述できます。要素の終了を示すには、EndElement() メソッドを使用します。

関連するメソッドは以下のとおりです。

Element()

```
method Element(tag, namespace As %String) as %Status
```

開始タグを記述します。要素のネームスペースを指定できます。これは、XML 対応クラスに NAMESPACE パラメータの値がない場合にのみ適用されます。"ネームスペースの割り当ての概要" を参照してください。

WriteAttribute()

```
method WriteAttribute(name As %String,
                      value As %String  = "",
                      namespace As %String,
                      valueNamespace As %String  = "",
                      global As %Boolean  = 0) as %Status
```

属性を記述します。属性名と属性値を指定する必要があります。引数 namespace は、属性名のネームスペースです。引数 valueNamespace は、属性値のネームスペースであり、XML スキーマ・ネームスペースで値が定義されるときに使用されます。

属性が、関連付けられている XML スキーマでグローバルであり、接頭語を持つ必要がある場合は、global に対して True を指定します。

このメソッドを使用する場合は、Element() の直後 (または RootElement() の後) に使用する必要があります。

WriteChars()

```
method WriteChars(text) as %Status
```

文字列を記述します。要素のコンテンツに適した文字列とするために必要なエスケープ処理も実行します。引数のデータ型は、%String または %CharacterStream とする必要があります。

WriteCData()

```
method WriteCData(text) as %Status
```

CDATA セクションを記述します。引数のデータ型は、%String または %CharacterStream とする必要があります。

WriteBase64()

```
method WriteBase64(binary) as %Status
```

指定したバイナリ・バイトを Base-64 としてエンコードし、得られたテキストを要素のコンテンツとして記述します。引数のデータ型は、%Binary または %BinaryStream とする必要があります。

WriteBinHex()

```
method WriteBinHex(binary) as %Status
```

指定したバイナリ・バイトを BINHEX としてエンコードし、得られたテキストを要素のコンテンツとして記述します。引数のデータ型は、%Binary または %BinaryStream とする必要があります。

EndElement()

```
method EndElement() as %Status
```

該当する要素を終了します。

これらのメソッドは、RootElement() メソッドを実行してから EndRootElement() メソッドを実行するまでの間でのみ使用できます。

注釈　ここで説明したメソッドは、具体的で論理的な内容を XML ドキュメントに記述できるように設計されていますが、ドキュメントに対するさらに高度な制御が必要なこともあります。%XML.Writer クラスには別のメソッドとして Write() があります。このメソッドを使用すると、任意の文字列を記述できます。ただし、その結果を検証する機能は用意されていないので、得られた結果が適格な XML ドキュメントであることをユーザ側で確認する必要があります。

8.6.2.1 例

以下に例を示します。

ObjectScript

```
//write output to current device
//simple demo of Element & related methods

set writer=##class(%XML.Writer).%New()
set writer.Indent=1

set status=writer.OutputToDevice()
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.StartDocument()
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.RootElement("root")
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.Element("SampleElement")
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.WriteAttribute("Attribute","12345")
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.Element("subelement")
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.WriteChars("Content")
if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

set status=writer.EndElement()
```

```
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

 set status=writer.Element("subelement")
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

 set status=writer.WriteChars("Content")
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

 set status=writer.EndElement()
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

 set status=writer.EndElement()
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

 set status=writer.EndRootElement()
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}

 set status=writer.EndDocument()
 if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}
```

このルーチンの出力は以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <SampleElement Attribute="12345">
    <subelement>Content</subelement>
    <subelement>Content</subelement>
  </SampleElement>
</root>
```

8.6.3 %XML.Element の使用

前述のセクションでは、Element() を使用して生成する要素を指定しましたが、ネームスペースを指定することもできます。場合によっては、要素名ではなく %XML.Element クラスのインスタンスを使用する必要がある場合があります。このクラスには、以下のプロパティがあります。

- Local プロパティは、この要素が親要素に対してローカルかどうかを指定し、ネームスペースの制御に影響を与えます。詳細は、後述の "要素が親に対してローカルかどうかの制御" を参照してください。
- Namespace プロパティは、この要素のネームスペースを指定します。
- Tagname プロパティは、この要素の名前を指定します。

ここでは、WriteAttribute() メソッドも使用できます。

8.7 ネームスペース使用の制御

"オブジェクトの XML への投影" で説明されているように、クラスをネームスペースに割り当てて、対応する XML 要素がそのネームスペースに属するようにすることができます。また、そのクラスのプロパティがそのネームスペースに属するかどうかを制御できます。

そのクラスのオブジェクトを XML にエクスポートすると、%XML.Writer クラスには、要素がその親に対してローカルであるかどうかの指定などのオプションが追加されます。知っておく必要があることを以下に示します。

- %XML.Writer クラスの既定でのネームスペースの処理方法
- ローカル要素を修飾付きにするかどうかを指定する方法
- 要素が親に対してローカルかどうか指定する方法
- 属性を修飾付きにするかどうかを指定する方法

- ネームスペースの割り当て方法の概要

注釈　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。一般に、各クラスには独自のネームスペース宣言がありますが、通常は、必要なのは 1 つまたは少数のネームスペースのみです。関連情報も、(グローバルにではなく) クラスごとに指定します。これには、要素が親に対してローカルかどうか、および下位要素が修飾されるかどうかを制御する設定が含まれます。作業を単純化するために、一貫性のある方法を使用することをお勧めします。

8.7.1 ネームスペースの既定の処理

XML 対応クラスをネームスペースに割り当てるには、"オブジェクトの XML への投影" の説明にあるとおり、そのクラスの NAMESPACE パラメータを設定します。**%XML.Writer** クラスは自動的にネームスペース宣言を挿入し、ネームスペース接頭語を生成し、その接頭語を適切な場所に適用します。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.Objects.WithNamespaces.Person Extends (%Persistent, %Populate, %XML.Adaptor)
{
Parameter NAMESPACE = "http://www.person.com";
Property Name As %Name [ Required ];
Property DOB As %Date(FORMAT = 5, MAXVAL = "+$h") [ Required ];
Property GroupID As %Integer(MAXVAL=10,MINVAL=1,XMLPROJECTION="ATTRIBUTE");
}
```

このクラスのオブジェクトをエクスポートする場合は、以下のようになります。

XML

```
<Person xmlns="http://www.person.com" GroupID="4">
  <Name>Uberoth,Amanda Q.</Name>
  <DOB>1952-01-13</DOB>
</Person>
```

以下のことに注意してください。

- ネームスペース宣言は、各 <Person> 要素に追加されます。
- <Person> 要素のローカル要素 (<Name> および <DOB>) は、既定では修飾されています。ネームスペースは既定のネームスペースとして追加されるので、これらの要素に追加されます。
- <Person> 要素の属性 (GroupID) は、既定では修飾されていません。この属性には接頭語がないので、修飾なしと見なされます。
- ここに示す接頭語は自動的に生成されました (オブジェクトにネームスペースを割り当てる際は、ネームスペースのみを指定し、接頭語は指定しないことに注意してください)。
- この出力は、ライターでネームスペース関連のプロパティを設定せず、ライターでネームスペース関連のメソッドを使用せずに生成されました。

8.7.1.1 ネームスペース割り当てのコンテキストの効果

XML 対応オブジェクトが割り当てられるネームスペースは、そのオブジェクトが最上位レベルでエクスポートされるか、または別のオブジェクトのプロパティとしてエクスポートされるかによって異なります。

Address という名前のクラスを考えてみます。NAMESPACE パラメータを使用して、Address クラスに "http://www.address.org" というネームスペースを割り当てるとします。Address クラスのオブジェクトを直接エクスポートした場合、以下のような結果が得られます。

XML

```
<Address xmlns="http://www.address.org">
  <Street>8280 Main Avenue</Street>
  <City>Washington</City>
  <State>VT</State>
  <Zip>15355</Zip>
</Address>
```

<Address> 要素とそのすべての要素が同じネームスペース ("http://www.address.org") にあることに注意してください。

一方、Address オブジェクトであるプロパティを持つ Person クラスを考えてみます。また、NAMESPACE パラメータを使用して、Person クラスに "http://www.person.org" というネームスペースを割り当てるとします。Person クラスのオブジェクトをエクスポートした場合、以下のような結果が得られます。

XML

```
<Person xmlns="http://www.person.org">
  <Name>Zevon,Samantha H.</Name>
  <DOB>1964-05-24</DOB>
  <Address xmlns="http://www.address.org">
     <Street>8280 Main Avenue</Street>
     <City>Washington</City>
     <State>VT</State>
     <Zip>15355</Zip>
  </Address>
</Person>
```

<Address> 要素が親オブジェクト ("http://www.person.org") のネームスペースにあることに注意してください。ただし、<Address> 要素は "http://www.address.org" のネームスペース内にあります。

8.7.2 ローカル要素が修飾されるかどうかの制御

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して処理されます。そして、そのローカル要素は、XML 対応オブジェクトの ELEMENTQUALIFIED パラメータの設定に応じて処理されます。このクラス・パラメータが設定されていない場合、ライター・プロパティ ElementQualified の値が代わりに使用されます。既定では、これはリテラル形式では 1、エンコード形式では 0 です。

以下の例は、ElementQualified の既定の設定 (1) のオブジェクトを示しています。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonNS xmlns="http://www.person.com" GroupID="M9301">
  <Name>Pybus,Gertrude X.</Name>
  <DOB>1986-10-19</DOB>
</PersonNS>
```

このネームスペースは <PersonNS> 要素に既定のネームスペースとして追加されるため、<Name> および <DOB> 下位要素に適用されます。

ライター定義を変更し、ElementQualified プロパティを 0 に設定したとします。この場合、同じオブジェクトが以下のように表示されます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<s01:PersonNS xmlns:s01="http://www.person.com" GroupID="M9301">
  <Name>Pybus,Gertrude X.</Name>
  <DOB>1986-10-19</DOB>
</s01:PersonNS>
```

この場合、ネームスペースは <PersonNS> 要素に接頭語を付けて追加されます。これは <PersonNS> 要素では使用されますが、その下位要素では使用されません。

8.7.3 要素が親に対してローカルかどうかの制御

既定では、Object() メソッドを使用して要素を生成し、その要素にネームスペースがある場合、その要素は親に対してローカルではありません。ただし、要素をその親のネームスペースに属するように強制できます。そのためには、Object() メソッドのオプションの local 引数を使用します。これは、4 番目の引数です。

8.7.3.1 Local 引数 0 の場合 (既定)

ここでは、NAMESPACE クラス・パラメータを `"http://www.person.com"` に指定する `Person` クラスについて考えてみます。

ルート要素を開いてから、Object() を使用して `Person` を生成する場合、`<Person>` 要素は `"http://www.person.com"` ネームスペースに入ります。以下の例を考えてみます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns="http://www.rootns.org">
   <Person xmlns="http://www.person.com">
      <Name>Adam,George L.</Name>
      <DOB>1947-06-29</DOB>
   </Person>
</Root>
```

`<Person>` 要素を他の要素の内部にさらに深く入れ子にすると、同様の結果が得られます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns="www.rootns.org">
   <GlobalEl xmlns="globalns">
      <Inner xmlns="innerns">
         <Person xmlns="http://www.person.com">
            <Name>Adam,George L.</Name>
            <DOB>1947-06-29</DOB>
         </Person>
      </Inner>
   </GlobalEl>
</Root>
```

8.7.3.2 Local 引数 1 の場合

`<Person>` 要素をその親に対して強制的にローカルにするには、local 引数を 1 に等しく設定します。そうして前述の出力を再度生成すると、入れ子がより浅い場合、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns="http://www.rootns.org">
   <s01:Person xmlns="http://www.person.com"
xmlns:s01="http://www.rootns.org">
      <Name>Adam,George L.</Name>
      <DOB>1947-06-29</DOB>
   </s01:Person>
</Root>
```

`<Person>` 要素が現在は、その親要素のネームスペースである `"http://www.rootns.org"` ネームスペースにあることに注意してください。同様に、入れ子がより深い場合には以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns="www.rootns.org">
   <GlobalEl xmlns="globalns">
      <Inner xmlns="innerns">
         <s01:Person xmlns="http://www.person.com" xmlns:s01="innerns">
            <Name>Adam,George L.</Name>
            <DOB>1947-06-29</DOB>
         </s01:Person>
      </Inner>
   </GlobalEl>
</Root>
```

8.7.4 属性が修飾されるかどうかの制御

オブジェクトをエクスポートする際、既定では属性は修飾されません。属性を修飾するには、ライター・プロパティ AttributeQualified を 1 に等しく設定します。以下の例は、AttributeQualified が 0 に等しい（または設定されていない）ライターで生成された出力を示しています。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <s01:Person xmlns:s01="http://www.person.com" GroupID="E8401">
      <Name>Leiberman,Amanda E.</Name>
      <DOB>1988-10-28</DOB>
   </s01:Person>
</Root>
```

一方、以下の例は、AttributeQualified が 1 に等しいライターで生成された同じオブジェクトを示しています。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <s01:Person xmlns:s01="http://www.person.com" s01:GroupID="E8401">
      <Name>Leiberman,Amanda E.</Name>
      <DOB>1988-10-28</DOB>
   </s01:Person>
</Root>
```

どちらの場合も、要素は修飾されません。

8.7.5 ネームスペースの割り当ての概要

このセクションでは、XML 出力の任意の要素に対してネームスペースを指定する方法を説明します。

8.7.5.1 最上位レベル要素

最上位レベルでエクスポートされる InterSystems IRIS クラスに相当する要素の場合、以下の規則が適用されます。

1. このクラスに NAMESPACE 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. このパラメータを指定していない場合、要素はその要素（RootObject()、RootElement()、Object()、または Element()）を生成するために使用したメソッドで指定されたネームスペースに入ります。
3. RootObject()、RootElement()、Object()、または Element() の呼び出し時にネームスペースを指定していない場合、要素はライターの DefaultNamespace プロパティで指定されたネームスペースに入ります。
4. DefaultNamespace プロパティが NULL の場合、要素はどのネームスペースにも入りません。

8.7.5.2 下位レベル要素

エクスポートしているクラスの下位要素は、そのクラスの ELEMENTQUALIFIED パラメータの影響を受けます。ELEMENTQUALIFIED が設定されていない場合、ライター・プロパティ ElementQualified の値が代わりに使用されます。既定では、これはリテラル形式では 1、エンコード形式では 0 です。

要素が指定されたクラスに対して修飾される場合、そのクラスの下位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。

1. 親オブジェクトの NAMESPACE パラメータを指定した場合、下位要素はそのネームスペースに明示的に割り当てられます。
2. このパラメータを指定していない場合、下位要素はその要素 (RootObject()、RootElement()、Object()、または Element()) を生成するために使用したメソッドで指定されたネームスペースに明示的に割り当てられます。
3. RootObject()、RootElement()、Object()、または Element() でネームスペースを指定していない場合、下位要素はライターの DefaultNamespace プロパティで指定されたネームスペースに明示的に割り当てられます。
4. DefaultNamespace プロパティが NULL の場合、下位要素はどのネームスペースにも明示的に割り当てられません。

8.8 ネームスペース割り当ての外観の制御

ネームスペース割り当ての制御に加え、ネームスペース割り当てが XML 出力でどのように表示されるかを制御できます。具体的には、以下を制御できます。

- ネームスペースの割り当てを明示的と暗黙的のいずれで行うか
- ネームスペースの接頭語

8.8.1 ネームスペースの割り当てを明示的と暗黙的のいずれで行うか

ネームスペースに要素や属性を割り当てる際、XML では、ライター・インスタンスの SuppressXmlns プロパティで制御される 2 つの同等の表現が存在します。

ネームスペース "http://www.person.org" に (NAMESPACE クラス・パラメータを使用して) 割り当てられている Person という名前のオブジェクトの XML 出力を生成するとします。既定の出力例 (SuppressXmlns が 0 に等しい) は以下のとおりです。

XML

```
<Person xmlns="http://www.person.com" GroupID="4">
  <Name>Uberoth,Amanda Q.</Name>
  <DOB>1952-01-13</DOB>
</Person>
```

まったく等しいもう 1 つの形式は、以下のとおりです。これは、1 に設定された SuppressXmlns で生成され、ネームスペースに明示的に割り当てられる各要素が、そのネームスペースの接頭語で確実に表されるようにします。

XML

```
<s01:Person xmlns:s01="http://www.person.com" GroupID="4">
  <s01:Name>Uberoth,Amanda Q.</s01:Name>
  <s01:DOB>1952-01-13</s01:DOB>
</s01:Person>
```

このプロパティは、ネームスペースの割り当ての表示方法のみに影響し、ネームスペースの割り当て方法は制御しないことに注意してください。ネームスペースを使用しない場合、このパラメータは何の効果も持ちません。

8.8.2 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。

- 指定した接頭語は、指定しなかった場合に自動的に生成される接頭語の代わりに使用されます。
- 指定する接頭語は、必須ではなくても XML 出力で宣言されます。

詳細は、″オブジェクトの XML への投影″ を参照してください。

8.9 空文字列 (″″) のエクスポート方法の制御

オブジェクトを XML 対応にする場合、NULL 値および空文字列を XML に投影する方法を指定します (″オブジェクトの XML への投影″ を参照)。

その指定方法の 1 つは、XML 対応クラスで XMLIGNORENULL を `"RUNTIME"` (大文字/小文字の区別なし) に設定することです。この場合、**%XML.Writer** を使用して出力を生成するときに、InterSystems IRIS は以下のようにライターの RuntimeIgnoreNull プロパティの値を使用して、プロパティが `""` の場合の処理方法を決定します。

- ライターの RuntimeIgnoreNull プロパティが 0 (既定) の場合、XMLNIL パラメータでプロパティのエクスポート方法を制御します。XMLNIL はクラス・パラメータおよびプロパティ・パラメータですが、プロパティ・パラメータが優先されます。
  - XMLNIL が 0 (既定) の場合、プロパティは投影されません。つまり、XML ドキュメントに格納されません。
  - XMLNIL が 1 の場合、プロパティが要素として使用されていると、そのプロパティは以下のようにエクスポートされます。

    XML

    ```
    <PropName xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
    ```

  - XMLNIL が 1 の場合、プロパティが属性として使用されていると、そのプロパティはエクスポートされません。
- ライターの RuntimeIgnoreNull プロパティが 1 の場合、プロパティは空要素または空属性としてエクスポートされます (これは、常に空要素または空属性としてエクスポートされる `$char(0)` 値と同様です)。

ライターの RuntimeIgnoreNull プロパティは、XML 対応クラスで XMLIGNORENULL が `"RUNTIME"` に設定されていない限り影響はありません。

8.9.1 例 : RuntimeIgnoreNull が 0 (既定)

まず、以下のクラスについて考えてみます。

Class Definition

```
Class EmptyStrings.Export Extends (%Persistent, %XML.Adaptor)
{
```

```
Parameter XMLNAME="Test";


Parameter XMLIGNORENULL = "RUNTIME";


///project this one as an element
///XMLNIL is 0, the default
Property Property1 As %String;


///project this one as an attribute
///XMLNIL is 0, the default
Property Property2 As %String(XMLPROJECTION = "ATTRIBUTE");


///project this one as an element with XMLNIL=1
Property Property3 As %String(XMLNIL = 1);


///project this one as an attribute with XMLNIL=1
Property Property4 As %String(XMLNIL=1,XMLPROJECTION="ATTRIBUTE");

}
```

このクラスの新しいインスタンスを作成し（いずれのプロパティにも値を設定せず）、%XML.Writer を使用してその出力を生成する場合、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Test>
  <Property3 xsi:nil="true"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
</Test>
```

8.9.2 例：RuntimeIgnoreNull が 1

この例では、ライターの RuntimeIgnoreNull プロパティを 1 に設定します。1 つ前の例で使用したものと同じオブジェクトの出力を生成する場合、以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Test Property2="" Property4="">
  <Property1></Property1>
  <Property3></Property3>
</Test>
```

この場合、RuntimeIgnoreNull が 1 なので、XMLNIL パラメータは使用されません。代わりに、空属性または空要素として "" がエクスポートされます。

8.10 タイプ情報のエクスポート

XML ライターは、既定ではタイプ情報を記述しません。出力にタイプ情報を追加するには、次の 2 つの方法を使用できます。

- ライターの OutputTypeAttribute プロパティ。このプロパティが 1 の場合は、ライターは、記述するオブジェクト内のすべての要素の XML タイプ情報を追加します（これらのオブジェクト自体のタイプ情報は除く）。以下はその例です。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Person>
    <Name xsi:type="s:string">Petersburg,Greta U.</Name>
    <DOB xsi:type="s:date">1949-05-15</DOB>
  </Person>
</Root>
```

XML ドキュメントのルートに適切なネームスペースが追加されていることに注目してください。

- Object() メソッドと RootObject() メソッドの className 引数。この引数を使用して、オブジェクト (クラス名) の想定される ObjectScript タイプを指定します。

  この引数が実際のタイプと同じ場合は、ライターはオブジェクトのタイプ情報を追加しません。

  この引数が実際のタイプと異なる場合は、ライターはオブジェクトの実際の XML タイプを追加します (既定ではクラス名)。例えば、`Test2.PersonWithAddress` のインスタンスの出力を記述して、className 引数の値として `MyPackage.MyClass` を指定するとします。`MyPackage.MyClass` は実際のクラス名とは異なるため、ライターは次の出力を生成します。

  XML

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <PersonWithAddress xsi:type="PersonWithAddress"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Name>Avery,Robert H.</Name>
    <Address>
      <City>Ukiah</City>
      <Zip>82281</Zip>
    </Address>
  </PersonWithAddress>
  ```

  Object() メソッドと RootObject() メソッドの全引数のリストは、該当するクラス・リファレンスを参照してください。

8.11 SOAP でエンコードされた XML の生成

%XML.Writer クラスの場合は、Format プロパティは、出力全体の形式を制御します。これは、以下のいずれかです。

- 既定値であり、ほとんどの例で使用されている `"literal"`。
- SOAP 1.1 規格で説明されている方法でエンコードされている `"encoded"`。
- SOAP 1.2 規格で説明されている方法でエンコードされている `"encoded12"`。

8.11.1 インライン参照の作成

エンコードされた形式では、任意のオブジェクト値プロパティが参照として含められ、参照オブジェクトが個別の要素としてエクスポートされます。

こうしたプロパティを個別の要素としてではなく、インラインでエクスポートするには、(ライターの) ReferencesInline プロパティを 1 に設定します。

Format を `"literal"` に設定すると、ReferencesInline プロパティには何の効果もなくなります。

8.12 エクスポート後のアンスウィズルの制御

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。

8.13 要素を閉じる形式の制御

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。

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

Object() メソッドは常に、最初の構文で要素をエクスポートします。ここに示す 2 番目の構文で要素を閉じる必要がある場合は、"要素の手動作成" で説明しているように、オブジェクトを手動で記述します。

8.14 その他のオプション

ここでは、%XML.Writer で提供されるその他のオプションについて説明します。

- Canonicalize() メソッド
- Shallow プロパティ
- Summary プロパティ
- Base64LineBreaks プロパティ
- CycleCheck プロパティ

8.14.1 Canonicalize() メソッド

Canonicalize() メソッドは、XML ノードを正規化された形式で記述します。このメソッドには、以下のシグニチャがあります。

```
method Canonicalize(node As %XML.Node, ByRef PrefixList, formatXML As %Boolean = 0) as %Status
```

以下はその説明です。

- node はドキュメントのサブツリーであり、%XML.Node のインスタンスとして、"XML ドキュメントの DOM 表現" で説明されています。
- PrefixList は、以下のいずれかです。

- 包含的な正規化の場合は、PrefixList を "c14n" と指定します。この場合は、https://www.w3.org/TR/xml-c14n で指定されたとおり、出力は XML Canonicalization バージョン 1.0 で指定された形式となります。
- 排他的な正規化の場合は、ノードを以下のようにして PrefixList を多次元配列と指定します。

ノード	値
PrefixList(prefix)。prefix はネームスペース接頭語です。	このネームスペース接頭語と併用されるネームスペース

この場合は、https://www.w3.org/TR/xml-exc-c14n/ で指定されたとおり、出力は Exclusive XML Canonicalization バージョン 1.0 で指定された形式となります。

- formatXML により形式を制御します。formatXML が True の場合、ライターは XML 正規化仕様による指定フォーマットではなく、ライター・インスタンス用の指定フォーマットを使用します。その際の出力はキャノニック XML ではありませんが、キャノニック XML 用のネームスペース処理が行われることになります。このオプションは、Web サービスの ProcessBodyNode() コールバックの SOAP 本文など、XML ドキュメントのフラグメント出力に役立ちつつ、一部のフォーマット制御を維持し続けます。

8.14.2 Shallow プロパティ

ライター・インスタンスの Shallow プロパティは、オブジェクト値を持つ出力プロパティに影響します。このクラスのプロパティを使用すると、オブジェクト値を持つすべての出力を小型化できます。つまり、参照先オブジェクトの詳細ではなく、参照先オブジェクトの ID が出力に記述されます。このプロパティと、XML 対応オブジェクトの XMLDEFAULTREFERENCE クラス・パラメータおよび XMLREFERENCE プロパティ・パラメータとの間には、以下のテーブルに示す相互関係があります。このテーブルは、各ケースごとに得られる出力を示しています。

テーブル 8-2: Shallow = 1 の場合の影響

XMLREFERENCE および XMLDEFAULTREFERENCE の値	Shallow=1 の場合の出力
プロパティ・パラメータ "XMLREFERENCE" が "SUMMARY" または "COMPLETE"	このプロパティについては出力なし
プロパティ・パラメータ XMLREFERENCE が "ID"、"OID"、または "GUID"	このプロパティについて、ID、OID、または GUID のうち、適切なタイプの出力を生成
プロパティ・パラメータ XMLREFERENCE が未設定で、クラス・パラメータ XMLDEFAULTREFERENCE が "SUMMARY" または "COMPLETE"	このプロパティについては出力なし
プロパティ・パラメータ XMLREFERENCE が未設定で、クラス・パラメータ XMLDEFAULTREFERENCE が "ID"、"OID"、または "GUID"	このプロパティについて、ID、OID、または GUID のうち、適切なタイプの出力を生成
プロパティ・パラメータ XMLREFERENCE およびクラス・パラメータ XMLDEFAULTREFERENCE がどちらも未設定	このプロパティについては出力なし

Shallow プロパティは、値にシリアル・オブジェクトを持つプロパティ、およびオブジェクト値ではない値を持つプロパティには影響を及ぼしません。

"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" も参照してください。

8.14.3 Summary プロパティ

ライター・インスタンスの Summary プロパティは、XML 対応オブジェクト全体をエクスポートするか、その概要をエクスポートするかを制御します。以下の 2 つの値のいずれかを指定します。

- 既定値である 0 を指定すると、オブジェクト全体がエクスポートされます。
- 値 1 を指定すると、概要として指定されているプロパティがエクスポートされます。

"オブジェクトの XML への投影" で説明したように、オブジェクトの概要は XMLSUMMARY クラス・パラメータによって指定される、コンマ区切りのプロパティのリストです。

例えば、XML 対応の `Person` クラスの出力を生成すると、既定の出力が以下のようになるとします。

XML

```
<Persons>
 <Person>
  <Name>Xenia,Yan T.</Name>
  <DOB>1986-10-21</DOB>
 </Person>
 <Person>
  <Name>Vivaldi,Ashley K.</Name>
  <DOB>1981-01-25</DOB>
 </Person>
</Persons>
```

ここで、XMLSUMMARY の値を `Person` クラスの `"Name"` に設定します。この場合に Summary プロパティを 1 に設定すると、出力は以下のようになります。

XML

```
<Persons>
 <Person>
  <Name>Xenia,Yan T.</Name>
 </Person>
 <Person>
  <Name>Vivaldi,Ashley K.</Name>
 </Person>
</Persons>
```

8.14.4 Base64LineBreaks プロパティ

タイプが %Binary または %xsd.base64Binary のプロパティには、自動改行を含めることができます。このためには、ライター・インスタンスの Base64LineBreaks プロパティを 1 に設定します。この場合、ライター・インスタンスの自動改行/復改が 76 文字ごとに発生します。このプロパティの既定値は 0 です。

8.14.5 CycleCheck プロパティ

ライター・インスタンスの CycleCheck プロパティは、エラーを引き起こす可能性がある参照オブジェクト内のサイクル (エンドレス・ループ) の有無をライターが確認するかどうかを制御します。既定は 1 で、ライターがサイクルの有無を確認します。

サイクルがないことが確実な場合は、CycleCheck を 0 に設定するとパフォーマンスがわずかに向上します。

8.15 その他の例

以下のメソッドは、%XML.Writer のプロパティを試したいユーザには便利な場合があります。ライターのバージョンを指定する文字列である、入力引数を受け入れます。各ライターのバージョンは、ライター・インスタンスのプロパティの特定の設定に対応します。

Class Definition

```
Class Utils.Writer
{

/// given a "name", return a writer with those properties
ClassMethod CreateWriter(wname) As %XML.Writer
{
 set w=##class(%XML.Writer).%New()
 set w.Indent=1
 set w.IndentChars="   "
 if wname="DefaultWriter" {
  set w.Indent=0  ; set back to default
  }
 elseif wname="EncodedWriter" {
  set w.Format="encoded"
  }
 elseif wname="EncodedWriterRefInline" {
  set w.Format="encoded"
  set w.ReferencesInline=1
  }
 elseif wname="AttQualWriter" {
  set w.AttributeQualified=1
  }
 elseif wname="AttUnqualWriter" {
  set w.AttributeQualified=0 ; default
  }
 elseif wname="ElQualWriter" {
  set w.ElementQualified=1 ; default
  }
 elseif wname="ElUnqualWriter" {
  set w.ElementQualified=0
  }
 elseif wname="ShallowWriter" {
  set w.Shallow=1
  }
 elseif wname="SOAPWriter1.1" {
  set w.Format="encoded"
  set w.ReferencesInline=1
  }
 elseif wname="SOAPWriter1.2" {
  set w.Format="encoded12"
  set w.ReferencesInline=1
  }
 elseif wname="SummaryWriter" {
  set w.Summary=1
  }
 elseif wname="WriterNoXmlDecl" {
  set w.NoXMLDeclaration=1
  }
 elseif wname="WriterRefInline" {
  set w.ReferencesInline=1
  }
 elseif wname="WriterRuntimeIgnoreNull" {
  set w.RuntimeIgnoreNull=1
  }
 elseif wname="WriterSuppressXmlns" {
  set w.SuppressXmlns=1
  }
 elseif wname="WriterUTF16" {
  set w.Charset="UTF-16"
  }
 elseif wname="WriterWithDefNS" {
  set w.DefaultNamespace="www.Def.org"
  }
 elseif wname="WriterWithDefNSSuppressXmlns" {
  set w.DefaultNamespace="www.Def.org"
  set w.SuppressXmlns=1
  }
 elseif wname="WriterWithDtdSettings" {
  set w.DocType ="MyDocType"
  set w.SystemID = "http://www.mysite.com/mydoc.dtd"
```

```
  set w.PublicID = "-//W3C//DTD XHTML 1.0 Transitional//EN"
  set w.InternalSubset = ""
  }
 elseif wname="WriterXsiTypes" {
  set w.OutputTypeAttribute=1
  }
 quit w
}

}
```

以下のフラグメントは、このメソッドを使用してドキュメントの例の生成に役立てる方法を示しています。

```
/// method to write one to a file
ClassMethod WriteOne(myfile,cls,element,wname,ns,local,rootns)
{
    set writer=..CreateWriter(wname)
    set mydir=..#MyDir
    set comment="Output for the class: "_cls
    set comment2="Writer settings: "_wname

    if $extract(mydir,$length(mydir))'="/" {set mydir=mydir_"/"}
    set file=mydir_myfile
    set status=writer.OutputToFile(file)
    if $$$ISERR(status) { do $System.Status.DisplayError(status) quit }

    set status=writer.WriteComment(comment)
    if $$$ISERR(status) { do $System.Status.DisplayError(status) quit }

    set status=writer.WriteComment(comment2)
...
}
```

出力にコメント行が 2 つ含まれることに注意してください。1 つは、ファイルに表示される XML 対応クラスの名前を示します。もう 1 つは、ファイルの生成に使用されるライター設定の名前を示します。出力ディレクトリは集中的に（パラメータで）制御され、この汎用メソッドには RootElement() メソッドおよび Object() メソッドの両方に渡される引数が含まれます。

9

XML ドキュメントの暗号化

このトピックでは、XML ドキュメントを暗号化する方法を説明します。

Tipヒント このネームスペースで SOAP ロギングを有効にしておくと、エラーの詳細を受信するので便利です。"InterSystems IRIS での SOAP の問題のトラブルシューティング" の "InterSystems IRIS SOAP ログ" を参照してください。

9.1 暗号化された XML ドキュメントについて

暗号化された XML ドキュメントには以下の要素が含まれます。

- <EncryptedData> 要素。ここには、ランダムに生成された対称鍵によって暗号化されたデータが含まれます (公開鍵よりも対称鍵を使用した方が、暗号化の効率はアップします)。
- 1 つ以上の <EncryptedKey> 要素。それぞれの <EncryptedKey> 要素には、データの暗号化に使用された対称鍵を暗号化したコピーが保持されています。また、公開鍵を使用した X.509 証明書も含まれています。一致する秘密鍵を持っている受信者は、対称鍵を解読してから <EncryptedData> 要素を解読できます。
- (オプション) クリア・テキストによる他の要素

以下に例を示します。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Container xmlns="http://www.w3.org/2001/04/xmlenc#">
  <EncryptedKey>
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
      <DigestMethod xmlns="http://www.w3.org/2000/09/xmldsig#"
                    Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
    </EncryptionMethod>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <X509Data>
        <X509Certificate>MIICnDCCAYQCAWUwDQYJKo... content omitted</X509Certificate>
      </X509Data>
    </KeyInfo>
    <CipherData>
      <CipherValue>J2DjVgcB8vQx3UCy5uejMB ... content omitted</CipherValue>
    </CipherData>
    <ReferenceList>
      <DataReference URI="#Enc-E0624AEA-9598-4436-A154-F746B07A2C55"></DataReference>
    </ReferenceList>
  </EncryptedKey>
  <EncryptedData Id="Enc-E0624AEA-9598-4436-A154-F746B07A2C55"
                 Type="http://www.w3.org/2001/04/xmlenc#Content">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc">
    </EncryptionMethod>
    <CipherData>
      <CipherValue>LmoBK7+nDelTOsC3 ... content omitted</CipherValue>
```

```
    </CipherData>
  </EncryptedData>
</Container>
```

暗号化されたドキュメントを作成するには、クラス %XML.Security.EncryptedData および %XML.Security.EncryptedKey を使用します。これら XML 対応のクラスは、適切なネームスペースにある有効な <EncryptedData> 要素と <EncryptedKey> に投影されます。

9.2 暗号化された XML ドキュメントの作成

暗号化された XML ドキュメントを作成する最も簡単な方法は、以下のとおりです。

1. 目的の XML ドキュメントに直接投影できる汎用コンテナ・クラスを定義して使用します。
2. 暗号化する XML を格納したストリームを作成します。
3. 対応する暗号化キーと共に、そのストリームを暗号化して、コンテナ・クラスの適切なプロパティに書き込みます。
4. コンテナ・クラスの XML 出力を生成します。

9.2.1 暗号化の前提条件

ドキュメントを暗号化するには、まず暗号化されたドキュメントの送信先となるエンティティの証明書を含む InterSystems IRIS® データ・プラットフォーム資格情報セットを作成する必要があります。この場合、関連付けられている秘密鍵は必要はありません (また、持つべきではありません)。詳細は、"設定およびその他の一般的なアクティビティ" を参照してください。

9.2.2 コンテナ・クラスの要件

汎用コンテナ・クラスには、以下のものが含まれている必要があります。

- タイプが %XML.Security.EncryptedData で、<EncryptedData> 要素として投影されているプロパティ。

  このプロパティには、暗号化されたデータが保持されます。

- タイプが %XML.Security.EncryptedKey で、<EncryptedKey> 要素として投影されている、少なくとも 1 つのプロパティ。

  これらのプロパティには、対応するキーの情報が保持されます。

以下に例を示します。

Class Definition

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

Property Data As %XML.Security.EncryptedData (XMLNAME="EncryptedData");

Property Key As %XML.Security.EncryptedKey (XMLNAME="EncryptedKey");

Parameter NAMESPACE = "http://www.w3.org/2001/04/xmlenc#";

//methods

}
```

9.2.3 暗号化された XML ドキュメントの生成

暗号化されたドキュメントを生成して記述する手順は次のとおりです。

1. XML ドキュメントを格納したストリームを作成します。

   このためには通常、**%XML.Writer** を使用して、XML 対応オブジェクトの出力をストリームに書き込みます。

2. 暗号化されたドキュメントの送信先となるエンティティの InterSystems IRIS 資格情報セットにアクセスする **%SYS.X509Credentials** のインスタンスを少なくとも 1 つ作成します。これには、このクラスの GetByAlias() クラス・メソッドを呼び出します。以下はその例です。

   ObjectScript

   ```
   set credset=##class(%SYS.X509Credentials).GetByAlias("recipient")
   ```

   このメソッドを実行するには、当該の資格情報セットの **OwnerList** に記載されたユーザとしてログインするか、**OwnerList** が NULL である必要があります。"プログラムによる資格情報セットの取得" も参照してください。

3. **%XML.Security.EncryptedKey** のインスタンスを少なくとも 1 つ作成します。このクラスのインスタンスを作成するには、このクラスの CreateX509() クラス・メソッドを使用します。以下はその例です。

   ObjectScript

   ```
   set enckey=##class(%XML.Security.EncryptedKey).Createx509(credset,encryptionOptions,referenceOption)
   ```

   - credset は、先ほど作成した **%SYS.X509Credentials** のインスタンスです。
   - encryptionOptions は **$$$SOAPWSIncludeNone** です (他のオプションもありますが、このシナリオには該当しません)。

     このマクロは、**%soap.inc** インクルード・ファイルで定義されているマクロです。
   - referenceOption は、暗号化された要素に対する参照の特性です。許可される値については、"X.509 証明書の参照オプション" を参照してください。

   ここで使用されるマクロは、**%soap.inc** インクルード・ファイルで定義します。

4. **%Library.ListOfObjects** のインスタンスを作成し、その Insert() メソッドを使用して、先ほど作成した **%XML.Security.EncryptedKey** のインスタンスを挿入します。

5. %New() メソッドを使用して、**%XML.Security.EncryptedData** のインスタンスを作成します。以下はその例です。

   ObjectScript

   ```
   set encdata=##class(%XML.Security.EncryptedData).%New()
   ```

6. **%XML.Security.EncryptedData** の EncryptStream() インスタンス・メソッドを使用して、手順 2 で作成したストリームを暗号化します。例えば、以下のようになります。

   ObjectScript

   ```
   set status=encdata.EncryptStream(stream,encryptedKeys)
   ```

   - stream は、手順 1 で作成したストリームです。
   - encryptedKeys は、手順 4 で作成したキーのリストです。

7. コンテナ・クラスのインスタンスを作成して更新します。
   - キーのリストを、このクラスの適切なプロパティに書き込みます。

- %XML.Security.EncryptedData のインスタンスを、このクラスの適切なプロパティに書き込みます。

この詳細は、クラスによって異なります。

8. %XML.Writer を使用して、コンテナ・クラスの出力を生成します。"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、前に示したコンテナ・クラスには以下のメソッドも含まれます。

Class Member

```
ClassMethod Demo(filename = "",obj="")
{
#include %soap

    if (obj="") {
        set obj=##class(XMLEncryption.Person).GetPerson()
    }

    //create stream from this XML-enabled object
    set writer=##class(%XML.Writer).%New()
    set stream=##class(%GlobalCharacterStream).%New()
    set status=writer.OutputToStream(stream)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit }
    set status=writer.RootObject(obj)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit }
    do stream.Rewind()

    set container=..%New()              ; this is the object we will write out
    set cred=##class(%SYS.X509Credentials).GetByAlias("servercred")
    set parts=$$$SOAPWSIncludeNone
    set ref=$$$KeyInfoX509Certificate
    set key=##class(%XML.Security.EncryptedKey).CreateX509(cred,parts,ref)
    set container.Key=key         ; this detail depends on the class

    //need to create a list of keys (just one in this example)
    set keys=##class(%Collection.ListOfObj).%New()
    do keys.Insert(key)

    set encdata=##class(%XML.Security.EncryptedData).%New()

    set status=encdata.EncryptStream(stream,keys)
    set container.Data=encdata    ; this detail depends on the class

    // write output for the container
    set writer=##class(%XML.Writer).%New()
    set writer.Indent=1
    if (filename'="") {
        set status=writer.OutputToFile(filename)
        if $$$ISERR(status) {do $system.OBJ.DisplayError(status) quit}
        }
    set status=writer.RootObject(container)
    if $$$ISERR(status) {do $system.OBJ.DisplayError(status) quit}
}
```

このメソッドは、任意の XML 対応クラスの OREF を受け入れることができます。何も提供されていない場合は、既定値が使用されます。

9.3 暗号化された XML ファイルの解読

9.3.1 解読の前提条件

暗号化された XML ドキュメントを解読するには、まず以下の両方を用意する必要があります。

- InterSystems IRIS が使用するための信頼された証明書。
- 暗号化で使用されている公開鍵と秘密鍵が一致している InterSystems IRIS 資格情報セット。

詳細は、"設定およびその他の一般的なアクティビティ" を参照してください。

9.3.2 ドキュメントの解読

暗号化された XML ドキュメントを解読する手順は次のとおりです。

1. %XML.Reader のインスタンスを作成し、それを使用してドキュメントを開きます。

   "オブジェクトへの XML のインポート" を参照してください。

2. リーダの Document プロパティを取得します。これは、XML ドキュメントを DOM として含む %XML.Document のインスタンスです。

3. リーダの Correlate() メソッドを使用して、1 つまたは複数の <EncryptedKey> 要素をクラス %XML.Security.EncryptedKey に関連付けます。以下はその例です。

   ObjectScript

   ```
   do reader.Correlate("EncryptedKey","%XML.Security.EncryptedKey")
   ```

4. ドキュメントで 1 つまたは複数の <EncryptedKey> 要素を繰り返し読み取ります。このためには、インポートされたオブジェクトがあれば参照によって返す、リーダの Next() メソッドを使用します。以下はその例です。

   ```
   if 'reader.Next(.ikey,.status) {
       write !,"Unable to import key",!
       do $system.OBJ.DisplayError(status)
       quit
       }
   ```

   インポートされたオブジェクトは %XML.Security.EncryptedKey のインスタンスです。

5. %Library.ListOfObjects のインスタンスを作成し、その Insert() メソッドを使用して、ドキュメントから先ほど取得した %XML.Security.EncryptedKey のインスタンスを挿入します。

6. クラス %XML.Security.EncryptedData の ValidateDocument() メソッドを呼び出します。

   ObjectScript

   ```
   set status=##class(%XML.Security.EncryptedData).ValidateDocument(.doc,keys)
   ```

   参照によって返される最初の引数は、手順 2 で取得した DOM を変更したものです。2 番目の引数は、前の手順からのキーのリストです。

7. 必要に応じて、%XML.Writer を使用して、変更された DOM の出力を生成します。"オブジェクトからの XML 出力の記述" を参照してください。

次に、例を示します。

Class Member

```
ClassMethod DecryptDoc(filename As %String)
{
#include %soap
    set reader=##class(%XML.Reader).%New()
    set status=reader.OpenFile(filename)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit }

    set doc=reader.Document
    //get <Signature> element
    do reader.Correlate("EncryptedKey","%XML.Security.EncryptedKey")
    if 'reader.Next(.ikey,.status) {
        write !,"unable to import key",!
        do $system.OBJ.DisplayError(status)
     quit
    }
```

```
    set keys=##class(%Collection.ListOfObj).%New()
    do keys.Insert(ikey)
    // the following step returns the decrypted document
    set status=##class(%XML.Security.EncryptedData).ValidateDocument(.doc,keys)

    set writer=##class(%XML.Writer).%New()
    set writer.Indent=1
    do writer.Document(doc)
    quit $$$OK
}

}
```

10

XML ドキュメントの署名

このトピックでは、XML ドキュメントにデジタル・シグニチャを追加する方法を説明します。

Tipヒント　このネームスペースで SOAP ロギングを有効にしておくと、エラーの詳細を受信するので便利です。"InterSystems IRIS での SOAP の問題のトラブルシューティング" の "InterSystems IRIS SOAP ログ" を参照してください。

別のダイジェスト、シグニチャ、正規化メソッドの詳細は、"デジタル・シグニチャの追加" を参照してください。

10.1 デジタル署名が行われたドキュメントについて

デジタル署名が行われた XML ドキュメントには 1 つ以上の `<Signature>` 要素が含まれ、その要素それぞれがデジタル・シグニチャです。それぞれの `<Signature>` 要素によって、以下のように、ドキュメント内の特定の要素に対して署名が行われます。

- 署名されたそれぞれの要素には `Id` 属性があり、この属性は何らかの一意の値となります。以下はその例です。

  ```
  <Person xmlns="http://mynamespace" Id="123456789">
  ```

- `<Signature>` 要素には、以下のように、その `Id` を指す `<Reference>` 要素が含まれています。

  ```
  <Reference URI="#123456789">
  ```

  `<Signature>` 要素は、秘密鍵によって署名されます。この要素には、認証機関によって署名された X.509 証明書が含まれています。この署名されたドキュメントの受信者は、この認証機関を信頼していれば、その証明書を検証し、含まれている公開鍵を使用してシグニチャを検証できます。

注釈　InterSystems IRIS® データ・プラットフォームでは、署名された要素が `Id` ではなく、`ID` という名前の属性を保持するというバリエーションもサポートしています。詳細は、このトピックの最後のセクションを参照してください。

以下の例では、見やすくするために空白を追加してあります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Person xmlns="http://mynamespace" Id="123456789">
  <Name>Persephone MacMillan</Name>
  <DOB>1976-02-20</DOB>
  <s01:Signature xmlns="http://www.w3.org/2000/09/xmldsig#"
                 xmlns:s01="http://mynamespace"
                 s02:Id="Id-BC0B1674-758D-40B9-84BF-F7BAA3AA19F4"
xmlns:s02="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <SignedInfo>
```

```
      <CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      </CanonicalizationMethod>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
      </SignatureMethod>
      <Reference URI="#123456789">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature">
          </Transform>
          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml1317c14n-20010315">
          </Transform>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"></DigestMethod>
        <DigestValue>FHwW2U58bztLI4cIE/mp+nsBNZg=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>MTha3zLoj8Tg content omitted</SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509Certificate>MIICnDCCAYQCAWUwDQYJ content omitted</X509Certificate>
      </X509Data>
    </KeyInfo>
  </s01:Signature>
</Person>
```

デジタル・シグニチャを作成するには、クラス %XML.Security.Signature を使用します。これは、XML 対応のクラスのうち、適切なネームスペースにおいてこのプロジェクションが有効な <Signature> 要素になっているものです。

10.2 デジタル署名が行われた XML ドキュメントの作成

デジタル署名が行われた XML ドキュメントを作成するには、%XML.Writer を使用して、適切に定義された 1 つ以上の XML 対応オブジェクトの出力を生成します。

オブジェクトの出力を生成する前に、必要とされるシグニチャを作成してオブジェクトに記述することによって、出力先に情報を書き込めるようにしておく必要があります。

10.2.1 署名の前提条件

ドキュメントに署名するには、まず少なくとも 1 つの InterSystems IRIS 資格情報セットを作成する必要があります。InterSystems IRIS 資格情報セットとは以下の情報セットの別名であり、システム管理者のデータベースに格納されています。

- 公開鍵を含む証明書。この証明書は、ドキュメントの受信者に信頼される認証機関によって署名されている必要があります。
- 関連付けられている秘密鍵。この秘密鍵は、必要なときに InterSystems IRIS が使用しますが、送信することはありません。

  秘密鍵は署名用に必要です。
- (オプション) 秘密鍵のパスワード。この秘密鍵は、必要なときに InterSystems IRIS が使用しますが、送信することはありません。秘密鍵をロードすることも、実行時にこれを指定するとこもできます。

詳細は、"設定およびその他の一般的なアクティビティ" を参照してください。

10.2.2 XML 対応クラスの要件

XML 対応クラスには、以下のものが含まれている必要があります。

- Id 属性として投影されているプロパティ。
- タイプが %XML.Security.Signature で、<Signature> 要素として投影されている、少なくとも 1 つのプロパティ (1 つの XML ドキュメントには、複数の <Signature> 要素が含まれる場合もあります)。

以下のクラスを考えてみます。

Class Definition

```
Class XMLSignature.Simple Extends (%RegisteredObject, %XML.Adaptor)
{

Parameter NAMESPACE = "http://mynamespace";

Parameter XMLNAME = "Person";

Property Name As %String;

Property DOB As %String;

Property PersonId As %String(XMLNAME = "Id", XMLPROJECTION = "ATTRIBUTE");

Property MySig As %XML.Security.Signature(XMLNAME = "Signature");

//methods
}
```

10.2.3 シグニチャの生成と追加

デジタル署名を生成して追加するには、以下の手順を実行します。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. 適切な InterSystems IRIS 資格情報セットにアクセスする %SYS.X509Credentials のインスタンスを作成します。これには、%SYS.X509Credentials の GetByAlias() クラス・メソッドを呼び出します。

   ```
   classmethod GetByAlias(alias As %String, pwd As %String) as %SYS.X509Credentials
   ```

   - alias は、証明書のエイリアスです。
   - pwd は、秘密鍵のパスワードです。秘密鍵のパスワードは、関連付けられている秘密鍵が暗号化されていて、その秘密鍵のファイルをロードしたときにパスワードがロードされなかった場合にのみ必要です。

   このメソッドを実行するには、当該の資格情報セットの OwnerList に記載されたユーザとしてログインするか、OwnerList が NULL である必要があります。"プログラムによる資格情報セットの取得" も参照してください。
3. 指定された資格情報セットを使用する %XML.Security.Signature のインスタンスを作成します。これには、そのクラスの CreateX509() クラス・メソッドを呼び出します。

   ```
   classmethod CreateX509(credentials As %SYS.X509Credentials,            signatureOption As %Integer,
                referenceOption As %Integer) as %XML.Security.Signature
   ```

   - credentials は、先ほど作成した %SYS.X509Credentials のインスタンスです。
   - signatureOption は $$$SOAPWSIncludeNone です (他のオプションもありますが、このシナリオには該当しません)。
   - referenceOption は、署名された要素に対する参照の特性です。許可される値については、"X.509 証明書の参照オプション" を参照してください。

   ここで使用されるマクロは、%soap.inc インクルード・ファイルで定義します。
4. このシグニチャが指す Id については、Id 属性の値を取得します。

   この詳細は、XML 対応オブジェクトの定義によって異なります。

5. その Id を指す %XML.Security.Reference のインスタンスを作成します。これには、そのクラスの Create() クラス・メソッドを呼び出します。

```
ClassMethod Create(id As %String,                    algorithm As %String,
prefixList As %String)
```

id は、この参照が指す ID です。

algorithm は、以下のいずれかにする必要があります。

- $$$SOAPWSEnvelopedSignature_","_$$$SOAPWSexcc14n – 排他的な正規化には、このバージョンを使用します。
- $$$SOAPWSEnvelopedSignature – これは、上記のオプションと同等です。
- $$$SOAPWSEnvelopedSignature_","_$$$SOAPWSexcc14n – 包含的な正規化には、このバージョンを使用します。

6. シグニチャ・オブジェクトの場合は、AddReference() メソッドを呼び出して、この参照をシグニチャに追加します。

```
Method AddReference(reference As %XML.Security.Reference)
```

7. XML 対応クラスの該当するプロパティを更新して、シグニチャを含むようにします。

この詳細は、XML 対応クラスによって異なります。以下はその例です。

```
set object.MySig=signature
```

8. XML としてシリアル化された XML 対応オブジェクトを含む %XML.Document のインスタンスを作成します。

署名されたドキュメントについての情報がシグニチャに記載されている必要があるため、これは必要です。

"例 2：オブジェクトの DOM への変換" を参照してください。

注釈　このドキュメントには、空白は含まれていません。

9. シグニチャ・オブジェクトの SignDocument() メソッドを呼び出します。

```
Method SignDocument(document As %XML.Document) As %Status
```

このメソッドの引数は、先ほど作成した %XML.Document のインスタンスです。SignDocument() メソッドでは、そのインスタンスに入っている情報を使用してシグニチャ・オブジェクトを更新します。

10. %XML.Writer を使用して、オブジェクトの出力を生成します。"オブジェクトからの XML 出力の記述" を参照してください。

注釈　生成する出力では、空白 (または空白の欠落) が、このシグニチャで使用されているドキュメントに含まれているものと同じである必要があります。シグニチャにはドキュメントのダイジェストが含まれており、ライターで Indent プロパティを 1 に設定している場合には、ダイジェストとドキュメントは一致しません。

以下はその例です。

Class Member

```
Method WriteSigned(filename As %String = "")
{
#include %soap
    //create a signature object
    set cred=##class(%SYS.X509Credentials).GetByAlias("servercred")
    set parts=$$$SOAPWSIncludeNone
    set ref=$$$KeyInfoX509Certificate

    set signature=##class(%XML.Security.Signature).CreateX509(cred,parts,ref,.status)
```

```
    if $$$ISERR(status) {do $system.OBJ.DisplayError(status) quit}

    // get the Id attribute of the element we will sign;
    set refid=$this.PersonId ; this detail depends on structure of your classes

    // then create a reference to that Id within the signature object
    set algorithm=$$$SOAPWSEnvelopedSignature_","_$$$SOAPWSc14n
    set reference=##class(%XML.Security.Reference).Create(refid,algorithm)
    do signature.AddReference(reference)

    //set the MySig property so that $this has all the information needed
    //when we generate output for it
    set $this.MySig=signature ; this detail depends on structure of your classes

    //in addition to $this, we need an instance of %XML.Document
    //that contains the object serialized as XML
    set document=..GetXMLDoc($this)

    //use the serialized XML object to sign the document
    //this updates parts of the signature
    set status=signature.SignDocument(document)
    if $$$ISERR(status) {do $system.OBJ.DisplayError(status) quit}

    // write output for the object
    set writer=##class(%XML.Writer).%New()
    if (filename'="") {
        set status=writer.OutputToFile(filename)
        if $$$ISERR(status) {do $system.OBJ.DisplayError(status) quit}
    }
    do writer.RootObject($this)
}
```

前のインスタンス・メソッドでは以下の汎用クラス・メソッドを使用しており、このクラス・メソッドはどのような XML 対応オブジェクトでも使用できます。

Class Member

```
ClassMethod GetXMLDoc(object) As %XML.Document
{
    //step 1 - write object as XML to a stream
    set writer=##class(%XML.Writer).%New()
    set stream=##class(%GlobalCharacterStream).%New()
    set status=writer.OutputToStream(stream)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}
    set status=writer.RootObject(object)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}

    //step 2 - extract the %XML.Document from the stream
    set status=##class(%XML.Document).GetDocumentFromStream(stream,.document)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit $$$NULLOREF}
    quit document
}
```

10.2.3.1 バリエーション：参照に URI="" を用いたデジタル・シグニチャ

バリエーションとして、シグニチャ用の <Reference> 要素は、そのシグニチャを含む XML ドキュメントのルート・ノードを参照する URI="" を保持できます。この方法でデジタル・シグニチャを作成するには、以下を実行します。

1. 必要に応じて、%soap.inc インクルード・ファイルを組み込みます。このファイルには、使用する可能性のあるマクロが定義されています。
2. 適切な InterSystems IRIS 資格情報セットにアクセスする %SYS.X509Credentials のインスタンスを作成します。そのためには、前述の手順の説明に従って、%SYS.X509Credentials の GetByAlias() クラス・メソッドを呼び出します。
3. 指定された資格情報セットを使用する %XML.Security.Signature のインスタンスを作成します。そのためには、前述の手順の説明に従って、そのクラスの CreateX509() クラス・メソッドを呼び出します。
4. 以下のように、%XML.Security.X509Data のインスタンスを作成します。

```
    set valuetype=$$$KeyInfoX509SubjectName_","_$$$KeyInfoX509Certificate
    set x509data=##class(%XML.Security.X509Data).Create(valuetype,cred)
```

cred は、先ほど作成した %SYS.X509Credentials のインスタンスです。これらの手順により、<X509SubjectName> 要素および <X509Certificate> 要素を含む <X509Data> 要素を作成します。

5. 以下のように、<X509Data> 要素をシグニチャの <KeyInfo> 要素に加えます。

```
do signature.KeyInfo.KeyInfoClauseList.Insert(x509data)
```

signature は %XML.Security.Signature のインスタンスであり、x509data は %XML.Security.X509Data のインスタンスです。

6. 以下のように、%XML.Security.Reference のインスタンスを作成します。

```
set algorithm=$$$SOAPWSEnvelopedSignature
set reference=##class(%XML.Security.Reference).Create("",algorithm)
```

7. 手順 6 (AddReference() の呼び出し) で前述の手順を続けます。

10.3 デジタル・シグニチャの検証

デジタル署名が行われたドキュメントを受信した場合は、シグニチャを検証できます。ドキュメントの内容と一致した XML 対応クラスは不要です。

10.3.1 シグニチャの検証の前提条件

デジタル・シグニチャを検証するには、まず、署名者の信頼された証明書を InterSystems IRIS に提供する必要があります。InterSystems IRIS が署名を検証できるのは、中間証明書 (もしあれば) を含め、署名者独自の証明書から、InterSystems IRIS が信頼する認証機関 (CA) の自己署名証明書までの署名者の証明書チェーンを検証できる場合です。

詳細は、"設定およびその他の一般的なアクティビティ" を参照してください。

10.3.2 シグニチャの検証

デジタル署名が行われた XML ドキュメントのシグニチャを検証するには、以下の手順を実行します。

1. %XML.Reader のインスタンスを作成し、それを使用してドキュメントを開きます。

   このクラスの詳細は、"オブジェクトへの XML のインポート" を参照してください。

2. リーダの Document プロパティを取得します。これは、XML ドキュメントを DOM として含む %XML.Document のインスタンスです。

3. リーダの Correlate() メソッドを使用して、1 つまたは複数の <Signature> 要素をクラス %XML.Security.Signature に関連付けます。以下はその例です。

   ObjectScript

```
 do reader.Correlate("Signature","%XML.Security.Signature")
```

4. ドキュメントで 1 つまたは複数の <Signature> 要素を繰り返し読み取ります。このためには、インポートされたオブジェクトがあれば参照によって返す、リーダの Next() メソッドを使用します。以下はその例です。

```
if 'reader.Next(.isig,.status) {
    write !,"Unable to import signature",!
    do $system.OBJ.DisplayError(status)
    quit
    }
```

インポートされたオブジェクトは %XML.Security.Signature のインスタンスです。

5. インポートされたシグニチャの ValidateDocument() メソッドを呼び出します。このメソッドに対する引数は、以前に取得した %XML.Document のインスタンスである必要があります。

   ObjectScript

   ```
   set status=isig.ValidateDocument(document)
   ```

   その他の検証オプションについては、クラス・リファレンスで、%XML.Security.Signature のこのメソッドを参照してください。

例えば、以下のようになります。

Class Member

```
ClassMethod ValidateDoc(filename As %String)
{
    set reader=##class(%XML.Reader).%New()
    set status=reader.OpenFile(filename)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit }

    set document=reader.Document
    //get <Signature> element
    //assumes there is only one signature
    do reader.Correlate("Signature","%XML.Security.Signature")
    if 'reader.Next(.isig,.status) {
        write !,"Unable to import signature",!
        do $system.OBJ.DisplayError(status)
        quit
    }
    set status=isig.ValidateDocument(document)
    if $$$ISERR(status) {do $System.Status.DisplayError(status) quit }
}
```

10.4 バリエーション :ID を参照するデジタル・シグニチャ

通常、<Signature> 要素には、ドキュメント内の別の場所にある一意の Id を指している <Reference> 要素が含まれます。InterSystems IRIS では、<Reference> 要素が Id ではなく、ID という名前の属性を指すというバリエーションもサポートしています。このバリエーションでは、ドキュメントの署名およびドキュメントの検証に余分な作業が必要になります。

ドキュメントをデジタル署名するには、"デジタル署名が行われた XML ドキュメントの作成" の手順に従いますが、以下の点が異なります。

- XML 対応クラスの場合、Id 属性ではなく、ID 属性として投影されたプロパティを含めます。
- シグニチャを生成して追加するときに、%XML.Document インスタンスの AddIDs() メソッドを呼び出します。これは、シリアル化した XML ドキュメントの取得後、シグニチャ・オブジェクトの SignDocument() メソッドを呼び出す前に行います。例えば、以下のようになります。

ObjectScript

```
//set the MySig property so that $this has all the information needed
//when we generate output for it
set $this.MySig=signature    ; this detail depends on structure of your classes

//in addition to $this, we need an instance of %XML.Document
//that contains the object serialized as XML
set document=..GetXMLDoc($this)

//***** added step when signature references an ID attribute *****
do document.AddIDs()

//use the serialized XML object to sign the document
//this updates parts of the signature
set status=signature.SignDocument(document)
if $$$ISERR(status) {do $system.OBJ.DisplayError(status) quit}
```

- ドキュメントを検証する場合は、Correlate() を呼び出す直前に以下の手順を含めます。

  1. %XML.Document インスタンスの AddIDs() メソッドを呼び出します。
  2. XML リーダの Rewind() メソッドを呼び出します。

  例えば、以下のようになります。

ObjectScript

```
set document=reader.Document

//added steps when signature references an ID attribute
do document.AddIDs()
do reader.Rewind()

//get <Signature> element
do reader.Correlate("Signature","%XML.Security.Signature")
```

11

XPath 式の評価

XPath (XML Path Language) は、XML ドキュメントからデータを取得するための XML ベースの式言語です。任意の XML ドキュメントを指定して、%XML.XPATH.Document クラスを使用し、簡単に XPath 式を評価できます。

注釈　使用する任意の XML ドキュメントの XML 宣言にはそのドキュメントの文字エンコードを明記する必要があり、明記しておけば、ドキュメントは宣言どおりにエンコードされるようになります。文字エンコードが宣言されていない場合、InterSystems IRIS® データ・プラットフォームでは、"入出力の文字エンコード" で説明されている既定値が使用されます。これらの既定値が正しくない場合は、XML 宣言を修正して、実際に使用されている文字セットを指定するようにします。

11.1 XPath 式の評価の概要

InterSystems IRIS XML サポートを使用して、任意の XML ドキュメントを使用して XPath 式を評価するには、次の手順を実行します。

1. %XML.XPATH.Document のインスタンスを作成します。そのためには、CreateFromFile()、CreateFromStream()、または CreateFromString() のいずれかのクラス・メソッドを使用します。これらのどのメソッドを使用する場合でも、入力する XML ドキュメントを最初の引数として指定し、%XML.XPATH.Document のインスタンスを出力パラメータとして受け取ります。

   この手順によって、組み込み XSLT プロセッサを使用して XML ドキュメントが解析されます。

2. %XML.XPATH.Document のインスタンスの EvaluateExpression() メソッドを使用します。このメソッドには、評価するノード・コンテキストと式を指定します。

   - ノード・コンテキストは、式を評価するコンテキストを指定します。XPath 構文を使用してその目的のノードへのパスを表します。以下はその例です。

     ```
     "/staff/doc"
     ```

   - 評価する式にも XPath 構文を使用します。以下はその例です。

     ```
     "name[@last='Marston']"
     ```

   出力パラメータとして (3 番目の引数として) 結果を受け取ります。以下に例を示します。

```
set status=##class(%XML.XPATH.Document).CreateFromFile("c:/test/myfile.xml,.mydoc)
//check status before proceeding
set status=mydoc.EvaluateExpression("/staff/doc","name[@last='Marston']",.myresults)
//check status before proceeding
write !, myresults
```

以下のセクションでは、これらのすべてのメソッドの詳細と例を示します。

注釈　大規模なドキュメント・セットの繰り返し処理を行いつつ、それら各自の XPath 式を評価している場合、次のドキュメントを開く前に、NULL に等しいドキュメントに対する処理実行時に OREF を設定することを推奨します。これはサードパーティ・ソフトウェアの制限を回避して機能します。この制限は、大量のドキュメントをループで処理する場合において、CPU 使用量が若干増加する原因になります。

11.2 XPATH ドキュメント作成時の引数リスト

%XML.XPATH.Document のインスタンスを作成するには、そのクラスの CreateFromFile()、CreateFromStream()、または CreateFromString() のいずれかのクラス・メソッドを使用します。これらのクラス・メソッドの全引数のリストを、以下に順番に示します。

1. pSource、pStream、または pString ― ソース・ドキュメント。
   - CreateFromFile() の場合は、この引数はファイル名です。
   - CreateFromStream() の場合は、この引数はバイナリ・ストリームです。
   - CreateFromString() の場合は、この引数は文字列です。
2. pDocument ― 出力パラメータとして返される結果。これは、%XML.XPATH.Document のインスタンスです。
3. pResolver ― ソースの解析時に使用される、オプションのエンティティ・リゾルバ。詳細は、"SAX パーサの使用法のカスタマイズ" の "カスタム・エンティティの解析実行" を参照してください。
4. pErrorHandler ― オプションのカスタム・エラー・ハンドラ。次のトピックの "エラー処理のカスタマイズ" を参照してください。カスタム・エラー・ハンドラを指定しない場合は、メソッドは %XML.XSLT.ErrorHandler の新しいインスタンスを使用します。
5. pFlags ― SAX パーサによる検証と処理を制御するオプションのフラグ。詳細は、"SAX パーサの使用法のカスタマイズ" の "パーサ・フラグの設定" を参照してください。
6. pSchemaSpec ― ドキュメント・ソースの検証の基準になる、オプションのスキーマ仕様。この引数は、ネームスペース/URL のペアをコンマで区切って指定したリストを含む文字列です。

   ```
   "namespace URL,namespace URL"
   ```

   ここで、namespace はスキーマに使用する XML ネームスペースで、URL はスキーマ・ドキュメントの位置を表す URL です。ネームスペースと URL の値の間は、1 つの空白文字で区切られています。
7. pPrefixMappings ― オプションの接頭語のマッピング文字列。詳細は、サブセクション "既定のネームスペースの接頭語のマッピングの追加" を参照してください。

CreateFromFile()、CreateFromStream()、および CreateFromString() の各メソッドはステータスを返しますが、このステータスをチェックする必要があります。以下はその例です。

ObjectScript

```
Set tSC=##class(%XML.XPATH.Document).CreateFromFile("c:\sample.xml",.tDocument)
If $$$ISERR(tSC) Do $System.OBJ.DisplayError(tSC)
```

11.2.1 既定のネームスペースの接頭語のマッピングの追加

XML ドキュメントで既定のネームスペースを使用すると、XPath に問題が発生します。以下の例を考えてみます。

XML

```
<?xml version="1.0"?>
<staff xmlns="http://www.staff.org">
  <doc type="consultant">
    <name first="David" last="Marston">Mr. Marston</name>
    <name first="David" last="Bertoni">Mr. Bertoni</name>
    <name first="Donald" last="Leslie">Mr. Leslie</name>
    <name first="Emily" last="Farmer">Ms. Farmer</name>
  </doc>
</staff>
```

この場合、<staff> 要素はネームスペースに属しますが、ネームスペース接頭語は持ちません。XPath では <doc> 要素に簡単にアクセスすることができません。

既定のネームスペースがあるノードに簡単にアクセスできるように、%XML.XPATH.Document クラスでは、以下の 2 つの方法で使用できる接頭語のマッピング機能を提供しています。

- %XML.XPATH.Document のインスタンスの PrefixMappings プロパティを設定できます。このプロパティは、XPath 式が完全なネームスペース URI ではなく接頭語を使用できるように、ソース・ドキュメント内のそれぞれの既定のネームスペースに一意の接頭語を提供することを目的としています。

  PrefixMappings プロパティはコンマ区切りのリストで構成される文字列です。各リスト項目は、接頭語の後にスペース文字が 1 つ、その後にネームスペース URI の順で記述されます。

- CreateFromFile()、CreateFromStream()、または CreateFromString() を呼び出すときには、pPrefixMappings 引数を指定できます。この文字列は前述の形式と同じである必要があります。

  "XPATH ドキュメント作成時の引数リスト" を参照してください。

その後、他のネームスペースの接頭語を使用する場合と同じ方法で、これらの接頭語を使用します。

例えば、前述の XML を %XML.XPATH.Document のインスタンスに読み込む場合に、以下のように PrefixMappings を設定したとします。

```
"s http://www.staff.org"
```

この場合、"/s:staff/s:doc" を使用して <doc> 要素にアクセスできます。

インスタンス・メソッド GetPrefix() を使用することで、ドキュメントで指定されたパスに対して指定した接頭語を取得できます。

11.3 XPath 式の評価

XPath 式を評価するには、%XML.XPATH.Document のインスタンスの EvaluateExpression() メソッドを使用します。このメソッドでは、次の順序で以下の引数を指定します。

1. pContext ― 式を評価するコンテキストを指定するノード・コンテキスト。その目的のノードへのパスの XPath 構文を含む文字列を指定します。以下はその例です。

   ```
   "/staff/doc"
   ```

2. pExpression ― 特定の結果を選択する述語。目的の XPath 構文を含む文字列を指定します。以下はその例です。

   ```
   "name[@last='Marston']"
   ```

   注釈　他の技術では、述語をノード・パスの末尾に連結するのが一般的です。%XML.XPATH.Document クラスではこの構文をサポートしていません。これは、基本となる XSLT プロセッサは個別の引数としてノード・コンテキストと述語を必要とするためです。

3. pResults ― 出力パラメータとして返される結果。結果の詳細は、"XPath の結果の使用法" を参照してください。

EvaluateExpression() メソッドではステータスが返されます。これはチェックする必要があります。以下はその例です。

ObjectScript

```
Set tSC=tDoc.EvaluateExpression("/staff/doc","name[@last='Smith']",.tRes)
If $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC)}
```

11.4 XPath の結果の使用法

XPath 式では、XML ドキュメントのサブツリー、複数のサブツリー、またはスカラ結果を返すことができます。%XML.XPATH.Document の EvaluateExpression() メソッドはこれらのすべてのケースを処理するように設計されています。具体的には、結果のリストを返します。リストの各項目には、以下の値のいずれかを持つ Type プロパティがあります。

- $$$XPATHDOM ― この項目に XML ドキュメントのサブツリーが含まれることを示します。この項目は、サブツリーをナビゲートおよび検証するためのメソッドを提供する %XML.XPATH.DOMResult のインスタンスです。詳細は、"XML サブツリーの検証" を参照してください。
- $$$XPATHVALUE ― この項目が単一のスカラ結果であることを示します。この項目は、%XML.XPATH.ValueResult のインスタンスです。詳細は、"スカラ結果の検証" を参照してください。

これらのマクロは、%occXSLT.inc インクルード・ファイルで定義されています。

以下のサブセクションでは、これらのクラスの詳細と、使用する可能性のある全般的アプローチの概要と例について説明します。

11.4.1 XML サブツリーの検証

このセクションでは、%XML.XPATH.DOMResult によって表される XML サブツリーのナビゲート方法と、そのサブツリーでの現在位置についての情報を取得する方法を説明します。

11.4.1.1 サブツリーのナビゲート

%XML.XPATH.DOMResult のインスタンスをナビゲートするために、インスタンスの Read()、MoveToAttributeIndex()、MoveToAttributeName()、MoveToElement()、および Rewind() のメソッドを使用できます。

ドキュメント内の次のノードに進むには、Read() メソッドを使用します。Read() メソッドは、前方のノードが存在しなくなるまで (つまり、ドキュメントの末尾に達するまで)、True の値を返します。

要素に移動すると、その要素に属性がある場合は、以下のメソッドを使用してその属性に移動できます。

- MoveToAttributeIndex() メソッドを使用して、インデックス (要素内の属性の通常の位置) により特定の属性に移動します。このメソッドの引数は、属性のインデックス番号のみです。AttributeCount プロパティを使用して、指定された要素内の属性の数を知ることができます。
- MoveToAttributeName() メソッドを使用して、名前により特定の属性に移動します。このメソッドは、属性名および (オプションで) ネームスペース URI の 2 つの引数をとります。

現在の要素に対する属性を終了したら、Read() などの検索メソッドを実行することで、ドキュメント内の次の要素に移動できます。また、MoveToElement() メソッドを実行して、現在の属性を含む要素に戻ることもできます。

ここで説明するメソッドは、Rewind() メソッドを除き、すべてドキュメント内を前進します。Rewind() メソッドは、ドキュメントの最初に移動し、すべてのプロパティをリセットします。

11.4.1.2 ノードのプロパティ

Type プロパティに加え、%XML.XPATH.DOMResult の以下のプロパティも現在位置についての情報を提供します。

AttributeCount

現在のノードが要素の場合、このプロパティは、要素の属性の番号を示します。

EOF

リーダがソース・ドキュメントの末尾に到達した場合は True、それ以外の場合は False です。

HasAttributes

現在のノードが要素の場合、その要素に属性があれば、このプロパティは True です (属性がなければ False です)。現在のノードが属性の場合、このプロパティは True です。

その他のタイプのノードでは、このプロパティは False になります。

HasValue

現在のノードが値を持つタイプのノードの場合 (その値が Null であっても)、True です。それ以外の場合、このプロパティは False です。

LocalName

attribute または element のタイプのノードの場合、これは現在の要素または属性の名前からネームスペースの接頭語を除いたものとなります。その他すべてのタイプのノードでは、このプロパティは Null になります。

Name

ノードのタイプに応じた、現在のノードの完全修飾名です。

NodeType

現在のノードのタイプであり、その値は、`attribute`、`chars`、`cdata`、`comment`、`document`、`documentfragment`、`documenttype`、`element`、`entity`、`entityreference`、`notation`、または `processinginstruction` のいずれかです。

Path

要素のタイプのノードの場合、これは要素へのパスとなります。その他すべてのタイプのノードでは、このプロパティは Null になります。

ReadState

全体の読み取り状態を示します。以下のいずれかの状態となります。

- `"initial"` は、Read() メソッドがまだ呼び出されていないことを示します。
- `"cursoractive"` は、Read() メソッドが少なくとも 1 回呼び出されたことを示します。
- `"eof"` は、ファイルの末尾に達したことを示します。

Uri

現在のノードの URI。返される値は、ノードのタイプによって異なります。

値

ノードのタイプに応じた、現在のノードの値（存在する場合）です。値のサイズが 32 KB より小さい場合、これは文字列です。それ以外の場合、文字ストリームです。詳細は、"スカラ結果の検証" を参照してください。

11.4.2 スカラ結果の検証

このセクションでは、%XML.XPATH.ValueResult クラスによって表される XPath 結果の使用法を説明します。このクラスは、Type プロパティ以外に Value プロパティも提供します。

値の長さが 32 KB より大きい場合、自動的にストリーム・オブジェクトに格納されることに注意してください。受け取る結果の種類が確実にわかっていない限り、Value がストリーム・オブジェクトであるかどうかを確認します。このためには、$IsObject 関数を使用できます（つまり、この値がオブジェクトの場合、それは、可能な唯一の種類のオブジェクトであるため、ストリーム・オブジェクトです）。

以下の部分は、テストの例を示しています。

ObjectScript

```
// Value can be a stream if result is greater than 32 KB in length
Set tValue=tResult.Value

If $IsObject(tValue){
    Write ! Do tValue.OutputToDevice()
} else {
    Write tValue
}
```

11.4.3 一般的なアプローチ

XPath 式を評価するときに受け取る結果の種類が確実にわかっていない限り、最も一般的で可能性のあるケースを処理するためのコードを記述します。コードの構成例を以下に示します。

1. 返される結果のリストにある要素の数を検索します。このリストの繰り返し処理を行います。
2. 各リスト項目について、Type プロパティを確認します。
   - Type が $$$XPATHDOM の場合、%XML.XPATH.DOMResult クラスのメソッドを使用してこの XML サブツリーをナビゲートし、検証します。
   - Type が $$$XPATHVALUE の場合、Value プロパティがストリーム・オブジェクトかどうかを確認します。ストリーム・オブジェクトの場合は、通常のストリーム・インタフェースを使用してデータにアクセスします。それ以外の場合は、Value プロパティは文字列です。

使用例は、%XML.XPATH.Document の ExampleDisplayResults() クラス・メソッドを参照してください。このメソッドは、前のリストで説明した方法で結果を検証し、その後、結果をターミナルに書き込みます。以下のセクションでは、このメソッドからの出力形式を示します。

11.5 例

このセクションの例は、以下の XML ドキュメントに対して XPath 式を評価します。

XML

```
<?xml version="1.0"?>
<staff>
  <doc type="consultant">
    <name first="David" last="Marston">Mr. Marston</name>
    <name first="David" last="Bertoni">Mr. Bertoni</name>
    <name first="Donald" last="Leslie">Mr. Leslie</name>
    <name first="Emily" last="Farmer">Ms. Farmer</name>
  </doc>
  <doc type="GP">
    <name first="Myriam" last="Midy">Ms. Midy</name>
    <name first="Paul" last="Dick">Mr. Dick</name>
    <name first="Scott" last="Boag">Mr. Boag</name>
    <name first="Shane" last="Curcuru">Mr. Curcuru</name>
    <name first="Joseph" last="Kesselman">Mr. Kesselman</name>
    <name first="Stephen" last="Auriemma">Mr. Auriemma</name>
  </doc>
</staff>
```

これらの例は、%XML.XPATH.Document クラスに含まれる大規模な例を変更したものです。詳細を確認するには、ソース・コードを参照してください。

11.5.1 サブツリー結果を持つ XPath 式の評価

以下のクラス・メソッドは XML ファイルを読み取り、XML サブツリーを返す XPath 式を評価します。

Class Member

```
/// Evaluates an XPath expression that returns a DOM Result
ClassMethod Example1()
{
    Set tSC=$$$OK
    do {

    Set tSC=##class(%XML.XPATH.Document).CreateFromFile(filename,.tDoc)
    If $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC) Quit}

    Set context="/staff/doc"
    Set expr="name[@last='Marston']"
    Set tSC=tDoc.EvaluateExpression(context,expr,.tRes)
    If $$$ISERR(tSC) Quit

        Do ##class(%XML.XPATH.Document).ExampleDisplayResults(tRes)

    } while (0)
    If $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC)}
    Quit
}
```

この例では、last 属性が Marston である <name> 要素を持つすべてのノードが選択されます。この式は、<staff> 要素の <doc> ノードで評価されます。

この例は %XML.XPATH.Document の ExampleDisplayResults() クラス・メソッドを使用することに注意してください。

前述の XML ファイルを入力として指定して Example1() メソッドを実行すると、以下の出力が表示されます。

```
XPATH DOM
element: name
         attribute: first Value: David
         attribute: last  Value: Marston

chars : #text Value: Mr. Marston
```

11.5.2 スカラ結果を持つ XPath 式の評価

以下のクラス・メソッドは XML ファイルを読み取り、スカラ結果を返す XPath 式を評価します。

Class Member

```
/// Evaluates an XPath expression that returns a VALUE Result
ClassMethod Example2()
{
    Set tSC=$$$OK
    do {

    Set tSC=##class(%XML.XPATH.Document).CreateFromFile(filename,.tDoc)
    If $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC) Quit}

    Set tSC=tDoc.EvaluateExpression("/staff","count(doc)",.tRes)
        If $$$ISERR(tSC) Quit

        Do ##class(%XML.XPATH.Document).ExampleDisplayResults(tRes)

    } while (0)
    If $$$ISERR(tSC) {Do $System.OBJ.DisplayError(tSC)}
    Quit
}
```

この例では、<doc> サブノードがカウントされます。この式は、<staff> 要素で評価されます。

前述の XML ファイルを入力として指定して Example2() メソッドを実行すると、以下の出力が表示されます。

```
XPATH VALUE
2
```

12
XSLT 変換の実行

XSLT (Extensible Stylesheet Language Transformations) は XML ベースの言語であり、これを使用して指定の XML ドキュメントを別の XML ドキュメントまたは人間が読める別の形式のドキュメントに変換する方法を記述します。**%XML.XSLT** および **%XML.XSLT2** パッケージ内のクラスを使用すると、XSLT 1.0 および 2.0 の変換を実行できます。

注釈　使用する任意の XML ドキュメントの XML 宣言にはそのドキュメントの文字エンコードを明記する必要があり、明記しておけば、ドキュメントは宣言どおりにエンコードされるようになります。文字エンコードが宣言されていない場合、InterSystems IRIS® データ・プラットフォームでは、"入出力の文字エンコード" で説明されている既定値が使用されます。これらの既定値が正しくない場合は、XML 宣言を修正して、実際に使用されている文字セットを指定するようにします。

12.1 InterSystems IRIS における XSLT 変換の実行の概要

InterSystems IRIS は、それぞれ独自の API を備えた以下の 2 つの XSLT プロセッサを提供します。

- Xalan プロセッサは XSLT 1.0 をサポートします。**%XML.XSLT** パッケージは、このプロセッサの API を提供します。
- Saxon プロセッサは XSLT 2.0 をサポートします。**%XML.XSLT2** パッケージは、このプロセッサの API を提供します。

  %XML.XSLT2 の API は、XSLT 2.0 ゲートウェイへの接続を介して Saxon に要求を送信します。ゲートウェイは複数の接続を可能にします。つまり、それぞれ独自の一連のコンパイル済みスタイル・シートを備えた 2 つの異なる InterSystems IRIS プロセスをゲートウェイに接続して、複数の変換要求を同時に送信したりすることができます。

  Saxon プロセッサでは、コンパイル済みスタイル・シートと isc:evaluate キャッシュは接続固有です。つまり、独自の接続を管理して、いずれかの機能を利用する必要があります。接続を開いてコンパイル済みスタイル・シートを作成した場合、または isc:evaluate キャッシュに生成する変換を評価した場合、その接続で評価されるその他の変換は、コンパイル済みスタイル・シートおよび isc:evaluate キャッシュ・エントリにアクセスできるようになります。新しい接続を開いた場合、その他の接続 (およびそれらのコンパイル済みスタイル・シートやキャッシュ) は無視されます。

2 つのプロセッサの API は類似していますが、**%XML.XSLT2** のメソッドでは追加の引数を用いて、使用するゲートウェイ接続を指定します。

XSLT 変換を実行するには、以下の手順に従います。

1. Saxon プロセッサを使用する場合、次のセクションで説明するように、XSLT ゲートウェイ・サーバを構成します。または、既定の構成を使用します。

   Xalan プロセッサを使用する場合、ゲートウェイは必要ありません。

   必要に応じて、システムは自動的にゲートウェイを起動します。ゲートウェイは手動で起動することもできます。

2. Saxon プロセッサを使用する場合、必要に応じて、XSLT ゲートウェイへの単一接続を示す %Net.Remote.Gateway のインスタンスを作成します。

   Saxon プロセッサを使用する際、コンパイル済みスタイル・シートや isc:evaluate キャッシュを利用するには、この手順を実行する必要があります。

3. 必要に応じて、コンパイル済みスタイル・シートを作成して、メモリにロードします。"コンパイル済みスタイル・シートの作成" を参照してください。Saxon プロセッサを使用する場合、コンパイル済みスタイル・シートの作成時に gateway 引数を必ず指定してください。

   この手順は同じスタイル・シートを繰り返して使用する場合に役立ちます。ただし、この手順ではメモリも消費します。コンパイル済みスタイル・シートは不要になったら必ず削除してください。

4. 該当する API の変換メソッドのいずれかを呼び出します。Saxon プロセッサを使用する場合、変換メソッドを呼び出す際、必要に応じて gateway 引数を指定します。

   "XSLT 変換の実行" を参照してください。

5. 必要に応じて、他の変換メソッドを呼び出します。Saxon プロセッサを使用する場合、変換メソッドを呼び出す際、必要に応じて gateway 引数を指定します。これにより、同じ接続を使用して他の変換を評価できます。この変換は、この接続に関連付けられているすべてのコンパイル済みスタイル・シートおよび isc:evaluate キャッシュ・エントリにアクセスできるようになります。新しい接続を開いた場合、その他の接続 (およびそれらのコンパイル済みスタイル・シートやキャッシュ) は無視されます。

スタジオには、XSLT 変換のテストに使用できるウィザードも用意されています。

12.2 XSLT 2.0 ゲートウェイの開始、停止、および構成

Saxon プロセッサを (XSLT 2.0 変換を実行するために) 使用する場合、InterSystems IRIS は、XSLT 2.0 ゲートウェイ (%XSLT Server) を自動的に起動し、使用します。このゲートウェイは、他の外部サーバ接続と同様に構成されます。詳細は、"外部サーバ接続の管理" を参照してください。XSLT 2.0 ゲートウェイは、%XSLT Server としてリストされます。

12.3 XSLT ゲートウェイ・サーバ接続の再使用 (XSLT 2.0)

Saxon プロセッサを使用する場合、InterSystems IRIS は、構成済みの XSLT 2.0 ゲートウェイを使用します。このゲートウェイと通信するために、InterSystems IRIS は XSLT ゲートウェイ接続 (%Net.Remote.Gateway のインスタンス) を内部で作成します。既定では、システムは接続を作成し、変換でそれを使用した後、その接続を破棄します。新しい接続を開くと、それに伴ってオーバーヘッドが発生するため、単一接続を維持して複数の変換を行うことで最高のパフォーマンスが実現されます。また、コンパイル済みスタイル・シートや isc:evaluate キャッシュを利用するには、独自の接続を維持する必要があります。

XSLT ゲートウェイ接続を再使用するには、以下の手順を実行します。

1. %XML.XSLT2.Transformer の StartGateway() メソッドを呼び出します。

   ObjectScript

   ```
   set status=##class(%XML.XSLT2.Transformer).StartGateway(.gateway)
   ```

   このメソッドは、XSLT 2.0 ゲートウェイを開始し (まだ実行されていない場合)、出力として %Net.Remote.Gateway のインスタンスを返します。このメソッドはステータスも返します。

   %Net.Remote.Gateway のインスタンスは、ゲートウェイへの接続を示します。

StartGateway() には、オプションの 2 つ目の引数 useSharedMemory があります。この引数が true (既定値) の場合、可能であれば、localhost または 127.0.0.1 への接続に共有メモリが使用されます。接続に TCP/IP のみを強制的に使用するには、この引数を false に設定します。

2. 前の手順で返されたステータスを確認します。

```
ObjectScript

 if $$$ISERR(status) {
     quit
 }
```

3. 任意のコンパイル済みスタイル・シートを作成します。これを行う際、手順 1 で作成した %Net.Remote.Gateway のインスタンスのインスタンスとして gateway 引数を指定します。
4. 必要に応じて、%XML.XSLT2.Transformer の変換メソッド (TransformFile()、TransformFileWithCompiledXSL()、TransformStream()、および TransformStreamWithCompiledXSL()) を呼び出します。これを行う際、手順 1 で作成した %Net.Remote.Gateway のインスタンスとして gateway 引数を指定します。
5. 指定されたコンパイル済みスタイル・シートが不要になった場合、%XML.XSLT2.CompiledStyleSheet の ReleaseFromServer() メソッドを呼び出します。

```
ObjectScript

 Set status=##class(%XML.XSLT2.CompiledStyleSheet).ReleaseFromServer(compiledStyleSheet,,gateway)
```

重要　　コンパイル済みスタイル・シートが不要になった場合、必ずこのメソッドを使用してください。

6. XSLT ゲートウェイ接続が不要になった場合、%XML.XSLT2.Transformer の StopGateway() メソッドを呼び出し、引数としてゲートウェイ接続を渡します。

```
ObjectScript

 set status=##class(%XML.XSLT2.Transformer).StopGateway(gateway)
```

このメソッドは接続を破棄し、現在のデバイスをリセットします。これは XSLT 2.0 ゲートウェイを停止しません。

重要　　接続が不要になった場合、必ずこのメソッドを使用してください。

12.4 XSLT 2.0 ゲートウェイ・サーバ接続のトラブルシューティング

XSLT 2.0 ゲートウェイが開いている間に、InterSystems IRIS とゲートウェイ・サーバの間の接続が無効になることがあります。例えば、ネットワーク・エラーが発生した場合や、InterSystems IRIS がゲートウェイ・サーバに接続した後にこのサーバが再起動された場合、接続が正しく閉じられないことがあります。その結果、エラーが発生する場合があります。

XSLT ゲートウェイ接続オブジェクトの %LostConnectionCleanup() メソッドと %Reconnect メソッドを連続して呼び出すことによって、InterSystems IRIS のゲートウェイ・サーバへの再接続を試みることができます。詳細は、XSLT ゲートウェイ接続オブジェクトの継承元のクラスである "%Net.Remote.Gateway" を参照してください。

切断時におけるゲートウェイ・サーバへの再接続プロセスを自動化する場合は、ゲートウェイ接続オブジェクトの AttemptReconnect プロパティを true に設定します。

12.5 コンパイル済みスタイル・シートの作成

同じスタイル・シートを繰り返し使用する場合は、スタイル・シートをコンパイルして速度を向上させることができます。この手順はメモリを消費することに注意してください。コンパイル済みスタイル・シートは不要になったら必ず削除してください。

コンパイル済みスタイル・シートを作成するには、以下の手順を実行します。

- Xalan プロセッサ (XSLT 1.0 用) を使用する場合、%XML.XSLT.CompiledStyleSheet の以下のクラス・メソッドのいずれかを使用します。
  - CreateFromFile()
  - CreateFromStream()
- Saxon プロセッサ (XSLT 2.0 用) を使用する場合、%XML.XSLT2.CompiledStyleSheet の以下のクラス・メソッドのいずれかを使用します。
  - CreateFromFile()
  - CreateFromStream()

  また、XSLT ゲートウェイ接続を作成する必要があります。"XSLT ゲートウェイ・サーバ接続の再使用 (XSLT 2.0)" を参照してください。

これらのメソッドすべてについて、完全な引数のリストを以下に順番に示します。

1. source ー スタイル・シート。

   CreateFromFile() の場合は、この引数はファイル名です。CreateFromStream() の場合は、この引数はストリームです。

2. compiledStyleSheet ー 出力パラメータとして返される、コンパイル済みスタイル・シート。

   これは、スタイル・シート・クラス (場合に応じて、%XML.XSLT.CompiledStyleSheet または %XML.XSLT2.CompiledStyleSheet) のインスタンスです。

3. errorHandler ー スタイル・シートのコンパイル時に使用するオプションのカスタム・エラー・ハンドラ。"エラー処理のカスタマイズ" を参照してください。

   これは、どちらのクラスのメソッドの場合も、%XML.XSLT.ErrorHandler のインスタンスになります。

4. (%XML.XSLT2.CompiledStyleSheet のみ) gateway ー %Net.Remote.Gateway のインスタンス。"XSLT ゲートウェイ・サーバ接続の再使用 (XSLT 2.0)" を参照してください。

CreateFromFile() メソッドと CreateFromStream() メソッドではステータスが返されます。これはチェックする必要があります。

以下はその例です。

ObjectScript

```
//set tXSL equal to the OREF of a suitable stream
Set tSC=##class(%XML.XSLT.CompiledStyleSheet).CreateFromStream(tXSL,.tCompiledStyleSheet)
If $$$ISERR(tSC) Quit
```

12.6 XSLT 変換の実行

XSLT 変換を実行するには、以下の手順に従います。

- Xalan プロセッサ (XSLT 1.0 用) を使用する場合、%XML.XSLT.Transformer の以下のクラス・メソッドのいずれかを使用します。
  - TransformFile() ー XSLT スタイル・シートを指定して、ファイルを変換します。
  - TransformFileWithCompiledXSL() ーコンパイル済み XSLT スタイル・シートを指定して、ファイルを変換します。
  - TransformStream() ー XSLT スタイル・シートを指定して、ストリームを変換します。
  - TransformStreamWithCompiledXSL() ー コンパイル済み XSLT スタイル・シートを指定して、ストリームを変換します。
  - TransformStringWithCompiledXSL() ー コンパイル済み XSLT スタイル・シートを指定して、文字列を変換します。

  これらのメソッドには、コンパイル済みスタイル・シートとして、%XML.XSLT.CompiledStyleSheet のインスタンスを使用します。"コンパイル済みスタイル・シートの作成" を参照してください。
- Saxon プロセッサ (XSLT 2.0 用) を使用する場合、%XML.XSLT2.Transformer の以下のクラス・メソッドのいずれかを使用します。
  - TransformFile() ー XSLT スタイル・シートを指定して、ファイルを変換します。
  - TransformFileWithCompiledXSL() ーコンパイル済み XSLT スタイル・シートを指定して、ファイルを変換します。
  - TransformStream() ー XSLT スタイル・シートを指定して、ストリームを変換します。
  - TransformStreamWithCompiledXSL() ー コンパイル済み XSLT スタイル・シートを指定して、ストリームを変換します。

  これらのメソッドには、コンパイル済みスタイル・シートとして、%XML.XSLT2.CompiledStyleSheet のインスタンスを使用します。"コンパイル済みスタイル・シートの作成" を参照してください。

これらのメソッドは、同様のシグニチャを持ちます。以下に、これらのメソッドの引数リストを順番に記載します。

1. pSource ー 変換されるソース XML。このリストの後に示すテーブルを参照してください。
2. pXSL ー スタイル・シートまたはコンパイル済みスタイル・シート。このリストの後に示すテーブルを参照してください。
3. pOutput ー 出力パラメータとして返される結果の XML。このリストの後に示すテーブルを参照してください。
4. pErrorHandler ー オプションのカスタム・エラー・ハンドラ。"エラー処理のカスタマイズ" を参照してください。カスタム・エラー・ハンドラを指定しない場合は、メソッドは %XML.XSLT.ErrorHandler の新しいインスタンスを使用します (どちらのクラスの場合でも)。
5. pParms ー スタイル・シートに渡されるパラメータを含む、オプションの InterSystems IRIS 多次元配列。"スタイル・シートで使用するためのパラメータの指定" を参照してください。
6. pCallbackHandler ー XSLT 拡張関数を定義する、オプションのコールバック・ハンドラ。"XSLT 拡張関数の追加と使用" を参照してください。
7. pResolver ー オプションのエンティティ・リゾルバ。"カスタム・エンティティの解析実行" を参照してください。

   (%XML.XSLT2.Transformer のみ) gateway ー %Net.Remote.Gateway のオプションのインスタンス。パフォーマンス向上のために XSLT ゲートウェイ接続を再使用する場合にこの引数を指定します。"XSLT ゲートウェイ・サーバ接続の再使用 (XSLT 2.0)" を参照してください。

参考のため、以下のテーブルにこれらのメソッドの最初の 3 つの引数を並べて比較表示します。

テーブル 12-1: XSLT 変換メソッドの比較

メソッド	pSource (入力 XML)	pXSL (スタイル・シート)	pOutput (出力 XML)
TransformFile()	ファイル・ネームを提供する文字列	ファイル・ネームを提供する文字列	ファイル・ネームを提供する文字列
TransformFileWithCompiledXSL()	ファイル・ネームを提供する文字列	コンパイル済みスタイル・シート	ファイル・ネームを提供する文字列
TransformStream()	ストリーム	ストリーム	参照によって返されるストリーム
TransformStreamWithCompiledXSL()	ストリーム	コンパイル済みスタイル・シート	参照によって返されるストリーム
TransformStringWithCompiledXSL()	文字列	コンパイル済みスタイル・シート	ファイル・ネームを提供する文字列

12.7 例

ここでは、以下のコードを使用する 2 つの変換を紹介します (ただし、入力ファイルは異なります)。

ObjectScript

```
  Set in="c:\0test\xslt-example-input.xml"
  Set xsl="c:\0test\xslt-example-stylesheet.xsl"
  Set out="c:\0test\xslt-example-output.xml"
  Set tSC=##class(%XML.XSLT.Transformer).TransformFile(in,xsl,.out)
  Write tSC
```

12.7.1 例 1 : 単純な置換

この例では、以下の入力 XML から始めます。

XML

```
<?xml version="1.0" ?>
<s1 title="s1 title attr">
  <s2 title="s2 title attr">
    <s3 title="s3 title attr">Content</s3>
  </s2>
</s1>
```

また、以下のスタイル・シートを使用します。

XML

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="xml" indent="yes"/>

<xsl:template match="//@* | //node()">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:apply-templates select="node()"/>
  </xsl:copy>
```

```
</xsl:template>

<xsl:template match="/s1/s2/s3">
<xsl:apply-templates select="@*"/>
<xsl:copy>
Content Replaced
</xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

この場合、出力ファイルは以下のようになります。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<s1 title="s1 title attr">
  <s2 title="s2 title attr">
    <s3>
Content Replaced
</s3>
  </s2>
</s1>
```

12.7.2 例 2：コンテンツの抽出

この例では、以下の入力 XML から始めます。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<MyRoot>
   <MyElement No="13">Some text</MyElement>
   <MyElement No="14">Some more text</MyElement>
</MyRoot>
```

また、以下のスタイル・シートを使用します。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.1"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsl:output method="text"
            media-type="text/plain"/>
<xsl:strip-space elements="*"/>

<!-- utilities not associated with specific tags -->
<!-- emit a newline -->
<xsl:template name="NL">
    <xsl:text>&#xa;</xsl:text>
</xsl:template>

<!-- beginning of processing -->

<xsl:template match="/">
    <xsl:apply-templates/>
</xsl:template>

<xsl:template match="MyElement">
    <xsl:value-of select="@No"/>
    <xsl:text>: </xsl:text>
    <xsl:value-of select="."/>
    <xsl:call-template name="NL"/>
</xsl:template>

</xsl:stylesheet>
```

この場合、出力ファイルは以下のようになります。

```
13: Some text
14: Some more text
```

12.7.3 その他の例

XSLT 1.0 の場合は、"Example()"、"Example2()"、および "%XML.XSLT.Transformer" のその他のメソッドを参照してください。

12.8 エラー処理のカスタマイズ

エラーが発生すると、XSLT プロセッサ (Xalan または Saxon) は現在のエラー・ハンドラの error() メソッドを実行し、引数としてメッセージをそのメソッドに送信します。同様に、致命的なエラーや警告が発生すると、XSLT プロセッサは fatalError() メソッドまたは warning() メソッドを必要に応じて実行します。

これら 3 つのメソッドすべての既定の動作は、現在のデバイスへのメッセージの書き込みです。

エラー処理をカスタマイズするには、以下の手順を実行します。

1. Xalan または Saxon プロセッサで、%XML.XSLT.ErrorHandler のサブクラスを作成します。このサブクラスで、必要に応じて error() メソッド、fatalError() メソッド、および warning() メソッドを実装します。

   これらのメソッドはそれぞれ、XSLT プロセッサから送信されたメッセージを含む文字列である単独の引数を受け入れます。

   これらのメソッドは値を返しません。

2. 次に、以下の操作を行います。

   - スタイル・シートのコンパイル中にこのエラー・ハンドラを使用するには、サブクラスのインスタンスを作成し、スタイル・シートのコンパイル時に引数リストでそれを使用します。"コンパイル済みスタイル・シートの作成" を参照してください。
   - XSLT 変換の実行中にこのエラー・ハンドラを使用するには、サブクラスのインスタンスを作成し、使用する変換メソッドの引数リストでそれを使用します。"XSLT 変換の実行" を参照してください。

12.9 スタイル・シートで使用するためのパラメータの指定

スタイル・シートで使用するためにパラメータを指定するには:

1. %ArrayOfDataTypes のインスタンスを作成します。
2. このインスタンスの SetAt() メソッドを呼び出して、このインスタンスのパラメータとその値を追加します。SetAt() では、最初の引数をパラメータ値として指定し、2 番目の引数をパラメータ名として指定します。

   必要なだけパラメータを追加します。

   例 :

   ```
   Set tParameters=##class(%ArrayOfDataTypes).%New()
   Set tSC=tParameters.SetAt(1,"myparameter")
   Set tSC=tParameters.SetAt(2,"anotherparameter")
   ```

3. このインスタンスを変換メソッドの pParms 引数として使用します。

%ArrayOfDataTypes の代わりに、InterSystems IRIS 多次元配列を使用できます。この多次元配列は、以下の構造と値を持つ任意の数のノードを持つことができます。

ノード	値
arrayname("parameter_name")	parameter_name で指定されたパラメータの値

12.10 XSLT 拡張関数の追加と使用

InterSystems IRIS で XSLT 拡張関数を作成し、以下のようにスタイル・シート内でそれらを使用することができます。

- XSLT 2.0 (Saxon プロセッサ) の場合、ネームスペース `com.intersystems.xsltgateway.XSLTGateway` の evaluate 関数またはネームスペース `http://extension-functions.intersystems.com` の evaluate 関数を使用できます
- XSLT 1.0 (Xalan プロセッサ) の場合、ネームスペース `http://extension-functions.intersystems.com` の evaluate 関数のみを使用できます

既定では (および一例として)、後者の関数は受け取る文字を反転します。ただし、その他の動作が実装されるため、この既定の動作は一般的には使用されません。複数の個別の関数をシミュレートするには、セレクタを最初の引数として渡し、その値を使用して実行する処理を選択するスイッチを実装します。

内部的には、evaluate 関数はメソッド (evaluate()) として XSLT コールバック・ハンドラに実装されます。

XSLT 拡張関数を追加および使用するには、以下の手順を実行します。

1. Xalan または Saxon プロセッサで、%XML.XSLT.CallbackHandler のサブクラスを作成します。このサブクラスで、必要に応じて evaluate() メソッドを実装します。この後のサブセクションを参照してください。
2. スタイル・シートで、evaluate 関数が属するネームスペースを宣言し、必要に応じて evaluate 関数を使用します。この後のサブセクションを参照してください。
3. XSLT 変換の実行中に、サブクラスのインスタンスを作成し、使用する変換メソッドの引数リストでそれを使用します。"XSLT 変換の実行" を参照してください。

12.10.1 evaluate() メソッドの実装

内部的には、XSLT プロセッサを呼び出すコードは、任意の数の位置を示す引数を現在のコールバック・ハンドラの evaluate() メソッドに渡すことができます。コールバック・ハンドラは以下の構造を持つ配列としてそれらを受け取ります。

ノード	値
Args	引数の数
Args(index)	位置 index にある引数の値

このメソッドは単独の返り値を持ちます。返り値は以下のいずれかになります。

- スカラ変数 (文字列または数字など)。
- ストリーム・オブジェクト。これを使用すると、文字列長の制限を超える、きわめて長い文字列を返すことができます。ストリームは、ストリームを読み取るための XSLT プロセッサを有効にする %XML.XSLT.StreamAdapter のインスタンスでラップされる必要があります。以下は、部分的な例です。

Class Member

```
Method evaluate(Args...) As %String
{
 //create stream
 ///...

 // create instance of %XML.XSLT.StreamAdapter to
 // contain the stream
 Set return=##class(%XML.XSLT.StreamAdapter).%New(tStream)

 Quit return

}
```

12.10.2 スタイル・シートでの evaluate の使用

XSLT で XSLT 拡張関数を使用するには、XSLT スタイル・シートで拡張関数のネームスペースを宣言する必要があります。前述のとおり、インターシステムズの evaluate 関数では、このネームスペースは `http://extension-functions.intersystems.com` または `com.intersystems.xsltgateway.XSLTGateway` です。

以下の例は、evaluate を使用するスタイル・シートを示します。

XML

```
<?xml version="1.0"?>

<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
xmlns:isc="http://extension-functions.intersystems.com">

  <xsl:output method="xml" indent="yes"/>

  <xsl:template match="//@* | //node()">
    <xsl:copy>
      <xsl:apply-templates select="@*"/>
      <xsl:apply-templates select="node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="/s1/s2/s3">
  <xsl:apply-templates select="@*"/>
  <xsl:choose>
  <xsl:when test="function-available('isc:evaluate')">
  <xsl:copy>
  <xsl:value-of select="isc:evaluate(.)" disable-output-escaping="yes"/>
  </xsl:copy>
  </xsl:when>
  <xsl:otherwise>
  <xsl:value-of select="."/>
  </xsl:otherwise>
  </xsl:choose>
  </xsl:template>

</xsl:stylesheet>
```

この例の詳細を確認するには、**%XML.XSLT.Transformer** の Example3() メソッドのソース・コードを参照してください。

12.10.3 isc:evaluate キャッシュを使用した作業

XSLT 2.0 ゲートウェイは、isc:evaluate キャッシュに evaluate 関数の呼び出しをキャッシュします。キャッシュの既定の最大サイズは 1000 項目ですが、このサイズを異なる値に設定できます。また、キャッシュをクリアしたり、キャッシュをダンプしたり、以下の形式で **%List** からキャッシュに自動生成したりすることができます。

- キャッシュ・エントリの総数
- 各エントリについて：
  1. evaluate 引数の総数

2. すべての evaluate 引数
3. evaluate 値

キャッシュには、キャッシュできる関数名のフィルタ・リストも含まれます。以下のことに注意してください。

- 関数名をフィルタ・リストに追加したり、フィルタ・リストから削除したりすることができます。
- フィルタ・リストをクリアできます。
- すべての evaluate の呼び出しをキャッシュするブーリアンを設定することで、フィルタ・リストをオーバーライドできます。

フィルタ・リストに関数名を追加しても、evaluate キャッシュのサイズは制限されません。同じ関数を多数呼び出す場合がありますが、異なる引数と返り値になることがあります。関数名と引数の各組合せは、evaluate キャッシュでは異なるエントリになります。

%XML.XSLT2.Transformer クラスからメソッドを使用することで、evaluate キャッシュを操作できます。詳細は、クラスリファレンスを参照してください。

12.11 XSL 変換ウィザードの使用

スタジオで提供されている XSLT 変換を実行するウィザードは、スタイル・シートやカスタム XSLT 拡張関数を短時間でテストしたい場合に便利です。このウィザードを使用する手順は次のとおりです。

1. **[ツール]**→**[アドイン]**→**[XSLT スキーマ・ウィザード]** を選択します。
2. 以下の必須情報を指定します。
   - **[XMLファイル]** で、**[参照]** を選択して、変換する XML ファイルを選択します。
   - **[XSLファイル]** で、**[参照]** を選択して、使用する XSL スタイル・シートを選択します。
   - **[レンダリング]** で、**[テキスト]** または **[XML]** を選択して、変換の表示方法を制御します。
3. この変換で使用する **%XML.XSLT.CallbackHandler** のサブクラスを作成済みの場合は、以下の情報を指定します。
   - **[XSLTヘルパークラス]** の 1 つ目のドロップダウン・リストで、ネームスペースを選択します。
   - **[XSLTヘルパークラス]** の 2 つ目のドロップダウン・リストで、そのクラスを選択します。

   "XSLT 拡張関数の追加と使用" を参照してください。
4. **[完了]** を選択します。

   ダイアログ・ボックスの下部に、変換後のファイルが表示されます。この領域からコピーして貼り付けることができます。
5. このダイアログ・ボックスを閉じるには、**[キャンセル]** を選択します。

13

InterSystems SAX パーサの使用法のカスタマイズ

InterSystems IRIS® データ・プラットフォームで XML ドキュメントを読み取るときは、InterSystems IRIS SAX (Simple API for XML) パーサが使用されます。このトピックでは、InterSystems IRIS SAX パーサを制御するためのオプションを説明します。

13.1 InterSystems IRIS SAX パーサについて

InterSystems IRIS SAX パーサは、InterSystems IRIS により XML ドキュメントが読み取られるたびに使用されます。

これはイベント駆動型の XML パーサです。このパーサは XML ファイルを読み取り、目的の項目 (XML 要素の開始、DTD の開始など) が見つかったときに、コールバックを発行します。

(より正確に言うと、パーサはコンテンツ・ハンドラと連携し、コンテンツ・ハンドラがコールバックを発行します。この区別は、SAX インタフェースをカスタマイズする場合にのみ重要です。詳細は、"カスタム・コンテンツ・ハンドラの作成" を参照してください。)

このパーサでは、XML 1.0 の推奨や関連する多数の標準に準拠する標準 Xerces-C++ ライブラリが使用されます。これらの標準のリストについては、http://xml.apache.org/xerces-c/ を参照してください。

13.2 パーサ・オプション

SAX パーサの動作は、以下の方法で制御できます。

- フラグを設定して、実行する検証や処理の種類を指定します。

  パーサは、ドキュメントが適格な形式の XML ドキュメントであるかどうかを常にチェックします。
- 目的のイベント (つまり、パーサに検出させる項目) を指定します。このためには、目的のイベントを示すマスクを指定します。
- ドキュメント検証の基準となるスキーマ仕様を指定します。
- エンティティの解析は、特殊な目的を持つエンティティ・リゾルバを使用することで無効にできます。
- エンティティの解析用にタイムアウト時間を指定できます。

- パーサがドキュメントの任意のエンティティに対する定義を検出する方法を制御する必要がある場合は、より一般的なカスタム・エンティティ・リゾルバを指定します。
- あるURL に存在するソース・ドキュメントにアクセスする場合は、%Net.HttpRequest のインスタンスとして、Web サーバに送信する要求を指定することができます。

  %Net.HttpRequest の詳細は、"インターネット・ユーティリティの使用法" を参照してください。または、%Net.HttpRequest のクラス・ドキュメントを参照してください。
- カスタム・コンテンツ・ハンドラを指定することができます。
- HTTPS を使用できます。

使用可能なオプションは、InterSystems IRIS SAX パーサをどのように使用しているかによって異なります。以下のテーブルに概略を示します。

テーブル 13-1: %XML クラスの SAX パーサ・オプション

オプション	%XML.Reader	%XML.TextReader	%XML.XPATH.Document	%XML.SAX.Parser
パーサ・フラグの指定	サポートされる	サポートされる	サポートされる	サポートされる
目的の解析イベント（要素の開始、要素の終了、コメントなど）の指定	サポートされない	サポートされる	サポートされない	サポートされる
スキーマ仕様の指定	サポートされる	サポートされる	サポートされる	サポートされる
エンティティの解析の無効化、またはエンティティの解析のカスタマイズ	サポートされる	サポートされる	サポートされる	サポートされる
カスタム HTTP 要求の指定（URL を解析する場合）	サポートされない	サポートされる	サポートされない	サポートされる
コンテンツ・ハンドラの指定	サポートされない	サポートされない	サポートされない	サポートされる
HTTPS の場所でのドキュメントの解析	サポートされる	サポートされない	サポートされない	サポートされる
HTTPS の場所でのエンティティの解決	サポートされない	サポートされない	サポートされない	サポートされる

重要　%XML.SAX.Parser の ParseFile()、ParseStream()、ParseString()、および ParseURL() の各メソッドは、属性のコレクション（属性の名前と値を含む）が指定の文字列長を超える XML 要素を処理できません。このようなドキュメントを処理する必要がある場合は、このような属性の処理が可能な %XML.Reader を使用します。

13.3 パーサ・オプションの指定

パーサの動作の指定方法は、InterSystems IRIS SAX パーサの使い方によって異なります。

- %XML.Reader を使用している場合は、リーダ・インスタンスの Timeout、SAXFlags、SAXSchemaSpec、および EntityResolver プロパティを設定できます。以下はその例です。

ObjectScript

```
#include %occInclude
#include %occSAX
// set the parser options we want
Set flags = $$$SAXVALIDATION
            + $$$SAXNAMESPACES
            + $$$SAXNAMESPACEPREFIXES
            + $$$SAXVALIDATIONSCHEMA

Set reader=##class(%XML.Reader).%New()
Set reader.SAXFlags=flags
```

これらのマクロは、%occSAX.inc インクルード・ファイルで定義されています。

- それ以外の場合は、使用しているメソッドの引数を指定します。以下はその例です。

ObjectScript

```
#include %occInclude
#include %occSAX

//set the parser options we want
Set flags = $$$SAXVALIDATION
            + $$$SAXNAMESPACES
            + $$$SAXNAMESPACEPREFIXES
            + $$$SAXVALIDATIONSCHEMA

Set status=##class(%XML.TextReader).ParseFile(myfile,.doc,,flags)
```

関連するメソッドの引数リストに関する詳細は、以下の章またはセクションを参照してください。

- %XML.Reader の詳細は、"InterSystems IRIS オブジェクトへの XML のインポート" を参照してください。
- %XML.TextReader の詳細は、"%XML.TextReader の使用" を参照してください。
- %XML.XPATH.Document の詳細は、"XPath 式の評価" を参照してください。
- %XML.SAX.Parser の詳細は、"カスタム・コンテンツ・ハンドラの作成" を参照してください。

または、このクラスのドキュメントを参照してください。

13.4 パーサ・フラグの設定

%occSAX.inc インクルード・ファイルには、Xerces パーサにより実行される検証を制御するために使用されるフラグのリストが含まれます。基本的なフラグは以下のとおりです。

- $$$SAXVALIDATION ― スキーマ検証を実行するかどうかを指定します。このフラグがオン (既定値) の場合、検証エラーがすべて報告されます。
- $$$SAXNAMESPACES ― ネームスペースを認識するかどうかを指定します。このフラグがオン (既定値) の場合、パーサによりネームスペースが処理されます。このフラグがオフの場合は、InterSystems IRIS は、%XML.SAX.ContentHandler の startElement() コールバックで、要素の localname を空文字列に設定します。
- $$$SAXNAMESPACEPREFIXES ― ネームスペースの接頭語を処理するかどうかを指定します。このフラグがオンの場合、ネームスペースの宣言に使用された元の接頭語付きの名前と属性がパーサから報告されます。既定では、このフラグはオフです。
- $$$SAXVALIDATIONDYNAMIC ― 検証を動的に実行するかどうかを指定します。このフラグがオン (既定値) の場合、検証は文法が指定されている場合のみ実行されます。
- $$$SAXVALIDATIONSCHEMA ― スキーマに対する検証を実行するかどうかを指定します。このフラグがオン (既定値) の場合、スキーマが指定されていれば、それに対する検証が実行されます。

- $$$SAXVALIDATIONSCHEMAFULLCHECKING ― 時間のかかるチェックや、大量のメモリを消費するチェックを含め、完全なスキーマ制約チェックを実行するかどうかを指定します。このフラグがオンの場合、制約チェックがすべて行われます。既定では、このフラグはオフです。
- $$$SAXVALIDATIONREUSEGRAMMAR ― 同じ InterSystems IRIS プロセスで、これ以降、行われる解析で再利用するために文法をキャッシュしておくかどうかを指定します。既定では、このフラグはオフです。
- $$$SAXVALIDATIONPROHIBITDTDS ― パーサが DTD に遭遇した場合にエラーをスローする特殊フラグ。DTD の処理を避ける必要がある場合は、このフラグを使用します。このフラグを使用するには、%XML.SAX.Parser のさまざまな解析メソッドに渡される解析フラグに値 $$$SAXVALIDATIONPROHIBITDTDS を明示的に追加する必要があります。

次のフラグは、複数の基本フラグを使いやすく組み合わせたものです。

- $$$SAXDEFAULTS ― SAX の既定値に相当します。
- $$$SAXFULLDEFAULT ― SAX の既定値、およびネームスペース接頭語を処理するためのオプションに相当します。
- $$$SAXNOVALIDATION ― スキーマの検証は行いませんが、ネームスペースとネームスペースの接頭語を認識します。SAX パーサは、ドキュメントが適格な形式の XML ドキュメントであるかどうかを常にチェックします。

詳細は、%occSAX.inc を参照してください。このファイルには、このような種類の検証の詳細へのリンクも用意されています。

次の部分は、パーサ・オプションを組み合わせる方法を示しています。

```
...
#include %occInclude
#include %occSAX
...
 ;; set the parser options we want
 set opt = $$$SAXVALIDATION
                + $$$SAXNAMESPACES
                + $$$SAXNAMESPACEPREFIXES
                + $$$SAXVALIDATIONSCHEMA
...
  set status=##class(%XML.TextReader).ParseFile(myfile,.doc,,opt)
  //check status
  if $$$ISERR(status) {do $System.Status.DisplayError(status) quit}
```

13.5 イベント・マスクの指定

%occSAX.inc インクルード・ファイルにも、処理の対象となるイベント・コールバックを指定するためのフラグのリストが記載されています。パフォーマンス上の理由から、必要なコールバックのみ処理することをお勧めします。マスクを指定する必要があるかどうかは、InterSystems IRIS SAX パーサの呼び出しに使用するクラスによって異なります。

- %XML.TextReader では、既定値は $$$SAXCONTENTEVENTS です。コメント以外のイベント・コールバックがすべて処理されます。
- %XML.SAX.Parser では、既定値は 0 で、これはパーサにより、コンテンツ・ハンドラの Mask() メソッドが呼び出されることを表します。このメソッドは、カスタマイズされたイベント・コールバックをすべて検出して、マスクを計算します。処理されるのはこれらのイベントのみです。カスタム・コンテンツ・ハンドラを作成した場合は、%XML.SAX.Parser を使用します。詳細は、"カスタム・コンテンツ・ハンドラの作成" を参照してください。

13.5.1 基本的なフラグ

基本的なフラグは以下のとおりです。

- $$$SAXSTARTDOCUMENT ー ドキュメントを開始したときにコールバックを発行するようにパーサに指示します。
- $$$SAXENDDOCUMENT ー ドキュメントを終了したときにコールバックを発行するようにパーサに指示します。
- $$$SAXSTARTELEMENT ー 要素の先頭を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXENDELEMENT ー 要素の末尾を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXCHARACTERS ー 文字を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXPROCESSINGINSTRUCTION ー プロセス指示を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXSTARTPREFIXMAPPING ー 接頭マッピングの先頭を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXENDPREFIXMAPPING ー 接頭マッピングの末尾を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXIGNORABLEWHITESPACE ー 無視可能な空白を見つけたときにコールバックを発行するようにパーサに指示します。これは、ドキュメントが DTD を持ち、検証が有効になっている場合にのみ適用されます。
- $$$SAXSKIPPEDENTITY ー スキップされたエンティティを見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXCOMMENT ー コメントを見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXSTARTCDATA ー CDATA セクションの先頭を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXENDCDATA ー CDATA セクションの末尾を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXSTARTDTD ー DTD の先頭を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXENDDTD ー DTD の末尾を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXSTARTENTITY ー エンティティの先頭を見つけたときにコールバックを発行するようにパーサに指示します。
- $$$SAXENDENTITY ー エンティティの末尾を見つけたときにコールバックを発行するようにパーサに指示します。

13.5.2 便利な組み合わせフラグ

次のフラグは、複数の基本フラグを使いやすく組み合わせたものです。

- $$$SAXCONTENTEVENTS ー コンテンツを含むイベントすべてに対してコールバックを発行するようにパーサに指示します。
- $$$SAXLEXICALEVENT ー 言語イベントすべてに対してコールバックを発行するようにパーサに指示します。
- $$$SAXALLEVENTS ー すべてのイベントに対してコールバックを発行するようにパーサに指示します。

13.5.3 1 つのマスクへのフラグの組み合わせ

次の部分は、複数のフラグを 1 つのマスクにまとめる方法を示しています。

```
...
#include %occInclude
#include %occSAX
...
 // set the mask options we want
 set mask = $$$SAXSTARTDOCUMENT
               + $$$SAXENDDOCUMENT
               + $$$SAXSTARTELEMENT
               + $$$SAXENDELEMENT
               + $$$SAXCHARACTERS
...
 // create a TextReader object (doc) by reference
 set status = ##class(%XML.TextReader).ParseFile(myfile,.doc,,,mask)
```

13.6 スキーマ・ドキュメントの指定

ドキュメント・ソースの検証の基準となるスキーマ仕様を指定します。次のように、ネームスペース/URL のペアをコンマで区切って指定したリストを含む文字列を指定してください。

```
"namespace URL,namespace URL,namespace URL,..."
```

ここで、namespace は (ネームスペース接頭語ではなく) XML ネームスペースであり、URL はネームスペースのスキーマ・ドキュメントの位置を表す URL です。ネームスペースと URL の値の間は、1 つの空白文字で区切られています。例えば、以下は 1 つのネームスペースが含まれたスキーマ仕様を示しています。

```
"http://www.myapp.org http://localhost/myschemas/myapp.xsd"
```

以下は 2 つのネームスペースが含まれたスキーマ仕様を示しています。

```
"http://www.myapp.org http://localhost/myschemas/myapp.xsd,http://www.other.org
http://localhost/myschemas/other.xsd"
```

13.7 エンティティの解析の無効化

検証を無効にするように SAX フラグを設定した場合でも、SAX パーサは外部エンティティを解析しようとします。これは、対象のエンティティの場所によっては時間のかかる操作になることがあります。

クラス %XML.SAX.NullEntityResolver は、常に空のストリームを返すエンティティ・リゾルバを実装します。エンティティの解析を無効にする場合は、このクラスを使用します。具体的には、XML ドキュメントの読み取りの際は、エンティティ・リゾルバとして %XML.SAX.NullEntityResolver のインスタンスを使用します。以下はその例です。

```
   Set resolver=##class(%XML.SAX.NullEntityResolver).%New()
   Set reader=##class(%XML.Reader).%New()
   Set reader.EntityResolver=resolver

   Set status=reader.OpenFile(myfile)
   ...
```

重要　この変更によって外部エンティティの解析はすべて無効になるため、この技法を使用すると、XML ドキュメント内のすべての外部 DTD およびスキーマ参照も無効になります。

13.8 カスタム・エンティティの解析実行

XMLドキュメントには、外部 DTD または他のエンティティへの参照が含まれている場合があります。既定では、InterSystems IRIS は、これらのエンティティのソース・ドキュメントを見つけて、それらを解析しようとします。InterSystems IRIS による外部エンティティの解析方法を制御するには、以下の手順を使用します。

1. エンティティ・リゾルバ・クラスを定義します。

   このクラスは、%XML.SAX.EntityResolver クラスを拡張し、resolveEntity() メソッドを実装する必要があります。このメソッドには以下のシグニチャがあります。

   ```
   method resolveEntity(publicID As %Library.String, systemID As %Library.String) as %Library.Integer
   ```

   このメソッドは、XML プロセッサが外部エンティティ (DTD など) への参照を検出するたびに呼び出されます。publicID と systemID は、そのエンティティのパブリック識別子文字列とシステム識別子文字列です。

   このメソッドは、エンティティまたはドキュメントを取得し、それをストリームとして返してから、そのストリームを %XML.SAX.StreamAdapter のインスタンスにラップする必要があります。このクラスは、ストリームの特性の決定に必要なメソッドを提供します。

   エンティティを解析できない場合、メソッドでは、$$$NULLOREF を返して、SAX パーサにエンティティを解析できないことを伝える必要があります。

   重要　メソッド・シグニチャでは返り値が %Library.Integer であることが示されているにもかかわらず、メソッドは、%XML.SAX.StreamAdapter またはそのクラスのサブクラスのインスタンスを返す必要があります。

   また、外部エンティティを参照する識別子はドキュメントに指定されているように常に resolveEntity() メソッドに渡されます。特に、そのような識別子が相対 URL を使用する場合、その識別子は相対 URL として渡されます。これは、参照ドキュメントの実際の場所が resolveEntity() メソッドに渡されないことを意味し、エンティティを解決できません。このようなシナリオでは、カスタムのエンティティ・リゾルバではなく既定のエンティティ・リゾルバを使用してください。

   エンティティ・リゾルバ・クラスの例は、%XML.SAX.EntityResolver のソース・コードを参照してください。

2. XML ドキュメントを読み取る場合は、以下の手順を実行します。

   a. エンティティ・リゾルバ・クラスのインスタンスを作成します。

   b. XML ドキュメントを読み取る場合は、"パーサ・オプションの指定" で説明しているように、そのインスタンスを使用します。

"エンティティの解析の無効化" も参照してください。%XML.SAX.NullEntityResolver (そのセクションで説明しています) は %XML.SAX.EntityResolver のサブクラスであることに注意してください。

13.8.1 例 1

例えば、以下の XML ドキュメントを考えてみます。

```
<?xml version="1.0" ?>
<!DOCTYPE html SYSTEM  "c://temp/html.dtd">
<html>
<head><title></title></head>
<body>
<p>Some < xhtml-content > with custom entities &entity1; and &entity2;.</p>
<p>Here is another paragraph with &entity1; again.</p>
</body></html>
```

このドキュメントでは、以下の DTD を使用します。

```
<!ENTITY entity1
         PUBLIC "-//WRC//TEXT entity1//EN"
         "http://www.intersystems.com/xml/entities/entity1">
<!ENTITY entity2
         PUBLIC "-//WRC//TEXT entity2//EN"
         "http://www.intersystems.com/xml/entities/entity2">
<!ELEMENT html (head, body)>
<!ELEMENT head (title)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT body (p)>
<!ELEMENT p (#PCDATA)>
```

このドキュメントを読み取るには、以下のようなカスタム・エンティティ・リゾルバが必要になります。

Class Definition

```
Class CustomResolver.Resolver Extends %XML.SAX.EntityResolver
{

Method resolveEntity(publicID As %Library.String, systemID As %Library.String) As %Library.Integer
{
    Try {
        Set res=##class(%Stream.TmpBinary).%New()
        //check if we are here to resolve a custom entity
        If systemID="http://www.intersystems.com/xml/entities/entity1"
        {
            Do res.Write("Value for entity1")
            Set return=##class(%XML.SAX.StreamAdapter).%New(res)
            }
            Elseif systemID="http://www.intersystems.com/xml/entities/entity2"
            {
                Do res.Write("Value for entity2")
                Set return=##class(%XML.SAX.StreamAdapter).%New(res)
            }
            Else //otherwise call the default resolver
            {
                Set res=##class(%XML.SAX.EntityResolver).%New()
                Set return=res.resolveEntity(publicID,systemID)
            }
    }
    Catch
    {
        Set return=$$$NULLOREF
    }
    Quit return
}

}
```

以下のクラスには、前述のファイルを解析してカスタム・リゾルバを使用するデモ・メソッドが含まれています。

Class Definition

```
Include (%occInclude, %occSAX)

Class CustomResolver.ParseFileDemo
{

ClassMethod ParseFile()
{
    Set res= ##class(CustomResolver.Resolver).%New()
    Set file="c:/temp/html.xml"
    Set parsemask=$$$SAXALLEVENTS+$$$SAXERROR
    Set status=##class(%XML.TextReader).ParseFile(file,.textreader,res,,parsemask,,0)
    If $$$ISERR(status) {Do $system.OBJ.DisplayError(status) Quit }

    Write !,"Parsing the file ",file,!
    Write "Custom entities in this file:"
    While textreader.Read()
    {
        If textreader.NodeType="entity"{
            Write !, "Node:", textreader.seq
            Write !,"    name: ", textreader.Name
            Write !,"    value: ", textreader.Value
        }
    }
```

```
}
}
```

以下に、ターミナル・セッションでのこのメソッドの出力を示します。

Terminal

```
GXML>do ##class(CustomResolver.ParseFileDemo).ParseFile()

Parsing the file c:/temp/html.xml
Custom entities in this file:
Node:13
    name: entity1
    value: Value for entity1
Node:15
    name: entity2
    value: Value for entity2
Node:21
    name: entity1
    value: Value for entity1
```

13.8.2 例 2

例として、以下のような XML ドキュメントを読み取る必要があると想定してみます。

```
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
 "c:\test\doctypes\docbook\docbookx.dtd">
```

この場合、resolveEntity メソッドは、`-//OASIS//DTD DocBook XML V4.1.2//EN` に設定された publicId と、`c:\test\doctypes\docbook\docbookx.dtd` に設定された systemId で実行されます。

resolveEntity メソッドは、外部エンティティに対する正確なソースを決定し、これをストリームとして返してから、このストリームを %XML.StreamAdaptor のインスタンスにラップします。XML パーサは、この特別なストリームからエンティティ定義を読み取ります。

この例は、InterSystems IRIS ライブラリの %XML.Catalog クラスと %XML.CatalogResolver クラスを参照してください。%XML.Catalog クラスは、パブリック識別子とシステム識別子を URL と関連させる、単純なデータベースを定義します。%XML.CatalogResolver クラスは、このデータベースを使用して、与えられた識別子に対する URL を検索する、エンティティ・リゾルバ・クラスです。%XML.Catalog クラスは、SGML 形式のカタログ・ファイルから、このデータベースをロードできます。このファイルは、識別子を標準形式で URL にマップします。

13.9 カスタム・コンテンツ・ハンドラの作成

InterSystems IRIS SAX パーサを直接呼び出す場合、自分固有のニーズに合わせて、カスタム・コンテンツ・ハンドラを作成することができます。このセクションでは、以下の項目について説明します。

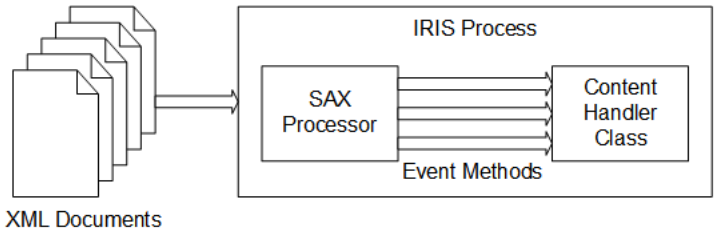
- 概要
- コンテンツ・ハンドラでカスタマイズするメソッドの説明
- %XML.SAX.Parser クラスにおける解析メソッドの引数リストの概要
- 例

13.9.1 カスタム・コンテンツ・ハンドラの作成の概要

InterSystems IRIS SAX パーサによる XML のインポートおよび処理方法をカスタマイズするには、カスタム SAX コンテンツ・ハンドラを作成して使用します。具体的には %XML.SAX.ContentHandler のサブクラスを作成します。次に、その新

しいクラスで、要求されたアクションを実行するために、必要に応じて、既定のメソッドをオーバーライドします。XML ドキュメントを解析するには、引数として、新しいコンテンツ・ハンドラを使用します。このためには、%XML.SAX.Parser クラスの解析メソッドを使用します。

以下の図は、この操作の詳細です。



カスタム・インポート・メカニズムを作成して使用するプロセスは以下のとおりです。

1. %XML.SAX.ContentHandler から派生するクラスを生成します。
2. このクラスに、オーバーライドしたいメソッドをインクルードし、必要に応じて、新しい定義を追加します。
3. %XML.SAX.Parser クラスの解析メソッド (ParseFile()、ParseStream()、ParseString()、ParseURL()) の 1 つを使用して、XML ドキュメントを読み取るクラス・メソッドを記述します。

   この解析メソッドを呼び出すときには、引数として、このカスタム・コンテンツ・ハンドラを指定します。

13.9.2 カスタマイズ可能な SAX コンテンツ・ハンドラのメソッド

%XML.SAX.ContentHandler クラスは、指定された時間に、特定のメソッドを自動的に実行します。これらのメソッドをオーバーライドして、コンテンツ・ハンドラの動作をカスタマイズすることができます。

13.9.2.1 イベントへの応答

%XML.SAX.ContentHandler クラスは XML ファイルを解析し、XML ファイルの特定の場所に到達したとき、イベントを生成します。実行されるメソッドは、イベントにより異なります。これらのメソッドは以下のとおりです。

- OnPostParse() ー XML 解析が完了したときにトリガされる
- characters() ー 文字データからトリガされる
- comment() ー コメントからトリガされる
- endCData() ー CDATA セクションの最後によってトリガされる
- endDocument() ー ドキュメントの最後によってトリガされる
- endDTD() ー DTD の最後によってトリガされる
- endElement() ー 要素の最後によってトリガされる
- endEntity() ー エンティティの最後によってトリガされる
- endPrefixMapping() ー ネームスペース接頭マッピングの最後によってトリガされる
- ignorableWhitespace() ー 要素コンテンツ内の無視可能な空白によってトリガされる
- processingInstruction() ー XML 処理命令によってトリガされる
- skippedEntity() ー スキップされたエンティティによってトリガされる
- startCData() ー CDATA セクションの最初によってトリガされる
- startDocument() ー ドキュメントの最初によってトリガされる

- startDTD() ー DTD の最初によってトリガされる
- startElement() ー 要素の最初によってトリガされる
- startEntity() ー エンティティの最初によってトリガされる
- startPrefixMapping() ー ネームスペース接頭マッピングの開始によってトリガされる

これらのメソッドは既定では空ですが、カスタム・コンテンツ・ハンドラでオーバーライドできます。期待される引数リストと返り値の詳細は、%XML.SAX.ContentHandler クラスのドキュメントを参照してください。

13.9.2.2 エラー処理

%XML.SAX.ContentHandler クラスも、特定のエラーが発生したときにメソッドを実行します。

- error() ー リカバリ可能なパーサ・エラーによってトリガされる
- fatalError() ー 致命的な XML 解析エラーによってトリガされる
- warning() ー パーサ警告の報告によってトリガされる

これらのメソッドは既定では空ですが、カスタム・コンテンツ・ハンドラでオーバーライドできます。期待される引数リストと返り値の詳細は、%XML.SAX.ContentHandler クラスのドキュメントを参照してください。

13.9.2.3 イベント・マスクの計算

InterSystems IRIS SAX パーサを (%XML.SAX.Parser クラス経由で) 呼び出すときに、必要なコールバックを示すマスク引数を指定できます。マスク引数が指定されていない場合、パーサにより、コンテンツ・ハンドラの Mask() メソッドが呼び出されます。このメソッドは、コンテンツ・ハンドラでオーバーライドされたメソッドに対応する複合マスクを表す整数を返します。

例えば、startElement() および endElement() メソッドの新規バージョンを含むカスタム・コンテンツ・ハンドラを作成したとします。この場合、Mask() メソッドは、$$$SAXSTARTELEMENT と $$$SAXENDELEMENT、つまりこれら 2 つのイベントに対応するフラグの合計に等しい数値を返します。解析メソッドにマスク引数を指定しなかった場合、パーサにより、コンテンツ・ハンドラの Mask() メソッドが呼び出されるため、これら 2 つのイベントのみ処理されます。

13.9.2.4 その他の便利なメソッド

%XML.SAX.ContentHandler クラスには、特定の状況に役立つメソッドが用意されています。

- LocatePosition() ー 解析されたドキュメントの現在位置を表す 2 つの引数を参照として返します。1 つ目の引数は行番号を、2 つ目の引数は行のオフセットを表します。
- PushHandler() ー スタックに新しいコンテンツ・ハンドラをプッシュします。これ以降、このハンドラが処理を完了するまで、SAX からのコールバックはすべて、この新しいコンテンツ・ハンドラに送られます。

  このメソッドは、あるタイプのドキュメントを解析しているときに、別の方法で解析する必要のある XML のセグメントに遭遇した場合に使用します。この場合、別の方法で処理する必要のあるセグメントを検出したときには、PushHandler() メソッドを呼び出します。このメソッドは、新しいコンテンツ・ハンドラ・インスタンスを作成します。直前のコンテンツ・ハンドラに戻るために PopHandler() を呼び出すまで、すべてのコールバックはこのコンテンツ・ハンドラに送られます。
- PopHandler() ー スタックに入っている直前のコンテンツ・ハンドラに戻ります。

これらのメソッドは確定されています。オーバーライドできません。

13.9.3 SAX 解析メソッドの引数リスト

ドキュメント・ソースを解析するには、%XML.SAX.Parser クラスの ParseFile()、ParseStream()、ParseString()、または ParseURL() メソッドを使用します。どのような場合でも、ソース・ドキュメントは、適格な XML ドキュメント、つまり、XML 構文の基本的な規約に従ったドキュメントである必要があります。以下に、全引数のリストを順番に記載します。

1. pFilename、pStream、pString、または pURL ー ドキュメント・ソース。
2. pHandler ー コンテンツ・ハンドラ。これは %XML.SAX.ContentHandler クラスのインスタンスです。
3. pResolver ー ソースの解析時に使用されるエンティティ・リゾルバ。"カスタム・エンティティの解析実行" を参照してください。
4. pFlags ー SAX パーサによる検証と処理を制御するフラグ。"パーサ・フラグの設定" を参照してください。
5. pMask ー XML ソース内の目的の項目を指定するためのマスク。%XML.SAX.Parser の解析メソッドに対する既定のマスクは 0 であるため、通常はこの引数を指定する必要はありません。つまり、パーサは、コンテンツ・ハンドラの Mask() メソッドを呼び出します。このメソッドはマスクを計算するために、イベント・ハンドラでカスタマイズされたイベント・コールバックをすべて (コンパイル中に) 検出します。処理されるのはこれらのイベント・コールバックのみです。ただし、マスクを指定する必要がある場合は、"イベント・マスクの指定" を参照してください。
6. pSchemaSpec ー ドキュメント・ソースの検証の基準となるスキーマ仕様。この引数は、ネームスペース/URL のペアをコンマで区切って指定したリストを含む文字列です。

   ```
   "namespace URL,namespace URL"
   ```

   ここで、namespace はスキーマに使用する XML ネームスペースで、URL はスキーマ・ドキュメントの位置を表す URL です。ネームスペースと URL の値の間は、1 つの空白文字で区切られています。
7. pHttpRequest (ParseURL() メソッドのみ) ー %Net.HttpRequest のインスタンスとしての、Web サーバへの要求。

   %Net.HttpRequest の詳細は、"インターネット・ユーティリティの使用法" を参照してください。または、%Net.HttpRequest のクラス・ドキュメントを参照してください。
8. pSSLConfiguration ー クライアント SSL/TLS 構成の構成名。

   "HTTPS の使用" を参照してください。

注釈　ただし、この引数リストは、%XML.TextReader クラスの解析メソッドの引数リストとは多少異なります。例えば、%XML.TextReader には、カスタム・コンテンツ・ハンドラを指定するためのオプションはありません。

13.9.4 SAX ハンドラの例

ファイルに表示される XML 要素すべてのリストが必要であるとします。このリストを作成するには、開始要素をすべてメモする必要があります。そのプロセスは、以下のとおりです。

1. %XML.SAX.ContentHandler から派生した、MyApp.Handler というクラスを生成します。

   Class Definition

   ```
   Class MyApp.Handler Extends %XML.SAX.ContentHandler
   {
   }
   ```

2. startElement() メソッドを、以下のコンテンツでオーバーライドします。

Class Definition

```
Class MyApp.MyHandler extends %XML.SAX.ContentHandler
{
// ...

Method startElement(uri as %String, localname as %String,
             qname as %String, attrs as %List)
{
    //we have found an element
    write !,"Element: ",localname
}

}
```

3. 外部ファイルを読み取り、解析する Handler クラスにクラス・メソッドを追加します。

Class Definition

```
Class MyApp.MyHandler extends %XML.SAX.ContentHandler
{
// ...
ClassMethod ReadFile(file as %String) as %Status
{
    //create an instance of this class
    set handler=..%New()

    //parse the given file using this instance
    set status=##class(%XML.SAX.Parser).ParseFile(file,handler)

    //quit with status
    quit status
}
}
```

処理を実行するためにアプリケーションで実行されるため、これはクラス・メソッドであることに注意してください。このメソッドは、以下を実行します。

a. これは、コンテンツ・ハンドラ・オブジェクトのインスタンスを生成します。

ObjectScript

```
set handler=..%New()
```

b. これは、%XML.SAX.Parser クラスの ParseFile() メソッドを呼び出します。filename で指定されたドキュメントを検証および解析し、コンテンツ・ハンドラ・オブジェクトのさまざまなイベント処理メソッドを実行します。

ObjectScript

```
set status=##class(%XML.SAX.Parser).ParseFile(file,handler)
```

イベントが発生するたびに、パーサはドキュメント（開始要素、または終了要素など）を解析しますが、パーサはコンテンツ・ハンドラ・オブジェクトで適切なメソッドを実行します。この例でオーバーライドされたメソッドは startElement() のみで、このメソッドはその後、要素名を書き出します。終了要素への到達など、その他のイベントが発生しても、何も起こりません（既定の動作です）。

c. ParseFile() メソッドがファイルの最後に到達したときに戻ります。ハンドラ・オブジェクトは範囲外になり、メモリから自動的に削除されます。

4. アプリケーションで適切な位置に到達したら、ReadFile() メソッドを実行し、解析のためにファイルに渡します。

ObjectScript

```
 Do ##class(Samples.MyHandler).ReadFile(filename)
```

filename は、現在読み取られているファイルのパスです。

例えば、このファイルのコンテンツが以下の場合、

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
  <Person>
    <Name>Edwards,Angela U.</Name>
    <DOB>1980-04-19</DOB>
    <GroupID>K8134</GroupID>
    <HomeAddress>
      <City>Vail</City>
      <Zip>94059</Zip>
    </HomeAddress>
    <Doctors>
      <Doctor>
        <Name>Uberoth,Wilma I.</Name>
      </Doctor>
      <Doctor>
        <Name>Wells,George H.</Name>
      </Doctor>
    </Doctors>
  </Person>
</Root>
```

この例の出力は、以下のようになります。

```
Element: Root
Element: Person
Element: Name
Element: DOB
Element: GroupID
Element: HomeAddress
Element: City
Element: Zip
Element: Doctors
Element: Doctor
Element: Name
Element: Doctor
Element: Name
```

13.10 HTTPS の使用

%XML.SAX.Parser は HTTPS をサポートしています。そのため、このクラスを使用すると、以下を実行できます。

- (ParseURL() の場合) HTTPS の場所で提供される XML ドキュメントの解析。
- (すべての解析メソッドの場合) HTTPS の場所でのエンティティの解析。

どのような場合でも、これらの項目が HTTPS の場所で提供されるときには、以下を実行します。

1. 管理ポータルを使用して、必要な接続の詳細を格納する SSL/TLS 構成を作成します。詳細は、インターシステムズの "TLS ガイド" を参照してください。

   これは、1 回限りの手順です。

2. 適用可能な %XML.SAX.Parser の解析メソッドを呼び出すときは、pSSLConfiguration 引数を指定します。

既定では、InterSystems IRIS は Xerces エンティティの解析を使用します。%XML.SAX.Parser は、以下の場合にのみ、専用のエンティティの解析を使用します。

- pSSLConfiguration 引数が NULL 以外。
- プロキシ・サーバが構成されている。

  "HTTP 要求の送信" の "プロキシ・サーバの使用法" を参照してください。

14

XML スキーマからのクラスの生成

スタジオは、(ファイルまたは URL からの) XML スキーマを読み取り、そのスキーマで定義されたタイプに対応する XML 対応クラスのセットを生成するウィザードを提供します。すべてのクラスは **%XML.Adaptor** を拡張します。クラスを収めるパッケージと、クラス定義の詳細を制御するさまざまなオプションを指定します。

ウィザードはクラス・メソッドとしても用意されているので、利用可能です。内部的には、SOAP ウィザードは、WSDL ドキュメントを読み取って Web クライアントまたは Web サービスを生成するときに、このクラス・メソッドを使用します。"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。

このウィザードでは、データは InterSystems IRIS® データ・プラットフォームにロードされません。クラスの生成後にデータをロードする必要がある場合は、"オブジェクトへの XML のインポート" を参照してください。

注釈　使用するどの XML ドキュメントの XML 宣言にも、そのドキュメントの文字エンコードを明記する必要があります。これにより、ドキュメントが宣言どおりにエンコードされます。文字エンコードが宣言されていない場合は、"入出力の文字エンコード" で説明されている既定値が使用されます。これらの既定値が正しくない場合は、XML 宣言を修正して、実際に使用されている文字セットを指定するようにします。

14.1 ウィザードの使用法

XML スキーマ・ウィザードを使用するには、以下の手順を実行します。

1. **[ツール]→[アドイン]→[XMLスキーマウィザード]** を選択します。
2. 最初の画面で、使用する XML スキーマを指定します。以下のいずれかを行います。
   - **[スキーマファイル]** で **[参照]** を選択して、XML スキーマ・ファイルを選択します。
   - **[URL]** で、スキーマの URL を指定します。
3. **[次へ]** を選択します。

   次の画面に、正しいものを選択したことを確認できるように、スキーマが表示されます。
4. 必要に応じて、以下のオプションも選択します。
   - **[空のクラスを保持]**。このオプションによりプロパティを持たない未使用のクラスを保存するかどうかを指定します。このオプションを選択すると、そのようなクラスがウィザードの最後に削除されなくなります。選択しない場合は、削除されます。
   - **[配列プロパティ以外を作成]**。このオプションは、ウィザードで配列プロパティを生成するかどうかを制御します。このオプションを選択すると、ウィザードでは、配列プロパティが生成されない代わりに別の形式が生成されます。"生成されたクラスの詳細" の "配列プロパティの作成" を参照してください。

- [nillable 要素のXMLNIL プロパティ・パラメータを生成]。このオプションは生成されたクラスの使用可能なプロパティに対して XMLNIL プロパティ・パラメータをウィザードで指定するかどうかを制御します。

  このオプションは `nillable="true"` で指定された XML 要素に対応する各プロパティに適用されます。このオプションを選択した場合、ウィザードにより `XMLNIL=1` がプロパティ定義に追加されます。それ以外の場合、このパラメータの追加はありません。このパラメータの詳細は、"空文字列および NULL 値の処理" を参照してください。

- [nillable 要素の XMLNILNOOBJECT プロパティ・パラメータを生成]。このオプションは生成されたクラスの使用可能なプロパティに対して XMLNILNOOBJECT プロパティ・パラメータをウィザードで指定するかどうかを制御します。

  このオプションは `nillable="true"` で指定された XML 要素に対応する各プロパティに適用されます。このオプションを選択した場合、ウィザードにより `XMLNILNOOBJECT=1` がプロパティ定義に追加されます。それ以外の場合、このパラメータの追加はありません。このパラメータの詳細は、"空文字列および NULL 値の処理" を参照してください。

5. [次へ] を選択します。

   次の画面に、生成するクラスについてのオプションに関する基本情報が表示されます。

6. この画面で、以下のオプションを指定します。

   - 生成されたクラスをウィザードによってコンパイルするには、希望に応じて [コンパイルにより生成されたクラス] を選択します。
   - 必要に応じて、[NAMESPACEクラスパラメータ追加] を選択し、NAMESPACE パラメータを指定します。この場合、NAMESPACE は、スキーマで targetNamespace の値に設定されます。

     このオプションのチェックを外したままにすると、NAMESPACE は指定されません。

     どのような場合でも、このオプションを選択することをお勧めします。これは、すべての XML 対応クラスは、XML ネームスペースに割り当てる必要があるためです。(ただし、下位互換性を確保するため、このオプションのチェックを外しておくことは可能です。)
   - 生成されたクラスを永続クラスにする場合は、[永続クラス作成] を選択します。これにより、クラスは %Persistent を拡張します。

     これは、個々のクラスについて、後からウィザードで変更できます。
   - 永続クラスを生成する場合、他の <complexType> B の <sequence> で構成される <complexType> A を扱う方法を決定するオプションがあります。ウィザードによりプロパティ A が含まれる永続クラスが生成されるとき、このプロパティで 3 つの形式があります。これは、オブジェクトのリスト、一対多のリレーションシップ (既定)、または親子リレーションシップとして定義できます。オプション内容をまとめたテーブルを以下に示します。

永続クラス内でコレクションプロパティにリレーションシップを使用する	インデックスを多対一のリレーションシップに追加する	親子リレーションシップを使用	生成されるプロパティAの形式
選択 (既定)	未選択	未選択	インデックスが存在しない一対多のリレーションシップ
選択 (既定)	選択	未選択	多側でインデックスが存在する一対多のリレーションシップ
選択 (既定)	[親子リレーションシップを使用] を選択した場合、このオプションは無視されます	選択	親子リレーションシップ
未選択	未選択	未選択	オブジェクトのリスト

また、**[親子リレーションシップを使用]** を選択していない場合、必要に応じて、**[カスケード削除するために %OnDelete メソッドをクラスに追加する]** オプションを選択します。このオプションを選択すると、ウィザードによりクラス定義が生成されるときに、これらのクラスに %OnDelete() コールバック・メソッドの実装が含まれます。生成された %OnDelete() メソッドによって、クラスで参照されるすべての永続オブジェクトが削除されます。**[親子リレーションシップを使用]** を選択している場合は、このオプションを選択しないでください。親子リレーションシップにより既に同様のロジックが提供されています。

注釈　生成されたクラスを変更する場合、必要に応じて必ず %OnDelete() コールバック・メソッドを変更してください。

- 永続クラスを生成する場合、オブジェクトの InterSystems IRIS 内部識別子を投影できるように、ウィザードでは各オブジェクトタイプ・クラスに一時プロパティを追加することができます。選択肢は以下のとおりです。
  - **[なし]** ー このオプションを選択すると、ウィザードは、ここで説明したいずれのプロパティも追加しません。
  - **[Id使用]** ー このオプションを選択すると、ウィザードは以下のプロパティを各オブジェクトタイプ・クラスに追加します。

    ```
    Property %identity As %XML.Id (XMLNAME="_identity", XMLPROJECTION="ATTRIBUTE") [Transient];
    ```

  - **[Oid使用]** ー このオプションを選択すると、ウィザードは以下のプロパティを各オブジェクトタイプ・クラスに追加します。

    ```
    Property %identity As %XML.Oid (XMLNAME="_identity", XMLPROJECTION="ATTRIBUTE") [Transient];
    ```

  - **[GUID使用]** ー このオプションを選択すると、ウィザードは以下のプロパティを各オブジェクトタイプ・クラスに追加します。

    ```
    Property %identity As %XML.GUID (XMLNAME="_identity", XMLPROJECTION="ATTRIBUTE") [Transient];
    ```

  "特殊なトピック" も参照してください。
- 下部のテーブルには、スキーマに含まれる XML ネームスペースがリストされます。ここでは、その行に表示された XML ネームスペースのクラスを格納するパッケージを指定します。これを行うには、その行の **[パッケージ名]** フィールドにパッケージ名を指定します。

7. **[次へ]** を選択します。

8. 次の画面で、以下のオプションを指定します。
   - **[Java有効]** ― このオプションを選択すると、各クラスに Java プロジェクションが含まれます。
   - **[データ生成]** ― このオプションを選択すると、各クラスは **%XML.Adaptor** に加えて **%Populate** を拡張します。
   - **[SQLカラム順序]** ― このオプションを選択すると、各プロパティは、プロパティのスキーマ内の順序と SQL 内の順序が同じになるよう、SqlColumnNumber キーワードの値を指定します。
   - **[シーケンスの非チェック]** ― このオプションにチェックを付けると、ウィザードは生成されたクラスで XMLSEQUENCE パラメータを 0 に設定します。このオプションは、XML ファイルが XML スキーマと同じ順序で要素を持たない状況で役立ちます。

     既定では、生成されたクラス内で XMLSEQUENCE パラメータは 1 に設定されています。これにより、プロパティが確実にスキーマ内の順序と同じ順序でクラス定義に含まれます。"特殊なトピック" も参照してください。
   - **XMLIGNORENULL** ― このオプションを選択した場合、ウィザードにより `XMLIGNORENULL=1` がクラス定義に追加されます。それ以外の場合、このパラメータの追加はありません。

     このクラス・パラメータの詳細は、"空文字列および NULL 値の処理" を参照してください。
   - **バイナリにストリームを使用** ― このオプションを選択すると、ウィザードは、`xsd:base64Binary` タイプのあらゆる要素に **%Stream.GlobalBinary** タイプのプロパティを生成します。このオプションを選択しないと、プロパティのタイプが **%xsd.base64Binary** になります。

     ウィザードは、タイプが `xsd:base64Binary` の属性を無視する点に注意してください。
   - チェック・ボックスの下のテーブルには、ウィザードにより生成されるクラスがリストされます。各クラスに、**[拡張/タイプ]** が適切に設定されていることを確認します。ここでは以下のいずれかを選択します。
     - **[永続]** ― このオプションを選択すると、そのクラスは永続クラスになります。
     - **[シリアル]** ― このオプションを選択すると、そのクラスはシリアル・クラスになります。
     - **[登録オブジェクト]** ― このオプションを選択すると、そのクラスは登録オブジェクト・クラスになります。

     生成されたすべてのクラスは、**%XML.Adaptor** も拡張します。
   - テーブルの右の列で、インデックス化する必要がある各プロパティの **[インデックス]** を選択します。
9. **[完了]** を選択します。

ウィザードはクラスを生成し、要求に応じてこれらのクラスをコンパイルします。

これらのクラスのプロパティについては、スキーマ内の対応する要素の名前がアンダースコア (_) で始まる場合、そのプロパティの名前はパーセント記号 (%) で始まります。

14.2 プログラムによるクラスの生成

XML スキーマ・ウィザードは **%XML.Utils.SchemaReader** クラスの Process() メソッドとしても使用できます。このメソッドを使用するには、以下の操作を実行します。

1. **%XML.Utils.SchemaReader** のインスタンスを作成します。
2. 必要に応じて、このインスタンスの動作を制御するプロパティを設定します。詳細は、**%XML.Utils.SchemaReader** のクラス・ドキュメントを参照してください。
3. オプションとして、追加設定についての情報を含める InterSystems IRIS 多次元配列を作成します。詳細は、**%XML.Utils.SchemaReader** のクラス・ドキュメントの Process() メソッドを参照してください。

4. インスタンスの Process() メソッドを呼び出します。

```
method Process(LocationURL As %String,
               Package As %String = "Test",
               ByRef Features As %String) as %Status
```

- LocationURL は、スキーマの URL にするか、スキーマ・ファイルの名前 (完全なパスを含む) にする必要があります。
- Package は、生成されたクラスを配置するパッケージの名前です。パッケージを指定しないと、InterSystems IRIS は、サービス名をパッケージ名として使用します。
- Features は、前の手順で必要に応じて作成した多次元配列です。

14.3 既定のデータ型

XML スキーマ・ウィザードは、生成したプロパティごとに、スキーマに指定されている XSD タイプに応じて、適切な InterSystems IRIS データ型クラスを自動的に使用します。以下のテーブルに、XSD タイプおよび対応する InterSystems IRIS データ型を示します。

テーブル 14-1: XML タイプに使用される InterSystems IRIS データ型

ソース・ドキュメントの XSD タイプ	生成された InterSystems IRIS クラスのデータ型
anyURI	%xsd.anyURI
base64Binary	%xsd.base64Binary または %Stream.GlobalBinary。選択したオプションによって異なります。各文字列が文字列長の制限を超えるかどうかを開発者が確認する必要があります。制限を超える場合は、生成されたプロパティを %xsd.base64Binary から適切なストリーム・クラスに変更する必要があります。
boolean	%Boolean
byte	%xsd.byte
date	%Date
dateTime	%TimeStamp
decimal	%Numeric
double	%xsd.double
float	%xsd.float
hexBinary	%xsd.hexBinary
int	%xsd.int
integer	%Integer
long	%Integer
negativeInteger	%xsd.negativeInteger
nonNegativeInteger	%xsd.nonNegativeInteger
nonPositiveInteger	%xsd.nonPositiveInteger

ソース・ドキュメントの XSD タイプ	生成された InterSystems IRIS クラスのデータ型
positiveInteger	%xsd.positiveInteger
short	%xsd.short
string	%String (メモ : 各文字列が文字列長の制限を超えるかどうかを開発者が確認する必要があります。制限を超える場合は、生成されたタイプを適切なストリーム・クラスに変更する必要があります。)
time	%Time
unsignedByte	%xsd.unsignedByte
unsignedInt	%xsd.unsignedInt
unsignedLong	%xsd.unsignedLong
unsignedShort	%xsd.unsignedShort
タイプを指定しない場合	%String

14.4 生成されたプロパティのプロパティ・キーワード

XML スキーマ・ウィザードは、生成したプロパティごとに、スキーマ内の情報を使用して、以下のキーワードも自動的に設定します。

- 説明
- 必須
- ReadOnly (対応する要素または属性が、`fixed` 属性で定義されている場合)
- InitialExpression (この値は、スキーマ内の `fixed` 属性から取り込まれます)
- リレーションシップに関連するキーワード

14.5 生成されたプロパティのパラメータ

XML スキーマ・ウィザードは、生成したプロパティごとに、XMLNAME や XMLPROJECTION など、XML に関連するパラメータすべてを必要に応じて自動的に設定します。これらの詳細は、"オブジェクトの XML への投影" を参照してください。また、MAXVAL、MINVAL、VALUELIST など、他のパラメータも必要に応じて設定されます。

14.6 生成されたクラスをきわめて長い文字列に合わせて調整する方法

まれに、文字列長の制限を超える、きわめて長い文字列やバイナリ値に対応できるように、生成されたクラスの編集が必要な場合があります。

どのような文字列タイプでも、XML スキーマには、文字列がどのくらいの長さになる可能性があるかを示す情報は含まれません。XML スキーマ・ウィザードでは、文字列の任意の値が InterSystems IRIS %String クラスにマップされ、base64Binary の任意の値が %xsd.base64Binary クラスにマップされます。クラスが保持することになっているデータによっては、これらの選択肢が不適切になる場合があります。

生成されたクラスを使用する前に、以下を行う必要があります。

- 生成されたクラスを調べて、%String または %xsd.base64Binary と定義されているプロパティを見つけます。これらのクラス、特にこれらのプロパティを使用するコンテキストを検討します。
- 文字列長の制限を超える文字列を %String プロパティに含めることが必要と判断した場合、そのプロパティを適切な文字ストリームに再定義します。同様に、同じ制限を超える文字列を %xsd.base64Binary プロパティに含めることが必要と判断した場合、そのプロパティを適切なバイナリ・ストリームに再定義します。
- プロパティのタイプが %String、%xsd.string、および %Binary の場合、MAXLEN プロパティ・パラメータは既定で 50 文字であることにも注意してください。正しい検証ができるように、上限をさらに高く指定することが必要な場合もあります。

  (プロパティのタイプが %xsd.base64Binary の場合、MAXLEN は "" になります。これは、検証の際に長さがチェックされないことを意味します。ただし、文字列長の制限は適用されます。)

15
クラスからの XML スキーマの生成

このトピックでは、%XML.Schema を使用して、XML 対応クラスから XML スキーマを生成する方法を説明します。

15.1 概要

同じ XML ネームスペース内の複数のクラス用のタイプを定義する完全なスキーマを生成するには、%XML.Schema を使用してスキーマを構築し、%XML.Writer を使用してその出力を生成します。

15.2 複数のクラスからのスキーマの構築

XML スキーマを構築する手順は次のとおりです。

1. %XML.Schema の新しいインスタンスを作成します。
2. オプションとして、インスタンスのプロパティを設定します。
   - ほかに割り当てられていないタイプのネームスペースを指定するには、DefaultNamespace プロパティを指定します。既定は Null です。
   - 既定では、クラスとそのプロパティのクラス・ドキュメントは、スキーマの <annotation> 要素に含まれます。これを無効にするには、IncludeDocumentation プロパティを 0 に指定します。

   注釈　AddSchemaType() メソッドを呼び出す前に、これらのプロパティを設定する必要があります。
3. インスタンスの AddSchemaType() メソッドを呼び出します。

```
method AddSchemaType(class As %String,
                     top As %String  = "",
                     format As %String,
                     summary As %Boolean  = 0,
                     input As %Boolean  = 0,
                     refOnly As %Boolean  = 0) as %Status
```

以下はその説明です。

- class は、XML 対応クラスのパッケージとクラスの完全な名前です。
- top はオプションです。これを指定すると、このクラスのタイプ名がオーバーライドされます。

- format では、このタイプの形式を指定します。これは、"literal" (リテラル形式、既定値)、"encoded" (SOAP エンコードの場合)、"encoded12" (SOAP 1.2 エンコードの場合)、または "element" である必要があります。値 "element" は、最上位レベルの要素を持つリテラル形式と同じです。
- summary が True の場合、InterSystems IRIS® データ・プラットフォームでは、XML 対応クラスの XMLSUMMARY パラメータが考慮されます。このパラメータが指定されている場合、スキーマには、そのパラメータで指定されているプロパティのみが含まれるようになります。
- input が True の場合、InterSystems IRIS では、出力スキーマではなく、入力スキーマが取得されます。ほとんどの場合、入力スキーマと出力スキーマは同じですが、クラスのプロパティに対して XMLIO プロパティ・パラメータが指定されている場合は、異なったものになります。
- refOnly が True の場合、InterSystems IRIS では、指定のクラスとすべての参照先タイプに対するスキーマではなく、参照先タイプに対するスキーマのみが生成されます。

このメソッドはステータスを返すので、それをチェックする必要があります。

4. 前述の手順を必要に応じて繰り返します。
5. オプションで、インポートされたスキーマの場所を定義するには、DefineLocation() メソッドを呼び出します。

```
method DefineLocation(namespace As %String, location As %String)
```

namespace は、1 つ以上の参照先クラスによって使用されるネームスペースです。location は、対応するスキーマ (XSD ファイル) の URL またはパスとファイル名です。

このメソッドを繰り返し呼び出して、インポートされた複数のスキーマの場所を追加することができます。

このメソッドを使用しない場合、スキーマには <import> 指示文が含まれますが、スキーマの場所は示されません。

6. オプションで、追加の <import> 指示文を定義するには、DefineExtraImports() メソッドを呼び出します。

```
method DefineExtraImports(namespace As %String, ByRef imports)
```

namespace は、<import> 指示文の追加先となるネームスペースです。imports は、以下の形式の多次元配列です。

ノード	値
arrayname("namespace URI")	このネームスペースに対するスキーマ (XSD ファイル) の場所を示す文字列。

15.3 スキーマの出力の生成

前のセクションで説明したように %XML.Schema のインスタンスを作成したら、以下の手順を実行して出力を生成します。

1. インスタンスの GetSchema() メソッドを呼び出し、ドキュメント・オブジェクト・モデル (DOM) のノードとしてスキーマを返します。

   このメソッドの引数は、スキーマのターゲットのネームスペースの URI のみです。このメソッドは、%XML.Node のインスタンスを返します。このことは、"XML ドキュメントの DOM 表現" で説明されています。

   スキーマにネームスペースが指定されていない場合、GetSchema() の引数として "" を使用します。

2. オプションとして、この DOM を変更します。
3. スキーマを生成する手順は以下のとおりです。

   a. %XML.Writer クラスのインスタンスを作成し、オプションで Indent などのプロパティを設定します。

b. オプションで、ライターの AddNamespace() などのメソッドを呼び出して、<schema> 要素にネームスペース宣言を追加します。"ネームスペース宣言の追加" を参照してください。

   多くの場合、スキーマは単純な XSD タイプを参照するため、AddSchemaNamespace() を呼び出して XML スキーマのネームスペースを追加することは有用です。

c. 引数としてスキーマを使用して、ライターの DocumentNode() メソッドまたは Tree() メソッドを呼び出します。

   これらのメソッドの使用方法の詳細は、"XML ドキュメントの DOM 表現" の "DOM からの XML 出力の記述" を参照してください。

15.4 例

その他の例は、クラスリファレンスで %XML.Schema を参照してください。

15.4.1 簡単な例

最初の例では、基本的な手順を示しています。

ObjectScript

```
Set schemawriter=##class(%XML.Schema).%New()

//add class and package (for example)
Set status=schemawriter.AddSchemaType("Facets.Test")

//retrieve schema by its URI
//which is null in this example
Set schema=schemawriter.GetSchema("")

//create writer
Set writer=##class(%XML.Writer).%New()
Set writer.Indent=1

//use writer
Do writer.DocumentNode(schema)
```

15.4.2 より複雑なスキーマの例

以下の例を考えてみます。Person クラスは以下のとおりです。

Class Definition

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

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

Property Name As %Name;

Property DOB As %Date(FORMAT = 5);

Property PatientID as %String;

Property HomeAddress as Address;

Property OtherAddress as AddressOtherNS ;
}
```

Address クラスは、同じ XML ネームスペース ("http://www.myapp.com") 内に存在するように定義されます。
OtherAddress クラスは、別な XML ネームスペース ("http://www.other.com") 内に存在するように定義されます。

Company クラスも、XML ネームスペース "http://www.myapp.com" 内に存在するように定義されます。以下のように定義されます。

Class Definition

```
Class SchemaWriter.Company Extends (%Persistent, %XML.Adaptor)
{

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

Property Name As %String;

Property CompanyID As %String;

Property HomeOffice As Address;

}
```

Person クラスと Company クラスを結び付けるプロパティ・リレーションシップはありません。

ネームスペース "http://www.myapp.com" に対するスキーマを生成するには、以下の構文を使用できます。

Class Member

```
ClassMethod Demo()
{
    Set schema=##class(%XML.Schema).%New()
    Set schema.DefaultNamespace="http://www.myapp.com"
    Set status=schema.AddSchemaType("SchemaWriter.Person")
    Set status=schema.AddSchemaType("SchemaWriter.Company")
    Do schema.DefineLocation("http://www.other.com","c:/other-schema.xsd")

    Set schema=schema.GetSchema("http://www.myapp.com")

    //create writer
    Set writer=##class(%XML.Writer).%New()
    Set writer.Indent=1

    Do writer.AddSchemaNamespace()
    Do writer.AddNamespace("http://www.myapp.com")
    Do writer.AddNamespace("http://www.other.com")

    Set status=writer.DocumentNode(schema)
    If $$$ISERR(status) {Do $system.OBJ.DisplayError() Quit }

}
```

出力は、以下のとおりです。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s01="http://www.myapp.com"
xmlns:s02="http://www.other.com"
elementFormDefault="qualified"
targetNamespace="http://www.myapp.com">
  <import namespace="http://www.other.com" schemaLocation="c:/other-schema.xsd"/>
  <complexType name="Person">
    <sequence>
      <element minOccurs="0" name="Name" type="s:string"/>
      <element minOccurs="0" name="DOB" type="s:date"/>
      <element minOccurs="0" name="PatientID" type="s:string"/>
      <element minOccurs="0" name="HomeAddress" type="s01:Address"/>
      <element minOccurs="0" name="OtherAddress" type="s02:AddressOtherNS"/>
    </sequence>
  </complexType>
  <complexType name="Address">
    <sequence>
      <element minOccurs="0" name="State">
        <simpleType>
          <restriction base="s:string">
            <maxLength value="2"/>
          </restriction>
        </simpleType>
      </element>
```

```
      <element minOccurs="0" name="Zip">
        <simpleType>
          <restriction base="s:string">
            <maxLength value="10"/>
          </restriction>
        </simpleType>
      </element>
    </sequence>
  </complexType>
  <complexType name="Company">
    <sequence>
      <element minOccurs="0" name="Name" type="s:string"/>
      <element minOccurs="0" name="CompanyID" type="s:string"/>
      <element minOccurs="0" name="HomeOffice" type="s01:Address"/>
    </sequence>
  </complexType>
</schema>
```

これから以下の点がわかります。

- スキーマには、Person とそのすべての参照先クラスに対するタイプが含まれています。また、Company とそのすべての参照先クラスに対するタイプも含まれています。
- <import> 指示文では、OtherAddress クラスによって使用されるネームスペースがインポートされます。DefineLocation() を使用したため、この指示文では、対応するスキーマの場所も示されます。
- DocumentNode() を呼び出す直前に AddSchemaNamespace() と AddNamespace() を使用したため、<schema> 要素には、ネームスペースの接頭語を定義するネームスペース宣言が含まれます。

  AddSchemaNamespace() と AddNamespace() を使用しない場合は、<schema> にそれらのネームスペース宣言が含まれることはなく、スキーマは以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
targetNamespace="http://www.myapp.com">
  <import namespace="http://www.other.com" schemaLocation="c:/other-schema.xsd"/>
  <complexType name="Person">
    <sequence>
      <element minOccurs="0" name="Name" type="s01:string"
xmlns:s01="http://www.w3.org/2001/XMLSchema"/>
     <element minOccurs="0" name="DOB" type="s02:date" xmlns:s02="http://www.w3.org/2001/XMLSchema"/>

      <element minOccurs="0" name="PatientID" type="s03:string"
xmlns:s03="http://www.w3.org/2001/XMLSchema"/>
     <element minOccurs="0" name="HomeAddress" type="s04:Address" xmlns:s04="http://www.myapp.com"/>

      <element minOccurs="0" name="OtherAddress" type="s05:AddressOtherNS"
xmlns:s05="http://www.other.com"/>
    </sequence>
  </complexType>
  <complexType name="Address">
    <sequence>
      <element minOccurs="0" name="State">
        <simpleType>
          <restriction base="s06:string" xmlns:s06="http://www.w3.org/2001/XMLSchema">
...
```

16
ネームスペースとクラスの検証

%XML.Namespaces クラスは、2 つのクラス・メソッドを提供します。これらは、XML ネームスペースとそこに含まれるクラスを検証するのに使用できます。

GetNextClass()

```
classmethod GetNextClass(namespace As %String,
           class As %String) as %String
```

指定した XML ネームスペース内の、指定したクラスの (アルファベット順の) 次のクラスを返します。これ以上クラスがない場合は、このメソッドは NULL を返します。

GetNextNamespace()

```
classmethod GetNextNamespace(namespace As %String) as %String
```

指定したネームスペースの (アルファベット順の) 次のネームスペースを返します。これ以上ネームスペースがない場合は、このメソッドは NULL を返します。

どちらの場合でも、現在の InterSystems IRIS® データ・プラットフォーム・ネームスペースのみが考慮されます。また、マップされたクラスは無視されます。

例えば、以下のメソッドは、現在の InterSystems IRIS ネームスペースの XML ネームスペースとそのクラスをリストアップします。

Class Member

```
ClassMethod WriteNamespacesAndClasses()
{
  Set ns=""
  Set ns=##class(%XML.Namespaces).GetNextNamespace(ns)

  While ns '=""
  {
    Write !, "The namespace ",ns, " contains these classes:"
    Set cls=""
    Set cls=##class(%XML.Namespaces).GetNextClass(ns,cls)

    While cls '=""
    {
      Write !, "   ",cls
      Set cls=##class(%XML.Namespaces).GetNextClass(ns,cls)
      }

      Set ns=##class(%XML.Namespaces).GetNextNamespace(ns)
      }
}
```

ターミナルで実行すると、このメソッドは以下のような出力を生成します。

```
The namespace http://www.address.org contains these classes:
   ElRef.NS.Address
   GXML.AddressNS
   MyApp4.Obj.Address
   MyAppNS.AddressNS
   Obj.Attr.Address
   Obj.Ns.Address
   Obj.Ns.AddressClass
The namespace http://www.doctor.com contains these classes:
   GXML.DoctorNS
The namespace http://www.one.org contains these classes:
   GXML.AddressNSOne
   GXML.DoctorNSOne
   GXML.PersonNSOne
...
```

17

XML 標準

InterSystems IRIS® データ・プラットフォームでの XML サポートは、以下の規格に従います。

- XML 1.0 (https://www.w3.org/TR/REC-xml/)
- XML 1.0 のネームスペース (https://www.w3.org/TR/REC-xml-names/)
- XML Schema 1.0 (https://www.w3.org/TR/xmlschema-0/、https://www.w3.org/TR/xmlschema-1/、https://www.w3.org/TR/xmlschema-2/)
- https://www.w3.org/TR/xpath で指定された XPath 1.0
- SOAP 1.1 規格のセクション 5 で指定された SOAP 1.1 エンコード。
- SOAP 1.2 規格のセクション 3 パート 2 : Adjuncts (https://www.w3.org/TR/soap12-part2/) で指定された SOAP 1.2 エンコード。

  SOAP の詳細は、W3 の Web サイト (例えば、https://www.w3.org/TR/2003/REC-soap12-part1-20030624/) を参照してください。
- https://www.w3.org/TR/xml-c14n で指定された XML Canonicalization バージョン 1.0 (包含的な正規化とも呼ばれる)。
- https://www.w3.org/TR/xml-exc-c14n/ で指定された XML Exclusive Canonicalization バージョン 1.0 (Inclusive-Namespaces PrefixList 機能 (https://www.w3.org/TR/xml-exc-c14n/#def-InclusiveNamespaces-PrefixList) を含む)
- XML 暗号化 (https://www.w3.org/TR/xmlenc-core/)

  InterSystems IRIS は、RSA-OAEP または RSA-1.5 を使用したキー暗号化、および AES-128、AES-192、または AES-256 を使用したメッセージ本文のデータ暗号化をサポートしています。
- Exclusive XML Canonicalization と RSA SHA-1 を使用した XML シグニチャ (https://www.w3.org/TR/xmldsig-core/)

InterSystems IRIS SAX パーサでは、XML 1.0 勧告に準拠する標準 Xerces-C++ ライブラリが使用されます。これらの標準のリストについては、http://xml.apache.org/xerces-c/ を参照してください。

InterSystems IRIS は、以下の 2 つの XSLT プロセッサを提供します。

- Xalan プロセッサは XSLT 1.0 をサポートします。
- Saxon プロセッサは XSLT 2.0 をサポートします。

"SOAP 標準" および "SOAP セキュリティ標準" も参照してください。

XML で想定される文字セットの詳細は、W3 Web サイト (https://www.w3.org/TR/2006/REC-xml-20060816/#charsets) を参照してください。

注釈　InterSystems IRIS では、1 つの要素内でそれぞれ別のネームスペースにある複数の同名属性をサポートしていません。

A

XML の背景

ここでは、InterSystems IRIS® データ・プラットフォームの %XML クラスを使用するユーザのための補足資料として、XML の用語や概念の概要を説明します。このトピックは、読者が XML の基本的な構文に精通していることを前提にしており、XML の初級解説ではありません。例えば、構文や予約文字のリストについては説明しません。ただし、参考資料として、用語とその簡単な定義のリストを示します。

attribute

名前と値の組み合わせで、形式は以下のとおりです。

```
ID="QD5690"
```

以下に示すように、属性は要素内に存在し、要素には任意の数の属性を含めることができます。

XML

```
<Patient ID="QD5690">Cromley,Marcia N.</Patient>
```

CDATA セクション

検証する必要のないテキストを、以下のように示します。

XML

```
<myelementname>
Non-validated data goes here.
You can even have stray "<" or ">" symbols in it.


</myelementname>
```

CDATA (文字データ) セクションには文字列 `]]>` を含めることはできません。これは、この文字列がセクションの終了を表すためです。つまり、CDATA セクションは入れ子にできないという意味もあります。

CDATA セクションの内容は、XML ドキュメントの残りの部分で指定されているように、XML ドキュメントに対して指定されているエンコードに従う必要があります。

comment

XML ドキュメントのメイン・データを構成しない、挿入されたメモ。コメントは、以下のようになります。

```
<!--Output for the class: GXML.PersonNS7-->
```

コンテンツ・モデル

XML 要素の利用可能なコンテンツに関する抽象的な説明です。利用可能なコンテンツ・モデルは、以下のとおりです。

- 空のコンテンツ・モデル (子要素やテキスト・ノードは不許可)
- 単純なコンテンツ・モデル (テキスト・ノードのみ許可)
- 複雑なコンテンツ・モデル (子要素のみ許可)
- 複合コンテンツ・モデル (子要素とテキスト・ノードの両方を許可)

いずれの場合も、要素は、属性を含んでいることも、含んでいないこともあります。コンテンツ・モデルは、要素内に属性があるかないかに言及するものではありません。

既定のネームスペース

任意のコンテキストで、未修飾の要素が属するネームスペース。既定のネームスペースは、接頭語を付けずに追加されます。以下はその例です。

XML

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

このネームスペースの宣言では接頭語を使用しないので、<Person>、<Name>、および <DOB> 要素はすべてこのネームスペースに属します。

以下の XML は既定のネームスペースを使用しないので、事実上前述の例と等しくなることに注意してください。

XML

```
<s01:Person s01:xmlns="http://www.person.org">
  <s01:Name>Isaacs,Rob G.</s01:Name>
  <s01:DOB>1981-01-29</s01:DOB>
</s01:Person>
```

DOM

ドキュメント・オブジェクト・モデル (DOM) は XML と関連する形式を表すためのオブジェクト・モデルです。

DTD (ドキュメント・タイプの定義)

XML ドキュメントまたは外部ファイル内に含まれる、一連のテキスト指示文。ドキュメント内で使用できるすべての要素および属性を定義します。DTD 自体は XML 構文を使用しません。

element

要素は通常、2 つのタグ (開始タグと終了タグ) で構成され、テキストおよびその他の要素を囲む場合があります。要素のコンテンツは、これら 2 つのタグの間にあるすべてのもので、テキストや子要素 (存在する場合) が含まれます。以下は、開始タグ、テキスト・コンテンツ、および終了タグを含む完全な XML 要素です。

XML

```
<Patient>Cromley,Marcia N.</Patient>
```

要素には、任意の数の属性および任意の数の子要素を含めることができます。

空白の要素には、開始タグと終了タグを含めるか、タグを 1 つだけ含めることができます。以下の例は同じものです。

```
<EndDate></EndDate>
```

```
<EndDate/>
```

実際には、以下のように要素はデータ・レコードの異なる部分を参照する傾向があります。

XML

```
<Student level="undergraduate">
   <Name>Barnes,Gerry</Name>
   <DOB>1981-04-23</DOB>
</Student>
```

entity

1 つ以上の文字を表す (XML ファイル内の) テキストの単位です。エンティティの構造は以下のとおりです。

```
&characters;
```

グローバル要素

グローバル要素およびローカル要素の概念は、ネームスペースを使用するドキュメントに適用されます。グローバル要素の名前は、ローカル要素の名前とは別のシンボル・スペースに配置されます。グローバル要素は、タイプにグローバル・スコープを含む要素です。つまり、そのタイプが、対応する XML スキーマの最上位レベルで定義される要素です。`<xs:schema>` 要素の子として表示される要素宣言は、グローバル宣言と見なされます。`ref` 属性を通してグローバル宣言を参照することによって実質的にグローバル要素にしない限り、その他の要素宣言はローカル要素です。

属性も同様にグローバルまたはローカルになります。

ローカル要素

グローバルではない XML 要素。ローカル要素は、要素が修飾されない限り、明示的にどのネームスペースにも属しません。修飾およびグローバル要素を参照してください。

ネームスペース

ネームスペースは識別子のドメインを定義する一意の文字列であり、XML ベースのアプリケーションで異なるタイプのドキュメント同士が混同されることを防止します。一般的には、URL (ユニフォーム・リソース・ロケーション) の形式の URI (ユニフォーム・リソース・インジケータ) として指定されます。これは、実際の Web アドレスには対応する場合もしない場合もあります。例えば、`"http://www.w3.org"` はネームスペースです。

以下の構文のいずれかを使用して、ネームスペース宣言を含めます。

```
xmlns="your_namespace_here"
pre:xmlns="your_namespace_here"
```

いずれの場合も、ネームスペースはネームスペース宣言を挿入したコンテキスト内でのみ使用します。後者の場合、ネームスペースは指定された接頭語 (pre) に関連付けられます。要素または属性にこの接頭語が含まれる場合にのみ、その要素または属性はこのネームスペースに属します。以下はその例です。

XML

```
<s01:Person xmlns:s01="http://www.person.com">
   <Name>Ravazzolo,Roberta X.</Name>
   <DOB>1943-10-24</DOB>
</s01:Person>
```

このネームスペース宣言では、s01 の接頭語を使用します。<Person> 要素もこの接頭語を使用するので、この要素はこのネームスペースに属します。ただし、<Name> 要素および <DOB> 要素は、明示的にどのネームスペースにも属しません。

処理命令 (PI)

XML ドキュメントの使用方法または XML ドキュメントを使用して実行する内容をアプリケーションに指示するための、(プロローグ内の) 命令。例は以下のとおりです。これは、スタイルシートとドキュメントを関連付けます。

```
<?xml-stylesheet type="text/css" href="mystyles.css"?>
```

プロローグ

XML ドキュメントのルート要素の前の部分。プロローグは XML 宣言 (使用される XML バージョンを示します) で始まり、次に処理命令および、DTD 宣言またはスキーマ宣言を含む場合があります (実際には、DTD もスキーマも不要です。また、実際には、同じファイルに両方を含めることもできます)。

ルート、ルート要素、ドキュメント要素

各 XML ドキュメントには、最も外側のレベルに要素を 1 つだけ含める必要があります。これは、ルート、ルート要素、またはドキュメント要素と呼ばれます。ルート要素はプロローグの後に位置します。

修飾

要素または属性がネームスペースに明示的に割り当てられる場合、その要素または属性は修飾されます。以下の例を考えてみます。ここでは、<Person> の要素および属性は修飾されていません。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <s01:Person xmlns:s01="http://www.person.com" GroupID="J1151">
      <Name>Frost,Sally O.</Name>
      <DOB>1957-03-11</DOB>
   </s01:Person>
</Root>
```

ここでは、ネームスペース宣言で s01 の接頭語を使用します。既定のネームスペースはありません。<Person> 要素もこの接頭語を使用するので、この要素はこのネームスペースに属します。<Name> 要素および <DOB> 要素、または <GroupID> 属性には接頭語がないので、明示的にどのネームスペースにも属しません。

一方、以下の場合を考えてみます。ここでは、<Person> の要素および属性が修飾されています。

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
   <Person xmlns="http://www.person.com" GroupID="J1151">
      <Name>Frost,Sally O.</Name>
      <DOB>1957-03-11</DOB>
   </Person>
</Root>
```

この場合、<Person> 要素は既定のネームスペースを定義し、そのネームスペースは子要素および属性に適用されます。

注釈　XML スキーマ属性である elementFormDefault 属性と attributeFormDefault 属性は、要素と属性が指定されたスキーマで修飾されるかどうかを制御します。InterSystems IRIS XML サポートでは、クラス・パラメータを使用して、要素が修飾されるかどうかを指定します。

スキーマ

DTD の代わりに、一連の XML ドキュメントのメタ情報を指定するドキュメント。DTD では、スキーマを使用して特定の XML ドキュメントのコンテンツを検証できます。XML スキーマは、以下のような特定の用途に対して、DTD を上回る利点がいくつかあります。

- XML スキーマは、有効な XML ドキュメントであり、スキーマ上で操作するツールをより簡単に開発できます。
- XML スキーマは、より豊富な一連の機能を指定でき、値の型情報を取り込みます。

形式的には、スキーマ・ドキュメントは W3 XML スキーマの仕様 (https://www.w3.org/XML/Schema) に準拠する XML ドキュメントです。スキーマ・ドキュメントは、XML の規則に従っており、いくつかの追加構文を使用します。通常、ファイルの拡張子は .xsd です。

スタイル・シート

任意の XML ドキュメントを別の XML または人間が読めるその他のドキュメントに変換する方法を説明する、XSLT で記述されたドキュメントです。

テキスト・ノード

開始要素とそれに対応する終了要素の間に含まれる 1 つ以上の文字。以下はその例です。

```
<SampleElement>
sample text node
</SampleElement>
```

type

データの解釈で課される制約。XML スキーマでは、各要素と属性の定義は、タイプに対応しています。

タイプは、単純か複雑のいずれかです。

各属性には単純なタイプがあります。さらに、単純なタイプは、属性も子要素も持たない (テキスト・ノードのみを持つ) 要素を表します。複雑なタイプは、それ以外の要素を表します。

以下のスキーマのフラグメントは、タイプ定義をいくつか示しています。

```
<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: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>
```

未修飾

要素または属性がネームスペースに明示的に割り当てられない場合、その要素または属性は未修飾です。修飾を参照してください。

適格な XML

開始タグに一致する終了タグがあるなど、XML の規則に従う XML ドキュメントまたはフラグメント。

XML 宣言

任意のドキュメントで使用される XML のバージョン（およびオプションで文字セット）を示す文。これを含める場合は、ドキュメントの最初の行に置く必要があります。以下に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
```

XPath

XPath（XML Path Language）は、XML ドキュメントからデータを取得するための XML ベースの式言語です。結果は、スカラまたは元のドキュメントの XML サブツリーのいずれかです。

XSLT

XSLT（Extensible Stylesheet Language Transformations）は XML ベースの言語であり、これを使用して指定の XML ドキュメントを別の XML ドキュメントまたは人間が読める別の形式のドキュメントに変換する方法を記述します。