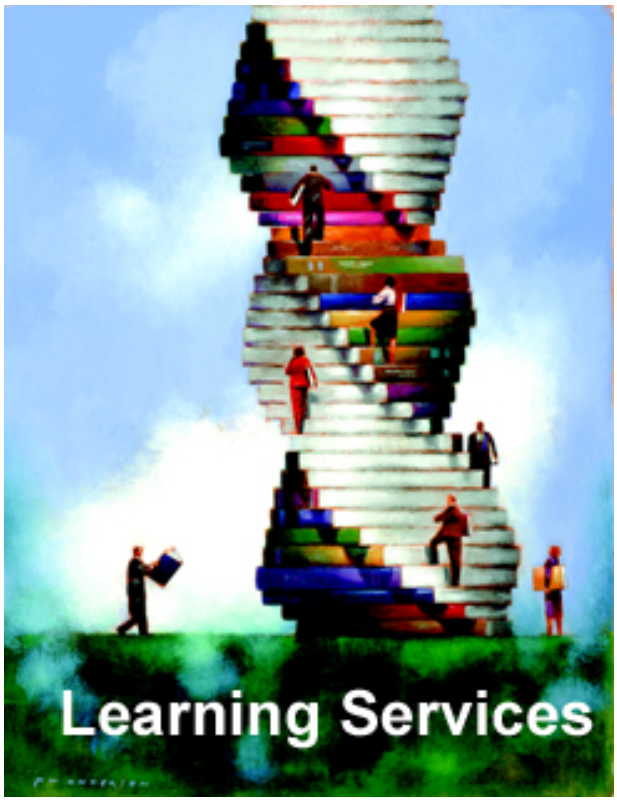

Caché XML ガイド

（Caché Version 2015.1 ベース）

V1.0

2015 年 4 月

インターシステムズジャパン株式会社

目次:

図表目次:

1. はじめに

本ガイドは、高性能オブジェクトデータベース Caché の XML 機能に関する入門書です。
本ガイドは、Cachéファーストステップガイド[1] を読み終えた方を対象としています。
まだ読んでいない方は、まず Caché ファーストステップガイドを読むことをお勧めします。

以下の説明では、作業場所として Caché をインストールすると標準で作成される USER ネームスペースを使用することを前提としています。もちろん適当なネームスペースを作成して作業を行うこともできます。ネームスペースの設定方法については、Caché ファーストステップガイドを参照して下さい。

本ガイドを習得すれば、Caché XML 機能をお使いいただく上において必要十分な知識が得られます。さらに詳しい情報については、オンライン・ドキュメント等をご参照ください。

また、本ガイドでは、XML とは何であるかということについての説明はしません。XML についての知識がなくても、このガイドを読み進めることは可能ですが、XML については、XML に関する解説書、インターネット上の情報等を合わせて参照戴くことでより理解が深まると思います。

[1] Caché ファーストステップガイドも含め、その他ガイドはこちらからダウンロードいただけます。

2. 簡単な HTTP 経由での XML 出力

Caché を使った簡単な XML 出力を行ってみましょう。Caché には、様々なデバイス（端末、ファイル、ソケットなど）に出力する機能が備わっていますが、ここでは、HTTP 経由でブラウザに出力する例を考えてみます。デバイスを切り替えることで同じ内容の出力先を変えることができます。

2.1. XML を出力するクラスの作成

Caché スタジオを起動し、メニューから新規作成を選び、表示される新規ウィザードの一般タブを選択し、Caché クラス定義を選びます。以下の新規クラスウィザードでパッケージ名、クラス名に適当な名前を入力します。（以下の例では、パッケージ名 XML、クラス名 Server）

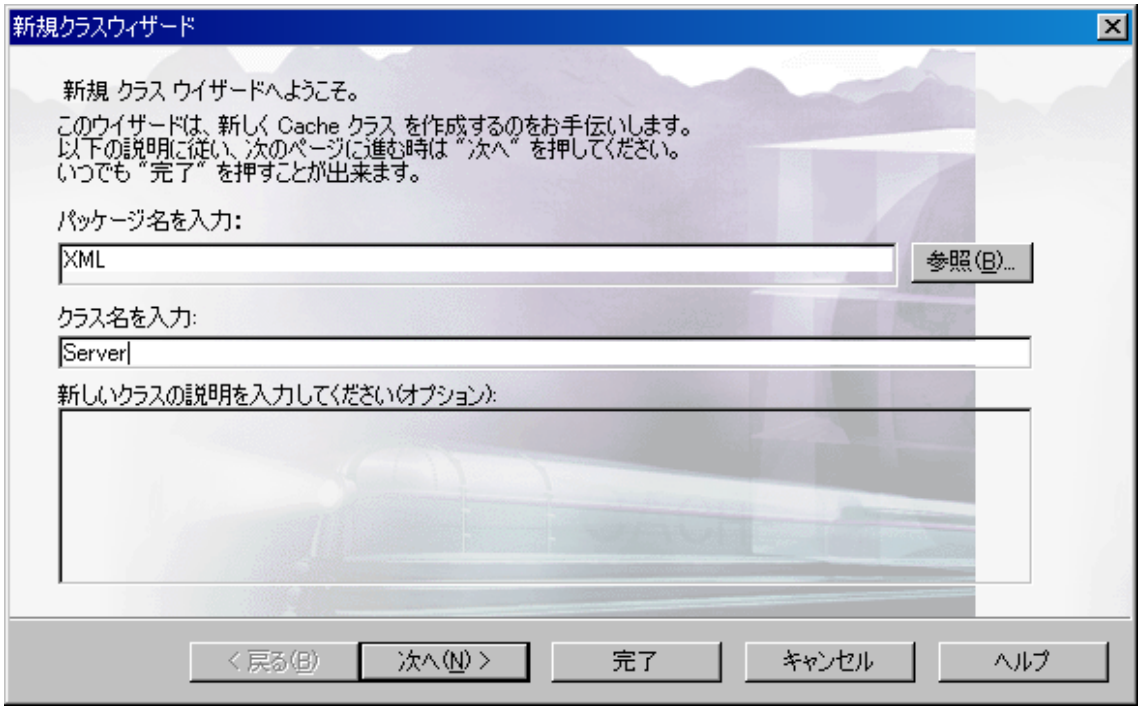

図 1 新規クラスウィザード：XML.Server クラスの作成

以下の様に CSP クラスタイプを選択して下さい。

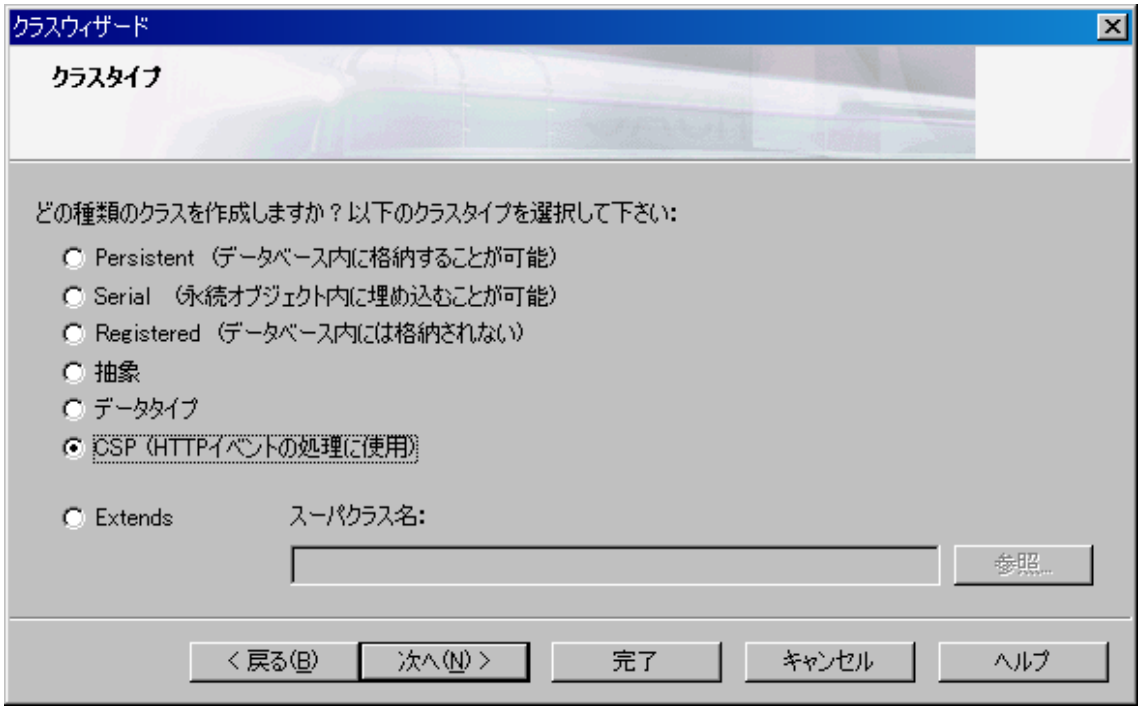

図 2 クラスタイプの設定:CSP

CSP が転送するコンテンツタイプに XML を指定します。

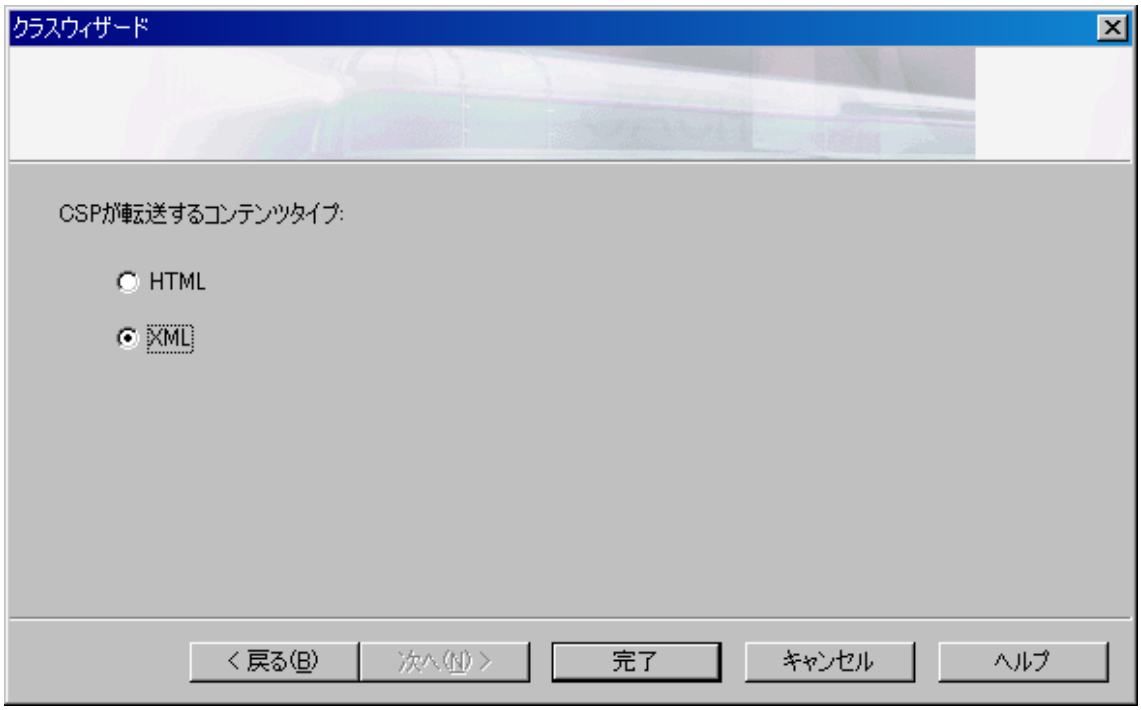

図 3 CSP が転送するコンテンツタイプの設定:XML

表示される編集ペインの OnPage()クラスメソッドに以下のコードを追加して下さい。

```
ClassMethod OnPage() As %Status
{
        Write "<?xml version=""1.0"" ?>",!
        Write "<Company>",!
        Write "<Name>インターシステムズジャパン</Name>",!
        Write "<Mission>キャシェの販売、サポート</Mission>",!
        Write "</Company>",!
        Quit $$$OK
}
```

図 4 XML.Server クラスの OnPage()メソッド

XML.Server.cls

```
1  Class XML.Server Extends %CSP.Page
2  [
3
4  ClassMethod OnPage() As %Status
5    {
6        Write "<?xml version=""1.0"" ?>",!
7        Write "<Company>",!
8        Write "<Name>インターシステムズジャパン</Name>",!
9        Write "<Mission>キャシェの販売、サポート</Mission>",!
10       Write "</Company>",!
11       Quit $$$OK
12   ]}
13
14 Parameter CONTENTTYPE = "text/xml";
15
16 }
17
```

図 5 XML.Server クラス全体

コンパイルを実行します。

2.2. XML を出力する

それでは、作成した CSP ページをウェブブラウザから表示させて見ましょう。

urlに http://localhost:57772/csp/user/XML.Server.clsを指定してオープンします。

以下の様に表示されるはずです。

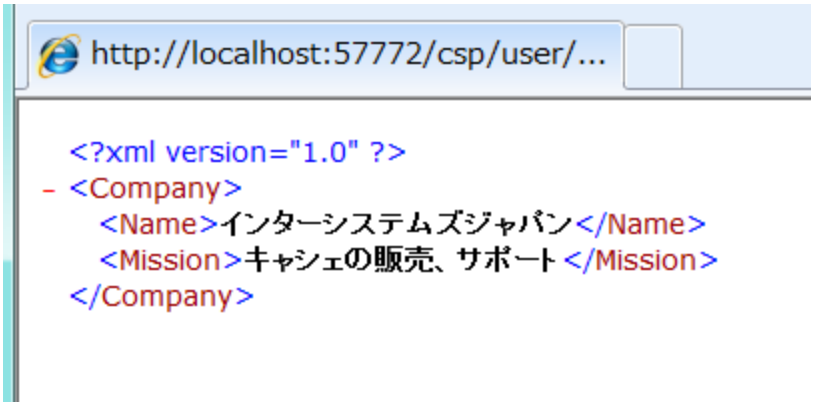

図 6 XML.Server:出力結果

＜この章で作成したクラス定義のサンプルは Step1 フォルダにあります。(XMLServer.xml)＞

3. XML のインポート、エクスポート

3.1. Caché クラスインスタンスを XML にエクスポート

3.1.1. クラスを XML 用に定義する

永続 Caché クラスのインスタンスを XML としてエクスポートするには、そのクラスは、通常の%Persistent クラスを継承すると共に、%XML.Adaptor クラスを継承する必要があります。
それでは、クラスを作成していきましょう。クラス作成に先立ち、step1 フォルダの step1.xml をスタジオのメニューで「ツール」→「ローカルからインポート」によりインポートし、コンパイルして下さい。
次にスタジオの新規作成をクリックし、新規作成ウィンドウから一般タブを選択し、Caché クラス定義をクリックします。以下の様にパッケージ名、クラス名を適当に入力します。

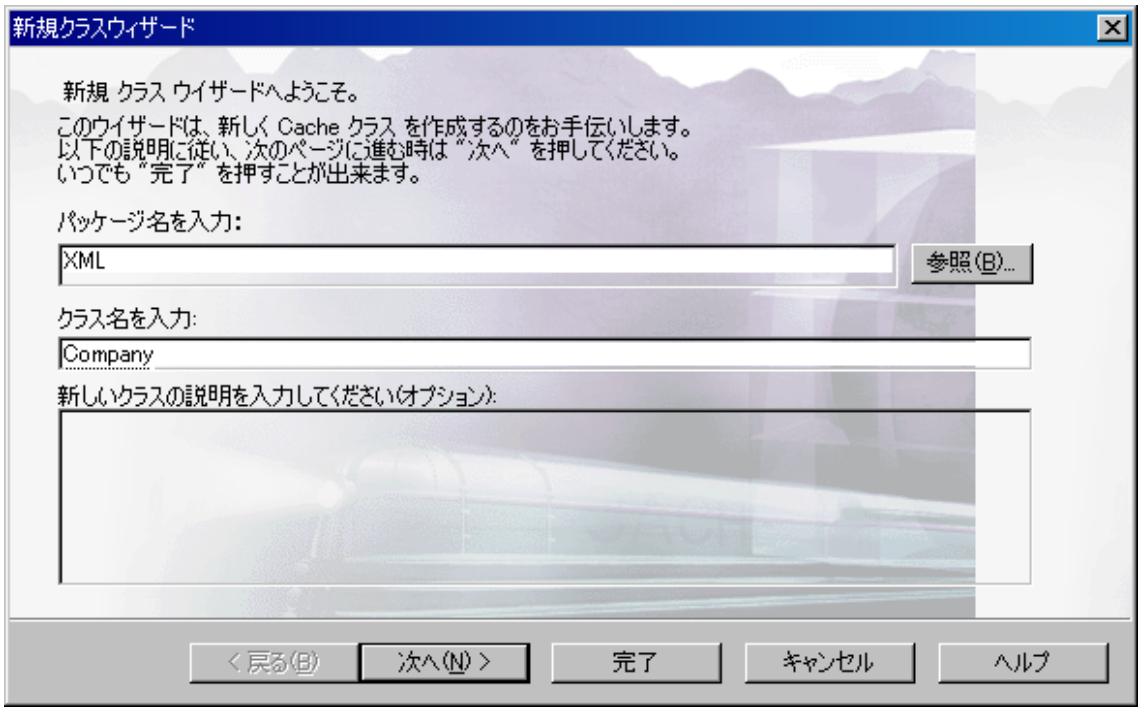

図 7 XML.Company の作成

"次へ"のボタンを押下し、次ページへ移動します。

クラスタイプを%Persistent にします。

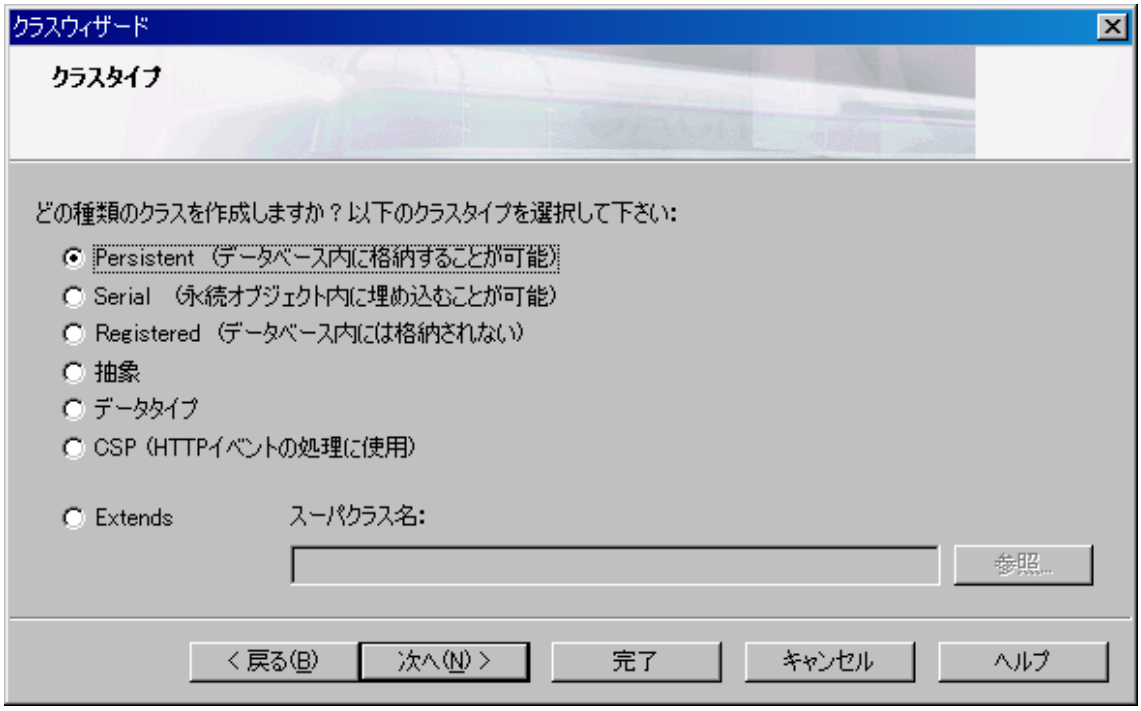

図 8 クラスタイプ: %Persistent の定義(ウィザード)

“次へ”ボタンを押下し、次のウィザードウィンドウにて、「XML 有効」と、「データ生成」 の両方をチェックします。

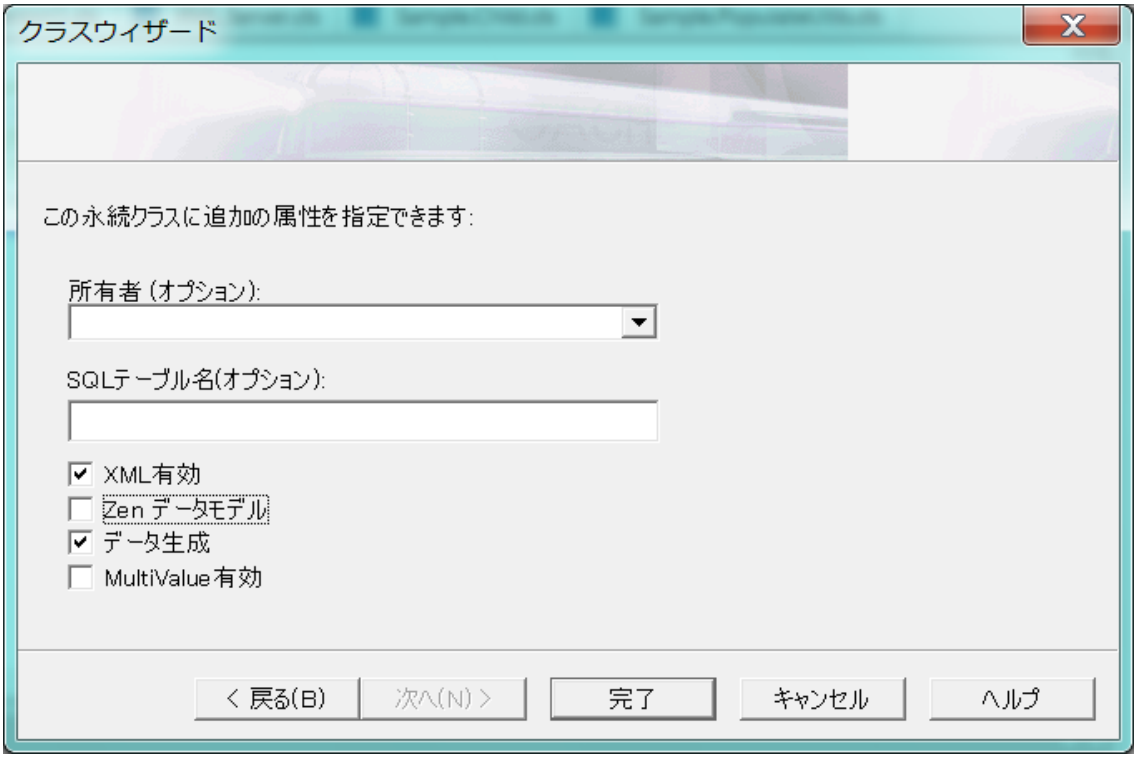

図 9 新規クラスウィザード:XML 有効、データ生成の設定

上記ウィザードで生成されたクラス定義に以下の様に 2 つのプロパティを追加し,コンパイルしてください。(Name プロパティと City プロパティ)

```
Class XML.Company Extends (%Persistent, %Populate, %XML.Adaptor)
{
Property Name As %String (POPSPEC = "##Class(Sample.PopulateUtils).Company()");
Property City As %String (POPSPEC = "##Class(Sample.PopulateUtils).City() ");
}
```

図 10 XML.Company:クラス定義

次に、Caché ターミナルを起動し、以下のコマンドを入力、実行してください。

```
Do ##class(XML.Company).Populate(100)
```

図 11 XML.Company:データ生成

このコマンド実行後、100 件のテストデータが生成されます。

3.1.2. クラスインスタンスを XML エクスポートする。

クラスインスタンスを XML としてエクスポートするには、%XML.Adaptor クラスを継承したクラスが持つ XMLExport()メソッドを使用します。
XML.Company クラスにクラスメソッド xmlCompExport()を作成します。

```
ClassMethod xmlCompExport()
{
        Read  "XML 形式で XML.Company データをエクスポートします",!,"ファイル名を指定してください>> " , f ,!
        Set file = ##class(%File).%New(f)
        Do file.Open("WSNK¥UTF8¥")
        Do file.WriteLine("<?xml version=""1.0"" encoding=""UTF-8""?>")
        Set company = ##class(XML.Company).%OpenId(1)
        Do company.XMLExportToStream(.file)
        Quit
}
```

図 12 XML.Company:xmlCompExport()メソッドコード

記述が終了したら、コンパイルし、Caché ターミナルから、以下を実行します。

```
USER>do ##class(XML.Company).xmlCompExport()
XML 形式で XML.Company データをエクスポートします
ファイル名を指定してください>> c:¥temp¥test.xml
```

図 13 XML.Company:ターミナルでの xmlCompExport()実行

＜クラスメソッドのコードをコピーされる方は、Step1 フォルダにある 貼り付け元.txt を使い、コードを貼り付けてください。＞

【ご参考】
Open時のパラメータについては、Web上ドキュメントか、以下URLよりご参照いただけます。
（Openコマンドと同じパラメータを使います）
http://localhost:57772/csp/docbook/DocBook.UI.Page.cls?KEY=GIOD_rmsseqfiles

xmlCompExport()メソッド実行後、作成された xml ファイルを開き、内容を確認してみてください。

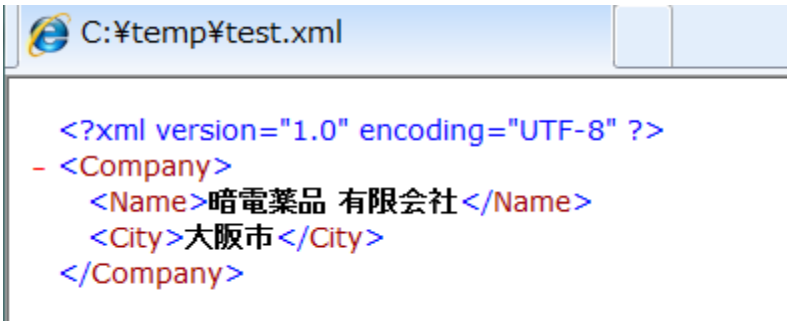

図 14 xmlCompExport():出力結果

3.2. Caché クラスインスタンスを XML からインポート

XML プロジェクションにより、XML ドキュメントを読み込み、そのドキュメントの要素に対応した Caché オブジェクトのインスタンスを生成することができます。これは、%XML.Reader クラスのインスタンスを使用して行うことができます。
XML.Company クラスにクラスメソッド xmlCompImport()を作成し、%XML.Reader の使い方を確認します。

```
ClassMethod xmlCompImport()
{
        Set reader = ##class(%XML.Reader).%New()
        Read "インポートする XML ファイル名を指定してください>> ",f
        // 指定したファイルを Open します。
        Do reader.OpenFile(f)
        // XML エレメント名とクラス名の関連付け
        //      第 1 引数:XML エレメント名、第 2 引数:クラス名(パッケージ名付き)
        Do reader.Correlate("Company","XML.Company")
        // XML ファイルからのデータ読み込み
        While (reader.Next(.object)) {
                Do object.%Save()
        }
}
```

図 15 XML.Company:xmlCompImport()メソッドコード

メソッドの記述が終了したらコンパイルします。
＜クラスメソッドのコードをコピーされる方は、Step1 フォルダにある 貼り付け元.txt を使い、コードを貼り付けてください。＞

Caché ターミナルから、以下を実行します。入力ファイルには、3.1.2 でエクスポートした XML ファイルを指定します。

```
USER>Do ##class(XML.Company).%KillExtent()

USER>do ##class(XML.Company).xmlCompImport()
インポートする XML ファイル名を指定してください>> c:¥temp¥test.xml
```

図 16 XML.Company:xmlCompImport()メソッドの実行

終了後、クラスインスタンスが 1 つ生成されているはずです。SQL ポータルから以下の SQL 文を入力、実行すると、以下の様にインスタンスが 1 つ(1 行)表示されます。

```
SELECT * FROM XML.Company
```

[ホーム]>[SQL] より[SQL 文の実行]を選択します。

図 17 SQL ポータル:SQL 文の実行

<この章で作成したクラス定義のサンプルは、Step1 フォルダにあります。
(XMLCompany.xml)>

4. Caché オブジェクト-XML プロジェクション

特定のクラスやプロパティ・パラメータに対する値を指定することで、クラスが XML に投影される方法を制御することができます。

4.1. XML プロジェクションの変更パラメーター覧

以下のリストは、XML プロジェクションを変更するためのパラメーター覧です。詳細は、オンラインマニュアルを参照して下さい。

パラメータ	説明
XMLIO	プロパティがインポート、またはエクスポートされる方法を制御します。
XMLITEMNAME	XML に投影されるとき、コレクション内の個別の項目に使用される名前を制御します。
XMLKEYNAME	XML に投影されるとき、配列要素に対するキー値を保持するために使用される属性の名前を制御します。
XMLNAME	XML に投影されるとき、クラスやプロパティに使用される名前（要素や属性）を制御します。
XMLPROJECTION	プロパティが XML に投影される方法を制御します。
XMLTYPE	クラス名が XML スキーマでどのように表わされるかを制御します。
XSDTYPE	クラスに対する XML スキーマを生成するときに、プロパティに対して使用される XSD タイプを上書きする方法を提供します。

図 18 XML プロジェクション：パラメーター一覧

4.2. コレクションの XML プロジェクション

XML の 1 つの特徴は、階層的な表現が可能であるという点です。この階層な構造をクラス構造に投影するためにコレクションを使用します。

4.2.1. サンプルデータの準備

管理ポータル→グローバル　より、Step1 フォルダにある step1.gsa をインポートして下さい。

図 19 管理ポータル：グローバルインポート

4.2.2. List コレクション

Caché ターミナルにログインし、Sample.Utils クラスの xmlexport()メソッドを実行します。

```
USER>do ##class(Sample.Utils).xmlexport()
データを XML 形式でエクスポートします
ファイル名を指定してください>> c:¥temp¥GrandParent.xml
```

図 20 Sample.Utils クラスの xmlexport()メソッドの実行

出力ファイル名を聞いてきますので、適当な名前を指定します。

メソッドの実行終了後、出力したファイルを IE 等のウェブブラウザで開いてください。
IE（インターネットエクスプローラー）の場合、以下の様に XML 形式で表示されます。この表示中いくつかの List コレクション構造を見ることができます。
例えば、Parent は、GrandParent クラスの P3 プロパティで定義された List コレクションです。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <GrandParent>
    <P1>ブレイクビーツ</P1>
    <P2>802</P2>
  - <P3>
    - <Parent>
        <P1>モダンアミュズメント</P1>
        <P2>1696</P2>
      - <P3>
        - <Child>
            <P1>フォワード</P1>
            <P2>7246</P2>
          - <P5>
              <P1>45rpm</P1>
              <P2>6702</P2>
            </P5>
          - <P3>
              <P3Item P3Key="H531">6664</P3Item>
              <P3Item P3Key="J507">3568</P3Item>
              <P3Item P3Key="R534">5309</P3Item>
              <P3Item P3Key="U377">8991</P3Item>
              <P3Item P3Key="X909">3261</P3Item>
            </P3>
          - <P4>
              <P4Item>モンスター</P4Item>
              <P4Item>ラブラ</P4Item>
              <P4Item>フォワード</P4Item>
              <P4Item>フォワード</P4Item>
              <P4Item>フラミンゴサルン</P4Item>
            </P4>
          </Child>
        - <Child>
            <P1>666</P1>
            <P2>9433</P2>
```

図 21 Sample.Utils クラス：xmlexport ()メソッド実行結果

4.2.3. Array コレクション

[図 21 Sample.Utils クラス:xmlexport ()メソッド実行結果]で Child クラスの P3 プロパティは、Array コレクションです。XML にエクスポートするとプロパティ名＋Item というタグとプロパティ名＋Key という Attribute にキー値が設定され、その後にプロパティ値があります。

5. XML スキーマウィザード

Caché スタジオには、XML スキーマからクラスを自動生成する XML スキーマウィザードがあります。

5.1. XML スキーマウィザードの実行

Caché スタジオの「ツール」→「アドイン」→「アドイン」を選択すると以下のダイアログが表示されますので、XML スキーマウィザードを選択し、OK ボタンをクリックします。

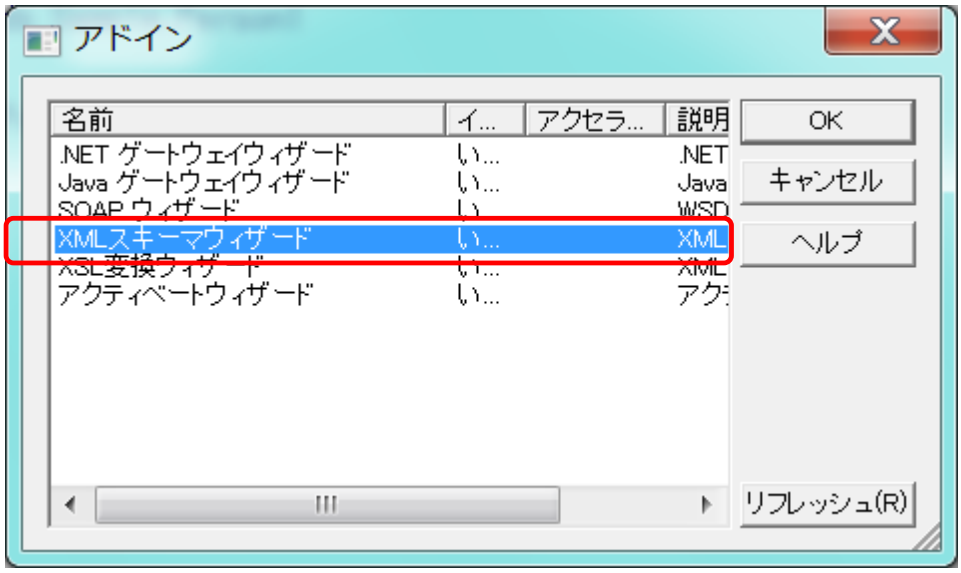

図 22 スタジオのツール:アドインメニュー 一覧

ここで、前準備として http://www.w3schools.com/schema/schema_howto.asp のサンプルから note.xsd(XML Schema)と note.xml(XML Document)をコピーペーストし、保存します。

参照ボタンをクリックし、上で作成した note.xsd ファイルを指定します。

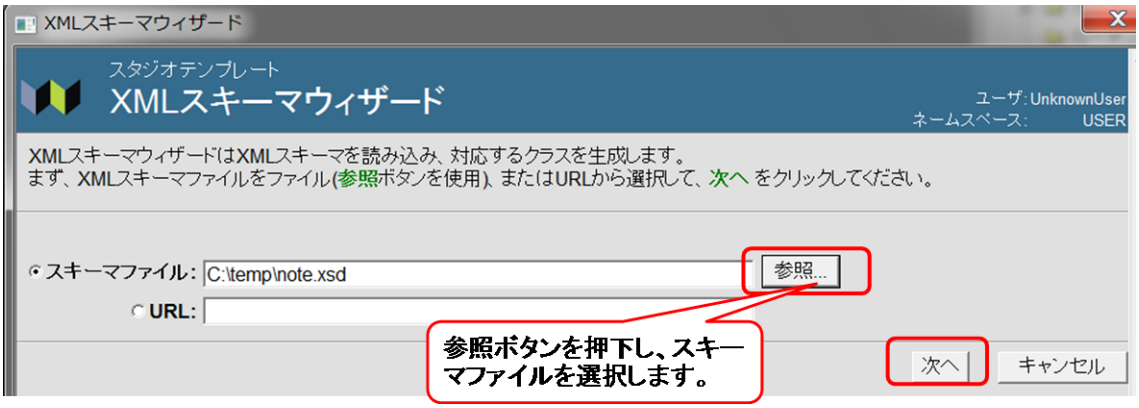

図 23 XML スキーマウィザード:note.xsd の指定

次へボタンを押下します。スキーマのソースが以下のように表示されます。

※ ご利用のバージョンによっては、以下表示が空欄となる場合もありますが正常に動作します。

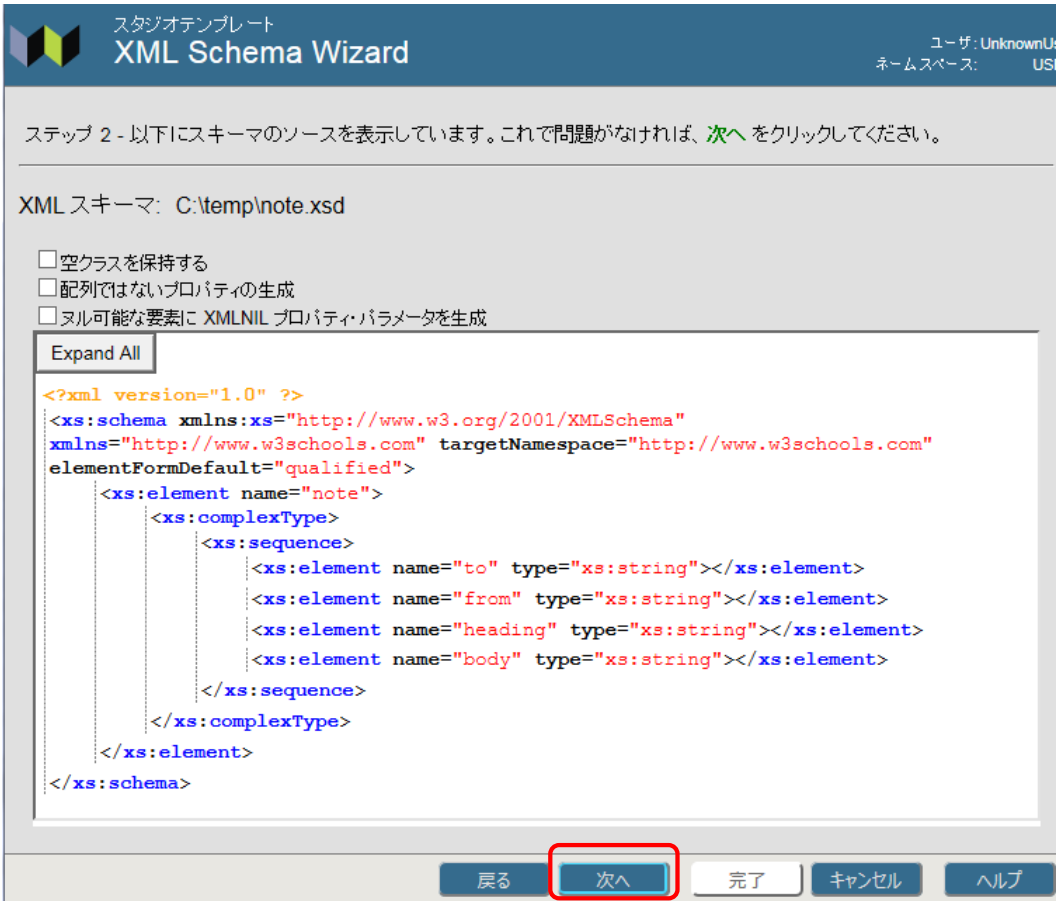

図 24 XML スキーマウィザード:スキーマのソース表示

次へボタンを押下します。

パッケージ名に任意名を入力し、次へボタンを押下します。

例では、パッケージ名は XML としています。

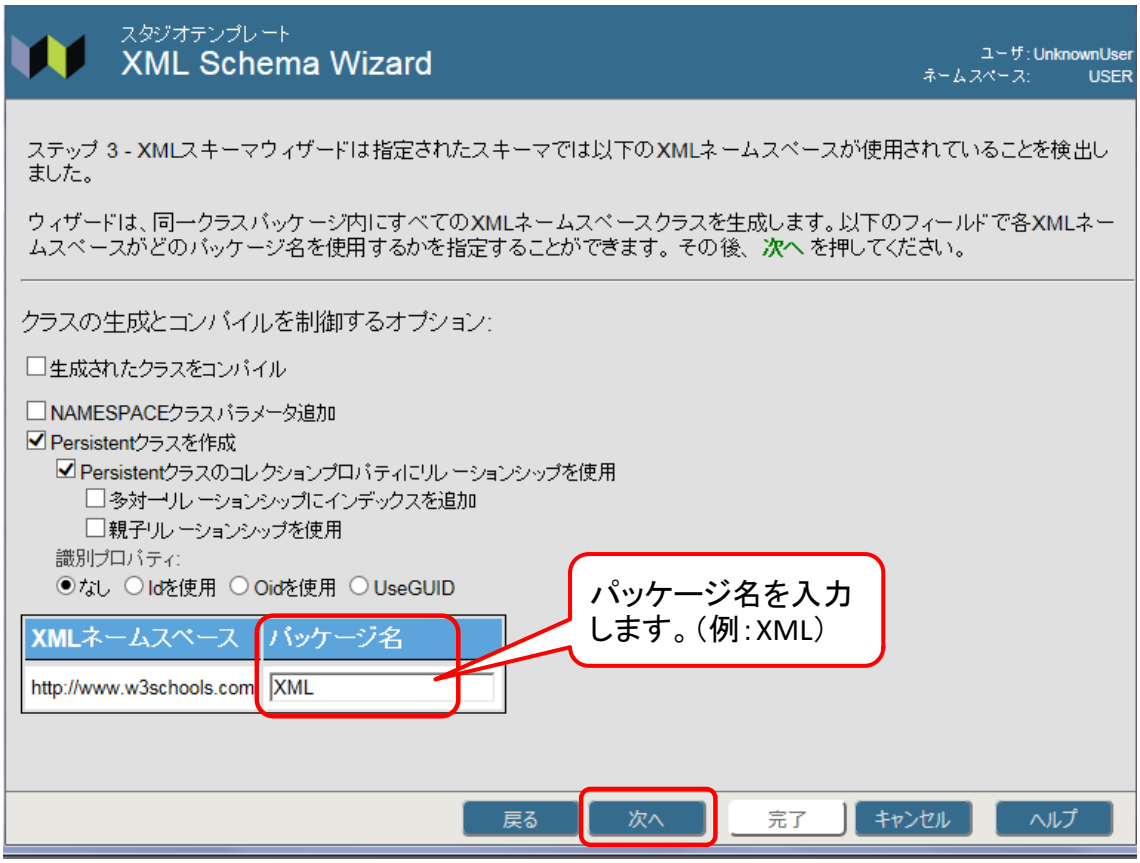

図 25 XML スキーマウィザード:パッケージ名の指定

次へボタンを押下します。

スキーマ用に生成するクラス情報が表示されます。次へボタンを押下します。

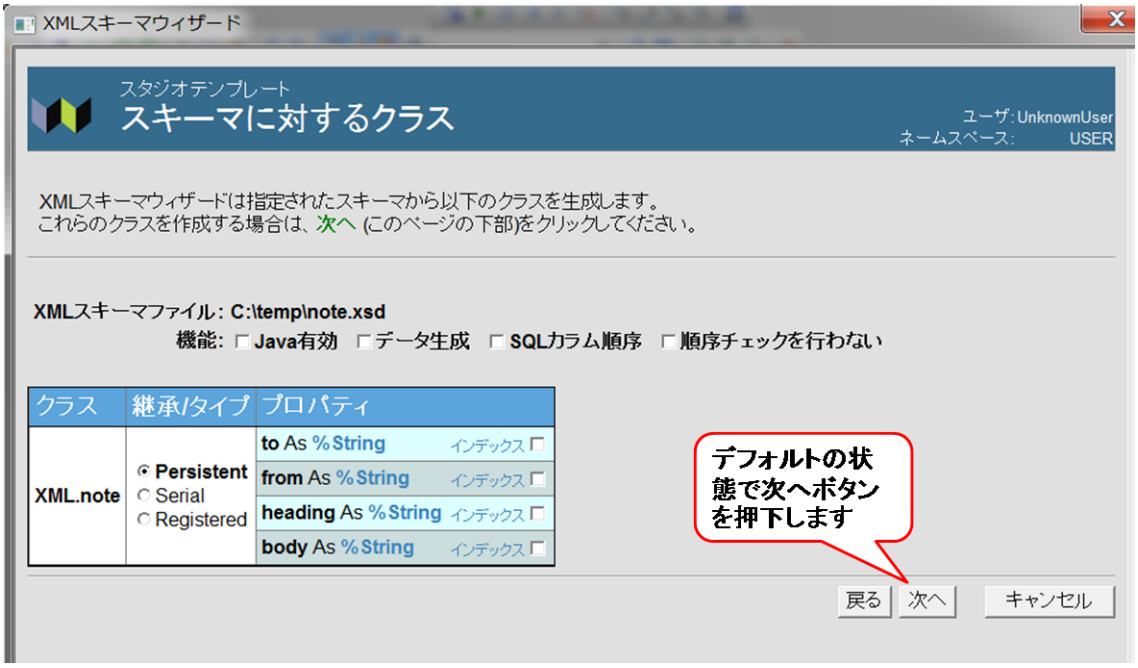

図 26 XML スキーマウィザード:クラス情報の確認

最後のページで完了ボタンを押下し、クラス定義を保存します。

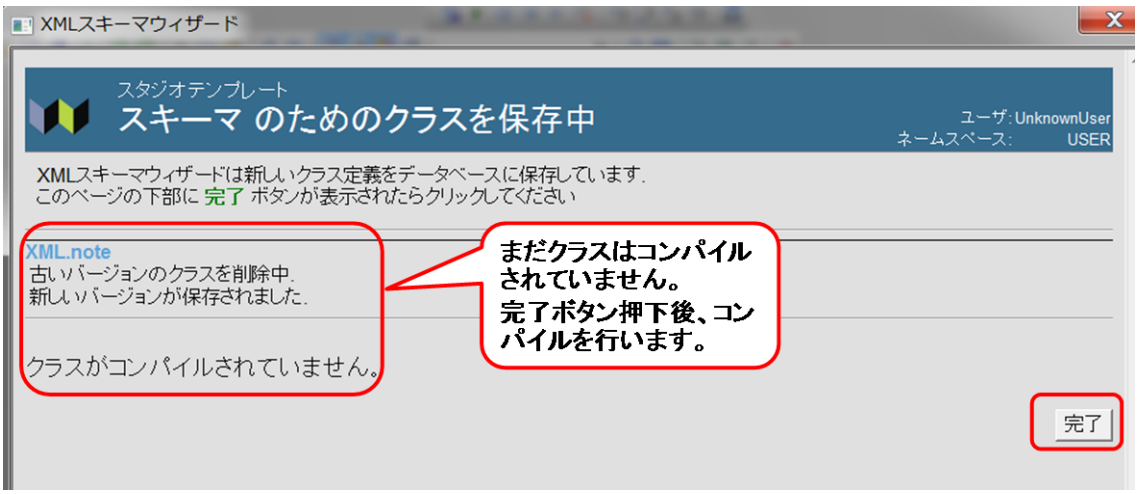

図 27 XML スキーマウィザード:最終画面

続いて、生成されたクラス XML.note.cls をスタジオでコンパイルします。

5.2. XML ファイルからクラスインスタンスの生成

それでは、クラスのインスタンスを xml ドキュメントからインポートして生成する方法を試していきましょう。以下のコード例にならって、XML.note クラスに新しいクラスメソッドを作成します。
なお、コードでは、先ほど note.xsd により作成したクラスを XML.note クラス として記述しています。パッケージ名を異なる名称とされた場合は、パッケージ名.note クラスに新メソッドを作成してください。

＜コード例のコピーや参照されたい方は、Step1 フォルダにある 貼り付け元.txt の中から xmlnoteimport()メソッドの記述をご利用ください。＞

```
ClassMethod xmlnoteimport()
{
	Set reader = ##class(%XML.Reader).%New()
	Read "インポートする XML ファイル名を指定して下さい>>  ",f,!
	Do reader.OpenFile(f)
	// XML エレメント名とクラス名の関連付け
	Do reader.Correlate( "note" , "XML.note" )
	// XML ファイルからデータ読み込み
	While ( reader.Next(.object )) {
		Set status = object.%Save()
		if $system.Status.IsError(status) {
			do $system.Status.DisplayError(status)
		}
	}
}
```

図 28 Sample.Utils クラス：xmlnoteimport()メソッド（note.xml の読み込み）

クラスメソッドの作成が完了したら、コンパイルを行います。
その後で、Caché ターミナルにて以下のコマンドを実行します。
Import ファイル名を聞いてきますので、さきほど作成した note.xml を指定します。

```
USER>do ##class(XML.note).xmlnoteimport()
インポートする XML ファイル名を指定して下さい>>  c:¥temp¥note.xml
```

実行後、XML.note クラスのインスタンスが1つ生成されます。正しく生成されているかどうかは、たとえば、SQL ポータルから確認できます。

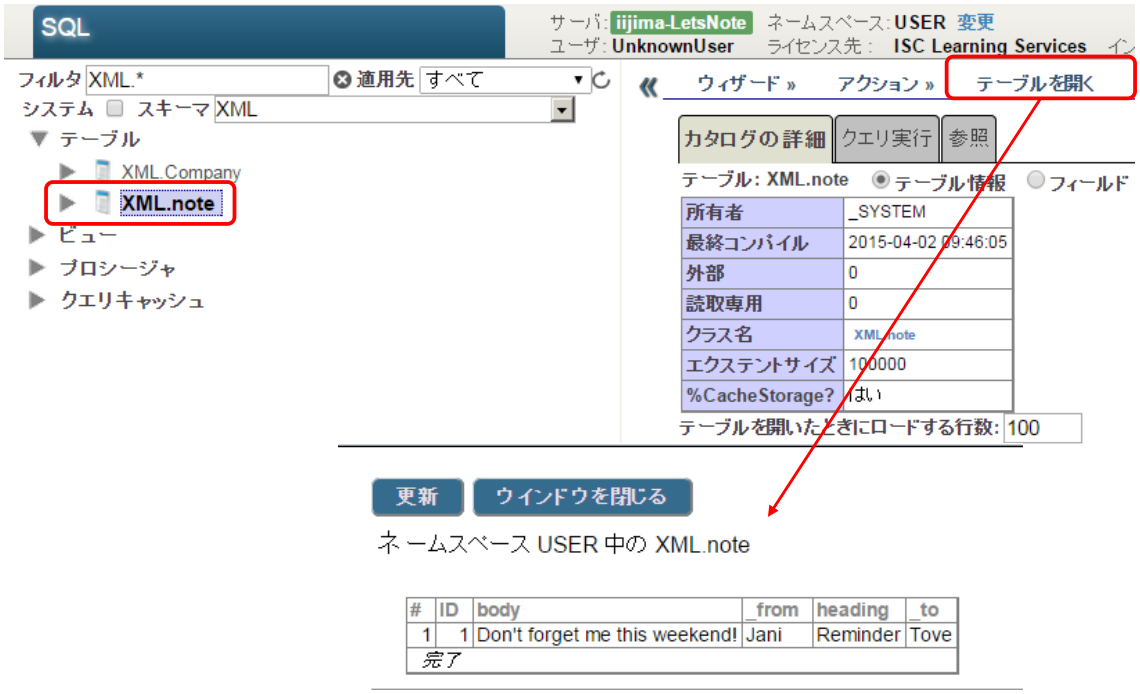

図 29 note.xml の読み込み結果確認

＜この章で作成したクラス定義のサンプルは、Step1 フォルダにあります。(XMLnote.xml)＞

6. %XML.TextReader クラス

%XML.TextReader クラスは、XMLドキュメントを処理する、単純で簡単な方法を提供します。ここでは、このクラスを使用した簡単な XPATH ライクな検索処理を実装しましょう。以下のクラスを作成、コンパイルします。
＜クラスをインポートされたい方は、Step1 フォルダにある TextReader.xml を使いインポートしてください。＞

```
Class XML.TextReader Extends ( %RegisteredObject , %XML.Adaptor )
{
ClassMethod ShowElement(filename As %String, xpath As %String)
{
        Set xpath="/"_$Translate(xpath,".","/")
        Set sc=##Class(%XML.TextReader).ParseFile(filename,.reader)
        If ($$$ISOK(sc)){
                While (reader.Read()){
                        If (reader.Path=xpath){
                                Do reader.Read()
                                Write "Value=",reader.Value,!
                        }
                }
        }
}
}
```

図 30 %XML.TextReader クラスを利用した例

次に以下の様な XML を作成し、適当なファイル名で保存します。
または、Step1 フォルダ内 test1.xml をご利用ください。

```
<?xml version="1.0" encoding="UTF-8" ?>
<ARTICLE>
<FRONT>
<KEYWORD>Treasure</KEYWORD>
</FRONT>
</ARTICLE>
```

コマンド行から以下を実行します。第 2 引数で指定された検索条件に基づき、該当要素を表示してくれるのがわかるでしょう。

```
Do ##class(XML.TextReader).ShowElement("c:¥temp¥test1.xml","ARTICLE.FRONT.KEYWORD")
```

この様に TextReader クラスを使用して、通常の Class-XML マッピングに比較してより複雑な処理を実現することができます。

7. SAX インタフェース

SAX (Simple API for XML) は、イベント駆動型の XML パーサーです。SAX を使用して、XML ファイルを読み、ファイル内の要素でメソッドを実行することができます。Caché には SAX コンテンツ・ハンドラ、%XML.SAX.ContentHandler クラスが備わっています。カスタムXML インポートと処理メカニズムを作成するには、%XML.SAX.ContentHandler のサブクラスを生成し、その新規のクラスで既定のコンテンツ・ハンドラのメソッドを上書きして、必要なアクションを実行します。このインタフェースはオーバヘッドの少ない高速処理を提供しますが、全ての処理を開発者が実装しなければならず、特にプログラム全体のコンテキストをプログラマが管理する必要があり、ロジックが複雑になりがちです。ここでは、簡単なSAXインタフェースの使い方のみの説明に留めることにします。

それではまず、以下のクラスを作成し、コンパイルします。
＜クラスをインポートされたい方は、Step1 フォルダにある XMLHandler.xml を使いインポートしてください。＞

```
Class XML.XMLHandler Extends %XML.SAX.ContentHandler
{
ClassMethod ReadFile(file As %String)
{
        // XMLHnder クラスのインスタンスを生成します
        Set handler=##Class(XMLHandler).%New()

        //  生成したインスタンスを使いファイルをパースします
        Set sc=##Class(%XML.SAX.Parser).ParseFile(file,handler)
}
///  以下メソッドはスーパークラスからオーバーライドしています。
Method startElement(uri As %String, localname As %String, qname As %String, attrs
As %List)
{
        Write !,"Element: ",localname
        Write !,"attrs: ",attrs
        Quit
}
Method characters(chars As %Library.String, length As %Library.Integer)
{
        Write !,"Characters: ",chars
        Quit
}
}
```

図 31 %XML.SAX.ContentHandler 利用例

上で作成したクラスを実行するには、以下の操作を行います。

```
Do ##class(XML.XMLHandler).ReadFile(filename)
```

ここで filename には適当な xml ファイル名を指定します。(c:¥temp¥test1.xml など)
入力された xml を順次読み込みながら、element 毎にイベントハンドラが動作し、そのイベントに関連した情報を表示するのがわかると思います。

8. DOM インタフェース

%XML.Reader クラスと%XML.Document クラスを組み合わせて使うことにより XML ドキュメントを DOM（Document Object Model）として表現できます。
ここでは、このクラスを使用した簡単な XML DOM 構造表示プログラムを作成しましょう。
以下のクラスを作成し、コンパイルします。

＜クラスをインポートされたい方は、Step1 フォルダにある XMLDOM.xml を使いインポートしてください。＞
以下、XML.DOM クラス内クラスメソッドの記載です。

```
ClassMethod TestDOM(pFileName As %String)
{
		set tNode = ..ReadFile(pFileName)
		do ..ShowNode(tNode)
		do tNode.MoveToFirstChild(1)
		do ..ShowNode(tNode)
		do tNode.MoveToFirstChild(1)
		do ..ShowNode(tNode)
}
ClassMethod ReadFile(pFileName) As %XML.Node
{
	set reader=##class(%XML.Reader).%New()
	set status=reader.OpenFile(pFileName)
	if $$$ISERR(status) do $System.Status.DisplayError(status) quit ""
	set document=reader.Document
	quit document.GetDocumentElement()
}
```

図 32 DOM インターフェース例 1

```
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"
        }
}
```

図 33 DOM インターフェース例 2

[図 32 DOM インターフェース例 1][図 33 DOM インターフェース例 2] で作成したクラスのメソッドを実行するには、以下の操作を行います。

```
Do ##class(XML.DOM).TestDOM(filename)
```

ここで filename には適当な xml ファイル名を指定します。(c:¥temp¥test1.xml など)
入力された xml の要素毎にその要素に関連した情報を表示するのがわかると思います。

9. 要素を探す

XML ドキュメントから特定のタグを探し、その内容を取得することが%XML.TextReader クラスを使ってできます。

1 つ前の項目でインポートしたクラス（XML.DOM）の以下メソッドを使用します。

```
ClassMethod XMLSearchTest(pFileName As %String)
{
        Do ##class(%XML.TextReader).ParseFile(pFileName,.textreader)
        If 'textreader.Read() Quit $$$ERROR($$$GeneralError,"リードエラー")
        If 'textreader.ReadStartElement("KEYWORD") {
                Quit $$$ERROR($$$GeneralError,"'KEYWORD'が見つかりません。")
        }
        If 'textreader.MoveToContent() {
                Quit $$$ERROR($$$GeneralError,"MoveToContent() エラー")
        }
        write "KEYWORD の内容 = ",textreader.Value
}
```

上記メソッドを実行した時の結果は、以下の通りです。

```
USER>Do ##class(XML.DOM).XMLSearchTest("c:¥temp¥test1.xml")
KEYWORD の内容 = Treasure
```