Avoiding Problems Reading DOM XML Nodes

By David Adams

Technical Note 06-44

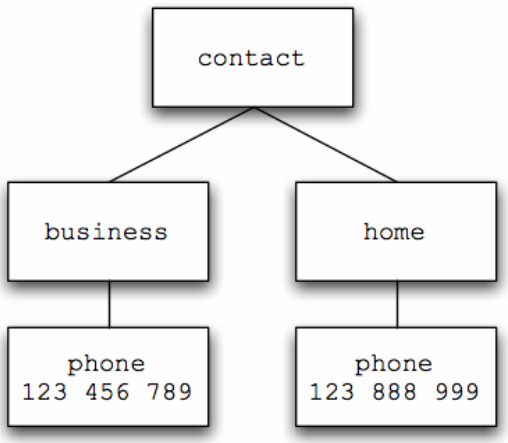
Overview

4th DimensionのDOM (Document Object Model) XMLコマンドは、ソースXMLドキュメントをリンクされたノードの構造木として解析します。DOMコマンドには、構造木を検索し、ノードから情報を取得するものが含まれています。**DOM Get Parent XML element** や **DOM Get first child XML element** などの構造木を移動するコマンドでは、システム変数OKを使用して、コマンドが構造木の終端を越え、存在しないノードに達したことを知ることができます。この仕様により、特定のノードが祖先や兄弟・子孫をいくつ持っているか事前に知らなくても、ノードの移動を行うことができます。しかしこれは同時に、これらのノード移動コマンドからは無効なノードが返されることもあるということです。これは要素の名前や値、属性を読み込む際に問題となります。このテクニカルノートでは、無効なノードを扱う際の単純な方針と、問題を避ける方法について説明します。

Review of DOM Features and Behavior

4th Dimension 2004は2種類のXML読み込みコマンド、DOMコマンドとSAXコマンドを提供しています。このテクニカルノートではDOMコマンドのみを扱います。前述のとおり、DOMコマンドは解析したXMLをリンクされたノードの構造木として扱います。例として、単純なXMLとDOM構造をあらわした図を見てみましょう:

```
<?xml
  version="1.0"
  encoding="UTF-8"
  standalone="no" ?>

<contact>

 <business>
  <phone>123 456 789</phone>
 </business>

 <home>
  <phone>123 888 999</phone>
 </home>

</contact>
```



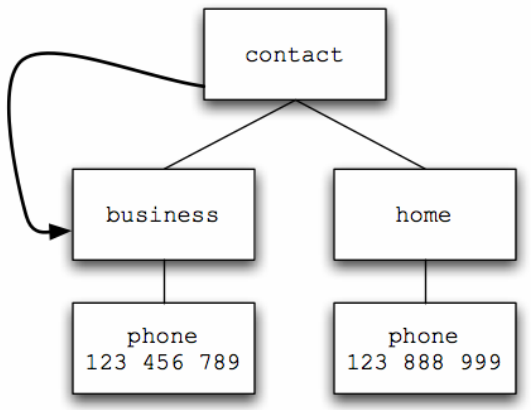
上のXMLを生成する方法はいくつかあります。例えば以下のようなコードになるでしょう:

```
C_STRING(16;$root_xmlref)
C_STRING(16;$businessPhone_xmlref)
C_STRING(16;$homePhone_xmlref)
$root_xmlref:=DOM Create XML Ref("contact")
$businessPhone_xmlref:=DOM Create XML element($root_xmlref;"/contact/business/phone")
$homePhone_xmlref:=DOM Create XML element($root_xmlref;"/contact/home/phone")
DOM SET XML ELEMENT VALUE($businessPhone_xmlref;"123 456 789")
DOM SET XML ELEMENT VALUE($homePhone_xmlref;"123 888 999")
```

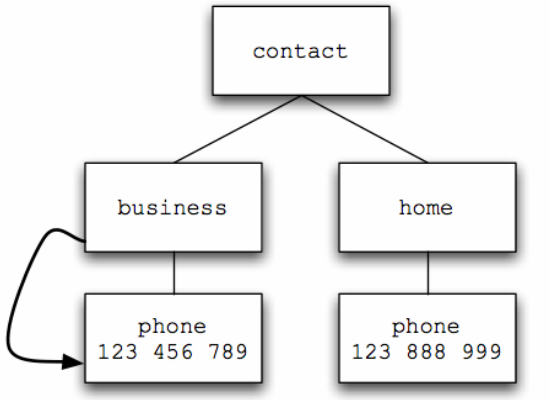
ここで、XML要素を走査し、それぞれの名前と値を読み込むコードを見てみましょう:

```
C_STRING(16;$working_xmlref)
C_TEXT($elementName_t)
C_TEXT($elementValue_t)
$working_xmlref:=DOM Get first child XML element($root_xmlref)
DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t) ` business
DOM GET XML ELEMENT VALUE($working_xmlref;$elementValue_t)

$working_xmlref:=DOM Get first child XML element($working_xmlref)
DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t) ` phone
DOM GET XML ELEMENT VALUE($working_xmlref;$elementValue_t) `123 456 789

` The next node is off the tree and invalid.
$working_xmlref:=DOM Get first child XML element($working_xmlref)
DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t)
DOM GET XML ELEMENT VALUE($working_xmlref;$elementValue_t)

DOM CLOSE XML($root_xmlref)
```

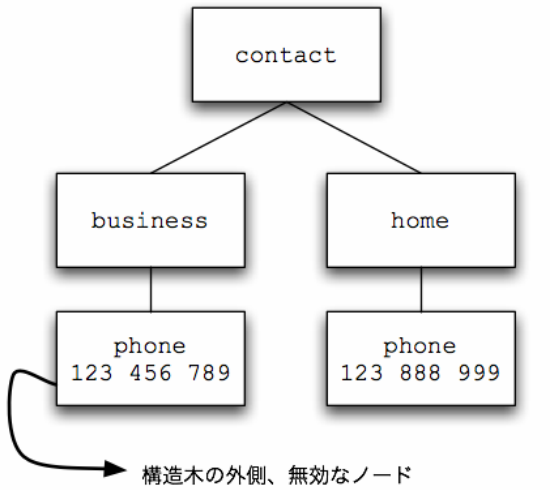
最初の行の$root_xmlrefは、先のコードで生成されたXMLの参照が保持されています。ルート参照は**DOM Parse XML source**や**DOM Parse XML variable**を使用してXMLを解析することによって得ることもできます。構造木の中で、最初の**DOM Get first child XML element**の呼び出しによって、`contact`ルート要素から`business`要素に移動します:



2番目の**DOM Get first child XML element**の呼び出しで、business要素からphone要素に移動します:



3番目の**DOM Get first child XML element**の呼び出しで、phone要素から構造木の外側に移動してしまいます:

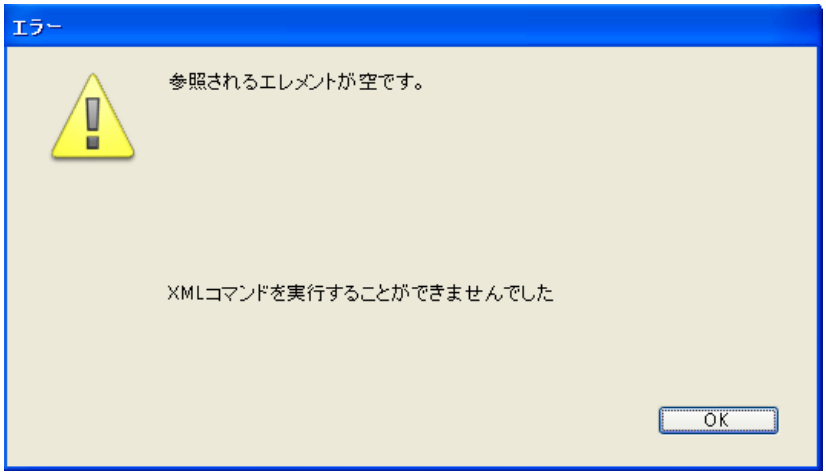


この時点で、4th DimensionはOKシステム変数を0に設定します。これらはすべて通常の振る舞いであり、バグではありません。ところで無効なノード参照を読み込むコードを実行すると、何が起こるでしょう？以下のコードで、$working_xmlrefが無効な参照を保持しているとします:

```
DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t)
```

一部のバージョンの4th Dimensionでは、無効な参照を渡すことにより4th Dimensionが予期せず終了することがありました。正しい振る舞いはエラーを投げる

ことです。エラーハンドラがインストールされていないと、4th Dimensionはエラーダイアログを表示します:



Avoiding Problems While Reading Nodes

Overview

今まで論じてきたことに関連する情報は以下のとおりです: DOMコマンドを使用していると、無効なノードに移動してしまうことがあり、いくつかのDOM情報関連コマンドにはそれら無効なノード参照に問題を抱えています。幸いなことにDOMコマンド利用時のこれらの問題に対応するシンプルな方針があります:

Test the OK Variable

DOMのナビゲーションコマンドは、無効なノードに移動した際、システム変数OKを0に設定します。この仕様に沿えば、例えば汎用のナビゲーションメソッドで最後の祖先、兄弟あるいは子孫ノードに達したことを検知することができます。同様に、すべてのDOMナビゲーションコマンドで、DOM読み込みコマンドを使用する前にOKシステム変数をチェックすることができます。前述の問題のあるコードを、OKシステム変数を使用する形に書き直してみましょう:

```
C_STRING(16;$working_xmlref)
C_TEXT($elementName_t)
C_TEXT($elementValue_t)

$working_xmlref:=DOM Get first child XML element($root_xmlref)
If (OK=1)
  DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t) ` business
  DOM GET XML ELEMENT VALUE($working_xmlref;$elementValue_t)
End if

$working_xmlref:=DOM Get first child XML element($working_xmlref)
If (OK=1)
  DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t) ` phone
  DOM GET XML ELEMENT VALUE($working_xmlref;$elementValue_t) `123 456 789
End if

$working_xmlref:=DOM Get first child XML element($working_xmlref)
If (OK=1)
  DOM GET XML ELEMENT NAME($working_xmlref;$elementName_t)
  DOM GET XML ELEMENT VALUE($working_xmlref;$elementValue_t)
End if
```

より簡単に管理でき、自由度も高い方法も他にありますが、このシンプルな例はOKシステム変数を使用することでほとんどの場合問題を避けることができることを示します。

Use the Optional Parameters While Navigating Nodes

XMLノードは名前と値、ときに属性値を持ちます。利便性のため、DOMナビゲーションコマンドは追加の引数をサポートしていて、選択されたノードの名前と値を受け取ることができます。この機能は非常に便利で、コードを簡単にするほか、**DOM GET XML ELEMENT NAME**や**DOM GET XML ELEMENT VALUE**に無効なノード参照を渡すことによる問題を避けることができます。以下のコード例は、この方針で書かれたコードが以下に簡便であるかを示します:

```
C_STRING(16;$working_xmlref)
C_TEXT($name_t)
C_TEXT($value_t)
$working_xmlref:=DOM Get first child XML element($root_xmlref;$name_t;$value_t) ` business
$working_xmlref:=DOM Get first child XML element($working_xmlref;$name_t;$value_t) ` phone
$working_xmlref:=DOM Get first child XML element($working_xmlref;$name_t;$value_t) ` <-- Bad
node
DOM CLOSE XML($root_xmlref)
```

最後の**DOM Get first child XML element**の呼び出しは無効なノード参照を返します。しかしエラーは発生しません。名前と値の変数は空で、OKシステム変数は0に設定されます。このテクニックの主たる制限は、ノードの属性を読み取ることをサポートしていないことと、汎用のナビゲーションコードを書くためには役立たないことです。

Use an ON ERR CALL Method

ON ERR CALLを使用してカスタムエラーハンドラをインストールすることは、プログラム実行に際して無効なノード参照によるエラーを避けるのに有効なテクニックです。このアプローチは、ノードの属性を数えたり読み取ったりする際に、OKシステム変数をテストしない場合に必要となります。

Write a Node Validation Routine

カスタムエラーハンドラをインストールして**DOM GET XML ELEMENT NAME**などのコマンドをコールするなどの方法で、無効なノード参照に対しエラーを生成する汎用的なノード参照検証メソッドを作成することができます。以下のコードはこのテクニックを示します:

```
C_BOOLEAN($0;$nodeRefIsOkay_b)
C_STRING(16;$1;$noderef)

$noderef:=$1
$nodeRefIsOkay_b:=True

Error:=0
ON ERR CALL("DOM_ReferenceIsValidOnError")

C_TEXT($elementName_t) ` Line below should throw an error if the element isn't valid.
DOM GET XML ELEMENT NAME($noderef;$elementName_t)

ON ERR CALL("")

If ($elementName_t="")
  $nodeRefIsOkay_b:=False
Else
  $nodeRefIsOkay_b:=True
End if

$0:=$nodeRefIsOkay_b
```

サンプルデータベースにはさらに改良されたバージョンのコード、*DOM_ReferenceIsValid*が収められています。

Special Case: XML Attributes and the #document Node

あともうひとつ、特定のケースについて記述するべきですね。XMLノードには0個以上の属性が含まれます。例えば以下のように:

```
<contact id="1">
```

DOMコマンドのひとつ**DOM Count XML attributes**を使用して、ノードに属する属性の数を数えることができます。残念ながら、一部のバージョンの4th Dimensionで、特定の状況下でこの関数をコールすると、予期せず4th Dimensionが終了してしまいます。**DOM Get parent XML element**を使用して、ルートノードのさらに上、XMLバー

ジョンやエンコーディングなどのXMLドキュメント情報を保持する理論上のノードにアクセスすることができます。先に示したノード検証コードでは、このノードの検証に失敗します。なぜならこのノードは名前を持っており、基本的には有効なノードだからです。しかし、**DOM Count XML attributes**関数はこのノードを正しく処理することができないかもしれません。幸いなことにこのノードを認識することは簡単です。なぜならこのノードの名前は`#document`だからです。以下のコードは無効なノードや`#document`でのエラーを避けつつ、XMLノードの属性数を数えるコードです**:**

```
C_BOOLEAN($0;$attributes_count)
C_STRING(16;$1;$noderef)

$noderef:=$1

$attributes_count:=0

Error:=0
ON ERR CALL("DOM_ReferenceIsValidOnError")

C_TEXT($elementName_t)` Line below should throw an error if the element isn't valid.
DOM GET XML ELEMENT NAME($noderef;$elementName_t)

ON ERR CALL("")

Case of
  ¥ ($elementName_t="")` Not a valid node, do not count attributes.
    $attributes_count:=0

  ¥ ($elementName_t="#document")` Special info node, do not count attributes.
    $attributes_count:=0

  Else ` A valid node, count attributes.
    $attributes_count:=DOM Count XML attributes($noderef)
End case

$0:=$attributes_count
```

サンプルデータベースにはこれを改良したバージョンのコード*DOM_CountAttributes*が収められています。

注 *XML属性に関するより詳細な説明は、テクニカルノート 06-43* ***Enhanced Tools for Reading XML Attributes*** *も参照してください。*

Summary

DOM 解析コマンドは XML をリンクされた構造木に変換します。DOM の XML コマンドを使用すれば、構造木内を自由に行き来することができます。しかし構造木を飛び出してしまうこともあります。この状況はおかしなことではないのですが、要素の名前や値、属性を読み込む際の問題の原因ともなります。このテクニカルノートで説明した組み込みの機能とシンプルなテクニックを使用して、4th Dimension のデベロッパは、DOM を通した XML ノード読み込みに関連するすべての問題を避けることができます。