Debugging XML with Export

By David Adams
Technical Note 06-41

Overview

4th DimensionのDOM (Document Object Model) XMLコマンドは、妥当なXML文書の読み書きをサポートします。ともすればXML文書をテキストファイルと考えてしまいがちですが、DOMコマンドは、リンクされたノードの構造木としてXMLを扱います。それぞれのノードは名前の他、値や属性を持つこともあります。最終的にはXMLをテキストに変換できますが、内部的に、コマンドは常に構造木を操作します。4th Dimension環境において、XMLはそれがXMLドキュメントに変換されるまでは目に見ることができません。XMLの作成に奮闘したことのある方はご存知だと思いますが、見ることのできないものをデバッグするのは大変なことです。DOMコマンドを使用してXMLを作成するとき、**DOM EXPORT TO VAR**や**DOM EXPORT TO FILE**を使用して木構造の内容を書き出すことで、デバッグを容易にすることができます。さらに**DOM EXPORT**コマンドを使用して、他のDOMコマンドの動作を学ぶこともできます。

Creating XML with the DOM Commands

DOM書き出しコマンドの前に、簡単なXMLを作成する以下のコードを見てください：

```
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")
DOM CLOSE XML($root_xmlref)
```

4th DimensionのXMLコマンドがどのように動作するかすでにご存知であれば、上記のコードが何を行うか簡単に推測できるでしょう。他方このコードは少々難解でもあります。これは、4D 2004のWebサービスウィザードを使用して、複合サービスのSOAPプロクシメソッドを作成したことのある方はよく感じることだと思います。以下はWebサービスウィザードが作成する、XML生成コード例の一部です：

```
C_STRING(16;$subelem)
C_TEXT($namespace)
$namespace:="http://www.4d.com/namespace/default"
$root:=DOM Create XML Ref("SendValveSettings";$namespace)
$subelem:=DOM Create XML element($root;"/SendValveSettings/Temperature")
DOM SET XML ELEMENT VALUE ($subelem;$1)
$subelem:=DOM Create XML element($root;"/SendValveSettings/Pressure")
DOM SET XML ELEMENT VALUE ($subelem;$2)
$subelem:=DOM Create XML element($root;"/SendValveSettings/Vector_One/item")
DOM SET XML ELEMENT VALUE ($subelem;$3)
$subelem:=DOM Create XML element($root;"/SendValveSettings/Vector_Two/item")
DOM SET XML ELEMENT VALUE($subelem;$4)
```

このようなプロキシコードから生成されたXMLに何かおかしなところがあったとしても、XMLをテキストに書き出して検査しない限り、問題点を探し出すことは困難です。

"Serializing" Incomplete Trees

XMLをテキストに変換する処理はしばしば「シリアライズ」と呼ばれます。**DOM EXPORT TO VAR**と**DOM EXPORT TO FILE**コマンドは、出力先が異なりますが、この作業を行います。**DOM EXPORT TO VAR**はBLOBもサポートしますが、ここではXMLをテキスト変数に書き出すことにします。さて最初の例を書き直して、ステップごとにXMLの内容を確認できるようにしてみましょう。(読みやすくするため、適宜改行を入れてます):

```
C_STRING(16;$root_xmlref)
C_STRING(16;$businessPhone_xmlref)
C_STRING(16;$homePhone_xmlref)

$root_xmlref:=DOM Create XML Ref("contact")
DOM_DumpTreeToClipboard ($root_xmlref;"DOM Create XML Ref('contact')";True)

$businessPhone_xmlref:=DOM Create XML element($root_xmlref;"/contact/business/phone")
DOM_DumpTreeToClipboard ($root_xmlref;"DOM Create XML Ref('contact/business/phone')")

$homePhone_xmlref:=DOM Create XML element($root_xmlref;"/contact/home/phone")
DOM_DumpTreeToClipboard ($root_xmlref;"DOM Create XML element('/contact/home/phone')")

DOM SET XML ELEMENT VALUE($businessPhone_xmlref;"123 456 789")
DOM_DumpTreeToClipboard
      ($root_xmlref;"DOM SET XML ELEMENT VALUE($businessPhone_xmlref;'123 456 789')")

DOM SET XML ELEMENT VALUE($homePhone_xmlref;"123 888 999")
DOM_DumpTreeToClipboard
      ($root_xmlref;"DOM SET XML ELEMENT VALUE($homePhone_xmlref;'123 888 999')")

DOM CLOSE XML($root_xmlref)
```

上記のサンプルコードでは、*DOM_DumpTreeToClipboard*メソッドを5回呼んでいます。それぞれの呼び出しでXML参照と、ヘッダとして使用される解析情報が渡されます。*DOM_DumpTreeToClipboard*は解析用の情報と、参照で渡されたXMLのすべての内容をクリップボードに追加します。このメソッドを実行すると、クリップボードには以下のテキストが格納されます:

```
DOM Create XML Ref('contact')
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact/>

DOM Create XML Ref('contact/business/phone')
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact>

 <business>
  <phone/>
 </business>

</contact>

DOM Create XML element('/contact/home/phone')
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact>

 <business>
  <phone/>
 </business>

 <home>
  <phone/>
 </home>

</contact>

DOM SET XML ELEMENT VALUE($businessPhone_xmlref;'123 456 789')
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact>

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

 <home>
  <phone/>
 </home>

</contact>

DOM SET XML ELEMENT VALUE($homePhone_xmlref;'123 888 999')
<?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>
```

DOM_DumpTreeToClipboard Code Listing

DOM_DumpTreeToClipboard メソッドは以下のとおりです:

```
C_STRING(16;$1;$xmlref)
C_TEXT($2;$introduction_text)
C_BOOLEAN($3;$clearClipboard_b)

$xmlref:=$1
$introduction_text:=Char(Carriage return )+$2+Char(Carriage return )
$clearClipboard_b:=False
If (Count parameters>=3)
  $clearClipboard_b:=$3
End if

C_TEXT($existing_text)
$existing_text:=""
If ($clearClipboard_b=False)
  $existing_text:=Get text from clipboard
End if

C_TEXT($xml_text)
$xml_text:=""
DOM EXPORT TO VAR($xmlref;$xml_text)

SET TEXT TO CLIPBOARD($existing_text+$introduction_text+$xml_text)
```

コードはとても汚いものです。例えば内容は4th Dimensionのテキスト変数の上限までしか格納できません。しかしながらここでお見せした単純な例においても、*DOM_DumpTreeToClipboard*を利用価値のあるツールです。上記の出力結果により、それぞれのコマンドがXMLをどのように更新していくかを簡単に見ることができます。例として以下のコマンドを見てみましょう:

```
$root_xmlref:=DOM Create XML Ref("contact")
```

単純なコードですが、これは完全なXMLを生成します。*DOM_DumpTreeToClipboard*で出力を見てみましょう:

```
DOM Create XML Ref('contact')
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<contact/>
```

Special Note: SET WEB SERVICE OPTION

4th DimensionがXMLをシリアライズするとき、4Dは常に完全なXMLを生成します。一行目は常に以下のようなXML宣言となります:

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

XMLは一行目にXML宣言を持ちます。XML要素の内部にXML宣言が含まれることは許されていません。このルールにより、XMLの内容をダンプする際に、誤解を招く状況が発生することがあります。**SET WEB SERVICE OPTION**コマンドは、Web Service SOAP Header オプションを使用するとき、第二引数にXML参照を渡します。この場合、4th Dimensionは引数で渡されたXMLの内容を次のSOAPリクエストメッセージに挿入します。しかし、許可されないXML宣言は挿入されません。あなたがSOAPヘッダで使用するために用意したXMLをデバッグする際には、デバッグ用に書き出されたXMLの宣言に惑わされないようにしてください。

ここで説明した状況は、サンプルのSOAPメッセージを見ることで、簡単に理解できます。まずSOAPヘッダなしの単純なSOAPリクエストXMLについて見てみましょう (読みやすさのために適宜改行を加えています):

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <SOAP-ENV:Body>
  <mns:SayHelloWorld xmlns:mns="http://www.4d.com/namespace/default">
  </mns:SayHelloWorld>
 </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

以下のコードは、リクエストに挿入するためのカスタマイズされたSOAPヘッダを準備します:

```
C_STRING(16;$root_xmlref)
C_STRING(16;$username_xmlref)
C_STRING(16;$password_xmlref)
$root_xmlref:=DOM Create XML Ref("credentials")
$username_xmlref:=DOM Create XML element($root_xmlref;"/credentials/username/")
$password_xmlref:=DOM Create XML element($root_xmlref;"/credentials/password/")
DOM SET XML ELEMENT VALUE($username_xmlref;"Donald")
DOM SET XML ELEMENT VALUE($password_xmlref;"Quack1")
SET WEB SERVICE OPTION(Web Service SOAP Header ;$root_xmlref)
```

このヘッダを挿入すると、SOAPリクエストXML中、以下の太字の部分が置き換えられます (読みやすさのために適宜改行を加えています):

```
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <SOAP-ENV:Header>
  <credentials>
   <username>Donald</username>
   <password>Quack1</password>
  </credentials>
 </SOAP-ENV:Header>

 <SOAP-ENV:Body>
  <mns:SayHelloWorld xmlns:mns="http://www.4d.com/namespace/default">
  </mns:SayHelloWorld>
 </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

XML宣言は両バージョンのメッセージ共同じで、かつ両方ともたった一つのXML宣言しか存在しません。しかしXMLをダンプしていると、XML宣言が重複してしまうのではと思うことがあります。以下のコードは、SOAPメッセージに挿入されるXMLをクリップボードにダンプします:

```
SET WEB SERVICE OPTION(Web Service SOAP Header ;$root_xmlref)
DOM_DumpTreeToClipboard ($root_xmlref;"Custom SOAP header";True)
```

クリップボードに格納されるXMLは以下のようになります:

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

 <username>Donald</username>

 <password>Quack1</password>

</credentials>
```

出力はXML宣言から始まります。なぜなら**DOM EXPORT TO VAR**や**DOM EXPORT TO FILE**は常にXMLを完全なXMLに変換するからです。しかし、すでに述べたとおり、**SET WEB SERVICE OPTION**は冗長なXML宣言を挿入しません。なので**SET WEB SERVICE OPTION**に渡すXMLの内容をダンプしたときに、XML宣言について心配する必要はありません。

Summary

4th DimensionのDOMコマンドはXMLを作成しますが、それをXMLドキュメントとして書き出すまでは、中身の調査を行うことは困難です。中身をダンプすることで、問題の解決や、Webサービスウィザードで生成されるプロキシコードのデバッグ、そしてDOMコマンドの動作を学ぶことの助けとなります。このテクニカルノートではXMLをテキストにダンプし、クリップボードに格納するユーティリティを紹介しました。