Identifying an XML Document's Type

By David Adams

Technical Note 07-02

Abstract

This technical note describes how to use the built-in 4th Dimension DOM commands to identify XML document types. A sample database includes the code documented in the technical note.

Overview

XML defines a standard system for the encoding and structuring of data. There are probably millions of different types of XML documents in use by various software applications today. Since all of these documents follow the XML standards, a generalized XML parser, such as the DOM parser in 4th Dimension, can read any XML file successfully.

However, there's a big difference between parsing a document and interpreting its contents. For example, WSDL (Web Service Description Language) documents, SVG (Scalable Vector Graphics) images, and XHTML Web pages are each XML document types. It's unlikely a program designed for editing WSDL files will also understand the rules for rendering an SVG image. Likewise, document formats may evolve over time or come in several closely related versions. As an example, there are four current XHTML document types. Therefore, programs that understand how to interpret specific XML formats need a way of distinguishing between the XML documents they understand and those they do not, and to distinguish between versions of formats they understand.

This technical note describes how to use the built-in 4th Dimension DOM commands to identify target document types. A sample database includes the code documented in the technical note.

How XML Documents Are Distinguished

XML is designed to work across platforms and indefinitely into the future, therefore features such as file types, file extensions, and MIME types may not be adequate. Instead or additionally, programs and open standards use any of several techniques to define file types:

- Document Type Declaration statements at the top of the document.
- Specific root element names.
- Specific namespace declarations.

As an example, consider a simple XHTML document:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Hello world!</title>
</head>
<body>
<p>This is an XHTML document.</p>
</body>
</html>
```

XHTML documents must have a root element named <html> with an XML namespace declaration of `http://www.w3.org/1999/xhtml`, such as the following:

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

The DTD declaration at the top of the document, listed earlier, allows programs to distinguish the specific version of XHTML a document supports.

Tip *If you create custom XML formats for exchange between your own programs, including a unique root element name with a distinctive namespace declaration simplifies identifying documents of interest.*

Reading the Identifying Elements with 4th Dimension

The exact rules for identifying an XML document type depend on the document type itself. We'll look at XHTML as it is a common document type with reasonably typical rules for document identification. The identifying elements are all highlighted in the fragment shown below:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
        "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

The table below lists how to recover each piece of identifying information from the XML tree:

`-//W3C//DTD XHTML 1.1//EN`	**DOM Get XML information** (xmlref;PUBLIC ID)
`http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd`	**DOM Get XML information** (xmlref;SYSTEM ID)
`html`	Locate the root element and read its name with **DOM GET XML ELEMENT NAME**.
`http://www.w3.org/1999/xhtml`	Locate the root element and read its `xmlns` attribute with **DOM GET XML ATTRIBUTE BY NAME**.

Note *XML paths, element names, and attribute names are all case-sensitive. See 4D Technical Note 05-41,* Case-Sensitive Operations in 4th Dimension*, for code to handle case-sensitive comparisons.*

Demo_ReadXHTMLDocument

The sample code discussed next shows how to organize a routine to accept XHTML documents and identify their version. As a convenience, a routine named *DOM_GetRootElementReference* is implemented to simplify locating the root element. (Whitespace adjusted below for legibility.)

```
C_STRING(16;$document_xmlref)
$document_xmlref:=DOM  Parse  XML  source("")`  Balance with a call to DOM CLOSE XML.

If (OK=1)
 ` Assume the following DOCTYPE in the example below.
 ` <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 `      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
 C_TEXT($publicID_text)
 C_TEXT($systemID_text)
 ` -//W3C//DTD XHTML 1.1//EN
 $publicID_text:=DOM  Get  XML  information($document_xmlref;PUBLIC   ID )
 ` http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd
 $systemID_text:=DOM  Get  XML  information($document_xmlref;SYSTEM   ID )
 C_TEXT($documentVersionDescription_text)
 $documentVersionDescription_text:=""

 C_BOOLEAN($documentIsXhtml_b)
 $documentIsXhtml_b:=False`  Set to True below if all tests are satisfied.

 ` A document was parsed, but is it an XHTML document?
 C_STRING(16;$root_xmlref)
 $root_xmlref:=DOM_GetRootElementReference ($document_xmlref)

 If ($document_xmlref#"0000000000000000")`  The reference looks valid
  C_TEXT($name_text)
  DOM  GET  XML  ELEMENT  NAME($root_xmlref;$name_text)

  If (CS_AlphasAreEqual ($name_text;"html"))
   ` The root element is <html>.
   ` Note: Element names are case-sensitive in XML.

   DOM_StartCustomErrorHandling
   C_TEXT($namespace_text)
   $namespace_text:=""
   DOM  GET  XML  ATTRIBUTE  BY  NAME($root_xmlref;"xmlns";$namespace_text)
   DOM_StopCustomErrorHandling

   If ($namespace_text="http://www.w3.org/1999/xhtml")
    ` The root element includes the required namespace declaration.
    $documentIsXhtml_b:=True

    ` Now try to distinguish the XHTML type. There are only
    ` four defined, at the moment.

    ` Note: The statements below use wildcards to ignore the language code.
    ` For example, a source document might include the following public ID:
    ` -//W3C//DTD XHTML 1.0 Strict//EN
    ` For matching purposes, we're only interested in this part of the string:
    ` -//W3C//DTD XHTML 1.0 Strict//
```

```
        Case of
          : (($publicID_text="-//W3C//DTD XHTML 1.0 Strict//@") &
                ($systemID_text="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"))
            $documentVersionDescription_text:="XHTML 1.0 Strict."

          : (($publicID_text="-//W3C//DTD XHTML 1.0 Transitional//@") &
            ($systemID_text="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"))
            $documentVersionDescription_text:="XHTML 1.0 Traditional."

          : (($publicID_text="-//W3C//DTD XHTML 1.0 Frameset//@") &
            ($systemID_text="http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd"))
            $documentVersionDescription_text:="XHTML 1.0 Frameset."

          : (($publicID_text="-//W3C//DTD XHTML 1.1//@") &
            ($systemID_text="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"))
            $documentVersionDescription_text:="XHTML 1.1."

          Else
            $documentVersionDescription_text:="Unrecognized or unsupported XHTML type."
        End case

      End if

    End if

  End if

  If ($documentIsXhtml_b)
    ALERT("The document appears to be an XHTML document."+
            Char(Carriage return )+Char(Carriage return )+$documentVersionDescription_text)
  Else
    BEEP
    ALERT("Warning: "+Char(Carriage return )+Char(Carriage return )+
        "The document does not appear to be XHTML.")
  End if

  DOM CLOSE XML($document_xmlref)` Balances earlier call to DOM Parse XML source.

End if
```

While the method above looks long and complicated, it's nothing more involved than a cascade of tests:

1) Is the document well-formed XML? If so,

2) Does the root element name equal `html` case-sensitively? If so,

3) Does the XML root element include an attribute named `xmlns` with a value of `http://www.w3.org/1999/xhtml`? If so,

4) The document is XHTML. Distinguish the version by comparing the system and public ID against hard-coded constants taken from the XHTML standards.

The exact rules for identifying documents depend on the document type itself. XHTML documents are considered in this note but should only be take as an illustration of the steps generally required to identify an XML document type.

The Routines

The other working methods in the sample database are listed below for reference.

DOM_GetRootElementReference

```
C_STRING(16;$0;$rootNode_xmlref)
C_STRING(16;$1;$noderef)

$noderef:=$1

$rootNode_xmlref:="0000000000000000"` Default to a null reference.

` Store existing error/error handling state.
If (Undefined(Error))
  Error:=0
End if
C_LONGINT($previousValueOfErrorVariable_l)
$previousValueOfErrorVariable_l:=Error
C_STRING(31;$previousErrorMethodName_s)
$previousErrorMethodName_s:=Method called on error
ON ERR CALL("DOM_ErrorTrappingRoutine")

C_TEXT($currentNodeName_text)
$currentNodeName_text:=""
` Default to starting element name.
DOM GET XML ELEMENT NAME($noderef;$currentNodeName_text)
While (OK=1)

  $rootNode_xmlref:=$noderef

  Case of
    : ($currentNodeName_text="")
      OK:=0` We're done scanning.

    : ($currentNodeName_text="#document")
      ` An artificial node above the tree. We're not treating this as a valid ancestor.
      $noderef:=DOM Get first child XML element($noderef;$currentNodeName_text)
      ` We're above the root, try moving down one.
      If (OK=1)` The first child of#document is the root.
        $rootNode_xmlref:=$noderef` This is the root reference.
      End if
      OK:=0` Stop the loop before it moves back up to #document.

    Else
      ` Try to get another parent.
      $noderef:=DOM Get parent XML element($noderef;$currentNodeName_text)

  End case
End while

` Restore previous error/error handling state.
Error:=$previousValueOfErrorVariable_l` Restore original error value.
ON ERR CALL($previousErrorMethodName_s)` Restore original error handler.

$0:=$rootNode_xmlref
```

DOM_StartCustomErrorHandling

The *DOM_StartCustomErrorHandling* routine stores existing error state information and installs a custom error handler.

```
If (Undefined(Error))
  Error:=0
End if

C_STRING(80;DOM_PreviousErrorHandler_s)
C_LONGINT(DOM_PreviousValueOfError_l)
C_LONGINT(DOM_Error)
DOM_PreviousErrorHandler_s:=Method called on error
DOM_PreviousValueOfError_l:=Error
DOM_Error:=0` Assign a value in error method, if run.
Error:=0

ON ERR CALL("DOM_ErrorTrappingRoutine")
```

DOM_StopCustomErrorHandling

The *DOM_StopCustomErrorHandling* routine clears the custom error handler and restores the error state information saved by *DOM_StartCustomErrorHandling*.

```
ON ERR CALL(DOM_PreviousErrorHandler_s)`  Restore original error handler.
Error:=DOM_PreviousValueOfError_l` Restore original value of Error.
```

DOM_ErrorTrappingRoutine

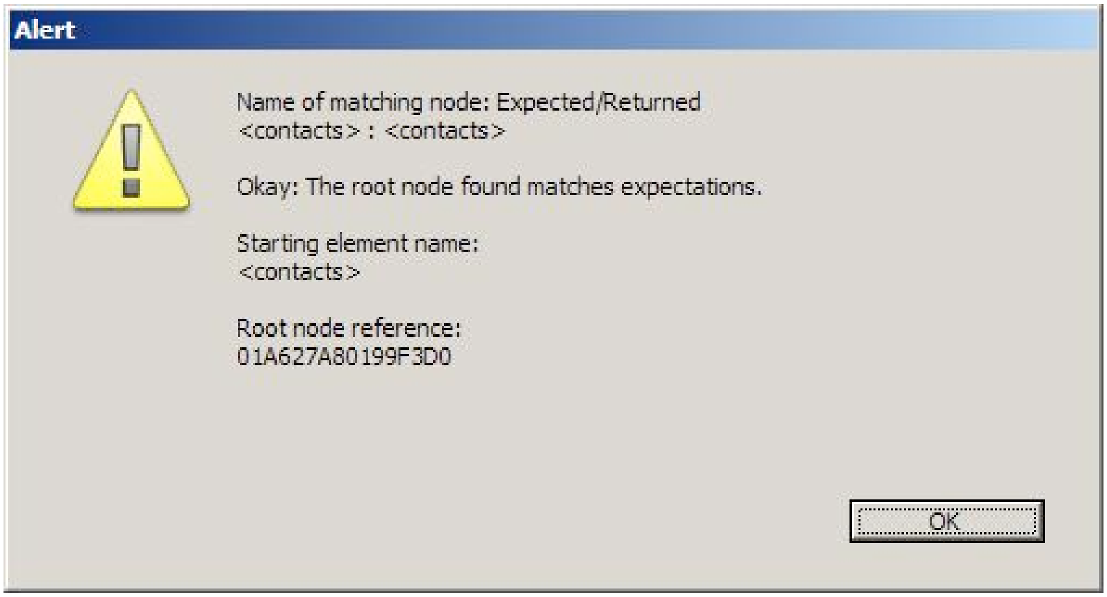
The *DOM_ErrorTrappingRoutine* is installed by *DOM_GetRootElementReference* to trap errors arising from attempting to read invalid node references. The error handler includes a single line of code, listed below:

```
DOM_Error:=Error
```

Note *The sample database also includes the* CS_AlphasAreEqual *routine from Technical Note 05-41,* Case-Sensitive Operations in 4th Dimension.

The Sample Database

The sample database includes the code listed above and a simple test routine named *Test_GetRootElementReference*. The test code is designed to exercise the *DOM_GetRootElementReference* method in various conditions, including passing in good nodes, bad nodes, and the synthetic `#document` node. For each test, the code displays a simple status alert reporting if the test conditions were met, such as the screen shown below:



Name of matching node: Expected/Returned
<contacts> : <contacts>

Okay: The root node found matches expectations.

Starting element name:
<contacts>

Root node reference:
01A627A80199F3D0

Summary

XML document types may be identified by various combinations of DTD values, element names, and element attributes. While the exact rules depend on the specific document type, 4th Dimension's DOM commands provide access to all of the required information. This technical note and its supporting database illustrate how to identify XHTML documents and distinguish between the four defined varieties of XHTML. The same basic steps can be adapted to distinguish other XML document types.