Enhancing the DOM XML Reading Functions

By David Adams

Technical Note 06-40

Overview

4th Dimension 2004 includes a full suite of commands for reading XML documents using the Document Object Model. Using the built-in DOM features, custom routines can scan through an XML tree to read the names, values, and attributes of each node. While the built-in DOM functions are feature complete, there is still room for some refinements. Accompanying this technical note is a sample database that adds automatic error handling, whitespace cleaning, and new node matching options to the native DOM routines. The main routines in the system are listed below:

DOM_AttributesToArrays
DOM_CountAttributes
DOM_CountElementByName
DOM_ElementExists
DOM_FindElementByName
DOM_GetElementName
DOM_GetElementValue
DOM_ReferenceIsValid

String_EqualCaseSensitively

XML_CleanWhitespace
XML_InitWhitespaceCharacters

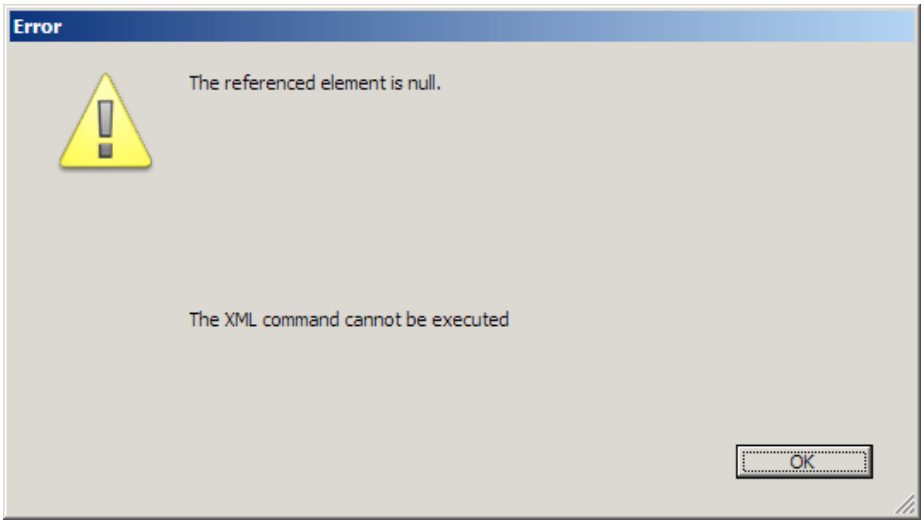
This note takes a quick look at the overall enhancements offered by these routines and then documents the behavior and documentation of each method. Within the database, each method is also documented internally and in the *DOM Read Me* routine. A demonstration screen illustrates the functionality of each behavior.

Background Information and Feature Overview

Tree Navigation and Bad Nodes

4th Dimension's DOM commands render a source XML document or variable as a tree of linked nodes. The DOM command set includes tools for navigating through the tree and reading information from nodes. The tree navigation commands, such as **DOM Get Parent XML element** and **DOM Get first child XML element**, use the OK system variable to indicate when the command has left the tree and reached a non-existent node. This feature makes it possible to navigate through the tree without knowing in advance exactly how many ancestors, siblings, or descendants a particular node has.

However, it also leads to the navigation commands returning references to invalid nodes. This causes problems when reading information, such as an element's name, value, or attributes. For example, calling the built-in **DOM GET XML ELEMENT NAME** command on an invalid node reference leads to a dialog like the one pictured below:



To avoid problems from invalid node references, the database included with this technical note automatically wraps the standard node reading routines in an error handler that suppresses error displays. Additionally, the *DOM_ReferenceIsValid* function offers a simple tool for checking node validity.

Note *For more details on testing DOM node references for validity and managing bad nodes, see Technical Note xx-xx,* ***Avoiding Problems Reading DOM XML Nodes****.*

Whitespace Handling

XML element values commonly include leading or trailing whitespace characters that help make the source XML easier to read. For example, it is common for element values to include extra tabs and carriage returns to format and indent an XML listing. There may be times when the leading and trailing whitespace are meaningful and other times when they are not. 4th Dimension's DOM and SAX commands always assume the whitespace may be meaningful and return it as part of an element's value. The database included with this technical note includes a routine named *XML_CleanWhitespace* that efficiently trims leading and trailing XML whitespace off a block of text. This routine may be called at any time with any value. Internally, it is used by *DOM_GetElementValue* to add whitespace trimming to the built-in **DOM GET XML ELEMENT VALUE** command.

Note *For more details on handling XML whitespace, see Technical Note xx-xx,* ***Cleaning Whitespace from XML Values****.*

Attribute Handling

XML element's may have any number of attributes, such as the id attribute in the element shown below:

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

The built-in DOM commands support reading the attributes of a node by name or index. As a convenience, the sample database includes a function named *DOM_AttributesToArrays* that copies all of a node's attributes into a pair of text arrays.

Apart from throwing an error when called on an invalid node reference, the **DOM Count XML attributes** function crashes certain version of 4th Dimension when called on the `#document` element, a special information-only node above the root of the XML tree. Consequently, the database provides a function named *DOM_CountAttributes* as a safe alternative to the native function.

Note *For more details on handling XML attributes, see Technical Note xx-xx,* ***Enhanced Tools for Reading XML Attributes****.*

Method Documentation

DOM_AttributesToArrays

DOM_AttributesToArrays (Alpha [16];Pointer;Pointer)
DOM_AttributesToArrays (XML reference;->Names array;->Values array)

This routine copies the attributes of an XML node into a pair of text arrays.

Note: The routine does not accept string arrays in the place of text arrays.

DOM_CountAttributes

DOM_CountAttributes (Alpha [16]) : Longint
DOM_CountAttributes (XML reference) : Count of attributes

This routine counts the number of attributes associated with an XML element. It enhances the behavior of **DOM Count XML attributes** with automatic error handling and avoiding reading the attributes of illegal nodes.

DOM_CountElementByName

DOM_CountElementByName (Alpha [16];Text;{Longint}) : Longint
DOM_CountElementByName (XML reference;Target element name;{Element to find max}) : Count of matching elements

This routine counts the number of times an element name appears in an XML tree. The optional Element to find max supports stopping the count when a specific instance is reached. The default value for this optional parameter is MAXLONG.

DOM_ElementExists

DOM_ElementExists (Alpha [16];Text;{Longint}) : Boolean
DOM_ElementExists (XML reference;Target element name;{Instance to find}) : Returns **True** if the requested instance exists.

This routine tests if an element exists within an XML tree. The optional Instance to find parameter supports finding a specific instance of an element name. The default value for this optional parameter is 1.

DOM_FindElementByName

DOM_FindElementByName (Alpha [16];Text;Longint) : Alpha [16]
DOM_FindElementByName (XML reference;Target element name;Instance to find) : Matching node reference or an empty string.

Finds the requested element within an XML tree and returns its node reference. The Instance to find parameter supports finding a specific instance of an element name.

DOM_GetElementName

DOM_GetElementName (Alpha [16]) : Text
DOM_GetElementName (XML reference) : Element name

This routine returns the name of the specified node. It also enhances the **DOM GET XML ELEMENT NAME** command with automatic error handling.

DOM_GetElementValue

DOM_GetElementValue (Alpha [16];{Boolean}) : Text
DOM_GetElementValue (XML reference;{Clean whitespace?}) : Element value

This routine returns the value of the specified node. Additionally, it enhances the **DOM GET XML ELEMENT NAME** command with automatic error handling and optional whitespace cleaning. The default value for the optional Clean whitespace argument is **False**.

DOM_ReferenceIsValid

DOM_ReferenceIsValid (Alpha [16]) : Boolean
DOM_ReferenceIsValid (XML reference) : Element reference is valid?

This routine tests if a node reference is valid.

DOM_ReferenceIsValidOnError

This custom error handler is used internally as a custom error handler by the *DOM_ReferenceIsValid* routine.

DOM_StartCustomErrorHandling

This routine is used internally to record the current error handler name and the current value of the Error system variable, as well as to install a custom error handler.

DOM_StopCustomErrorHandling
This routine is used internally to reverse the operations of the *DOM_StartCustomErrorHandling* routine.

String_EqualCaseSensitively
String_EqualCaseSensitively (Text;Text): Boolean
String_EqualCaseSensitively (Base text;Comparison text) : Equal?

This routine tests if two strings/texts are equal case-sensitively. This functionality is required to compare XML element names, which are always case-sensitive.

XML_CleanWhitespace
XML_CleanWhitespace (Text) : Text
XML_CleanWhitespace (Source text): Result text

This routine returns the source string without leading or trailing XML whitespace.

XML_InitWhitespaceCharacters
This routine initializes an array containing the XML whitespace characters.

Summary

The built-in DOM information reading commands and functions offer a complete set of features, but they can still be refined with some simple code. The sample database included with this technical note makes the built-in commands safer and easier to work with by automatically handling errors on bad nodes, offering whitespace trimming, and simplifying and improving access to XML node attributes.