Cleaning Whitespace from XML Values

By David Adams

Technical Note 06-42

Overview

XML element values commonly include leading or trailing whitespace characters that help make the source XML easier to read. For example, consider the simple XML example below:

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

  <business>ACME Black Dot
    <phone>123 456 789</phone>
  </business>

</contact>
```

What is the value of the element named `business`? The obvious answer, "ACME Black Dot", is incorrect. If using the DOM commands, the complete value includes all of the text between the opening and closing of the `business` tag. If using the SAX commands, the complete value includes all of the text from the start of the business element to the end of the business element or the start of the next element, whichever comes first. The whitespace characters in the XML are shown below:

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

  <business>ACME Black Dot CARRIAGE_RETURN
SPACE SPACE SPACE SPACE<phone>123 456 789</phone>CARRIAGE_RETURN
SPACE SPACE</business>
</contact>
```

The complete value of the `business` element is shown below:

Read with DOM Commands

```
ACME Black Dot CARRIAGE_RETURN
SPACE SPACE SPACE SPACE CARRIAGE_RETURN
SPACE SPACE
```

Read with SAX Commands

```
ACME Black Dot CARRIAGE_RETURN
SPACE SPACE SPACE SPACE
```

Unfortunately, 4th Dimension's element reading commands, such as **DOM GET XML ELEMENT VALUE** and **SAX GET XML ELEMENT VALUE**, don't support trimming leading and trailing whitespace. Fortunately, this missing functionality is easy to write and is implemented in a sample database accompanying this note. The note itself reviews the basic rules for XML whitespace and strategies for trimming it from values.

XML Whitespace

The XML specifications unambiguously identify four characters as whitespace (ASCII codes given here for simplicity since 4th Dimension doesn't support Unicode character references):

Name	ASCII
Tab	9
Line feed	10
Carriage return	13
Space	32

Designing a Trimming Function

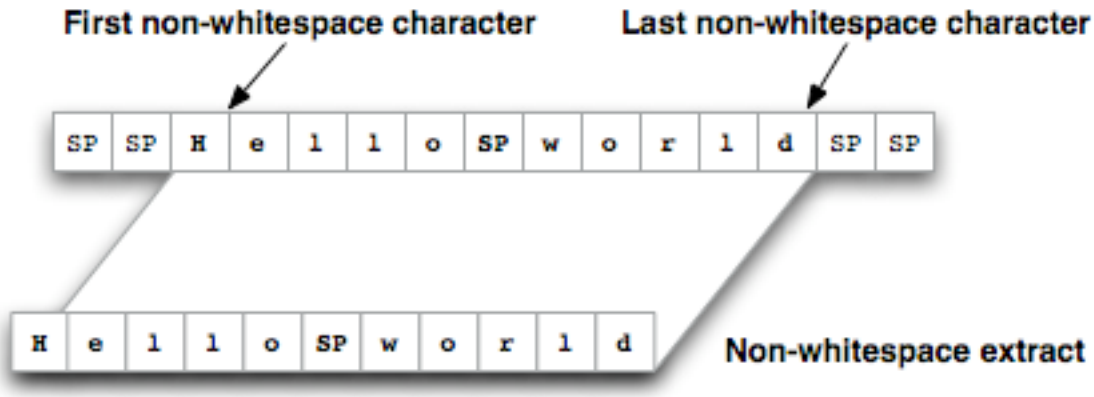
Trimming XML whitespace is a conceptually simple task that can be implemented in many ways. One straightforward approach is to call **Substring** or **Delete string** repeatedly on the original string until all leading and trailing whitespace are eliminated. Imagine the source string below:

```
SPACE SPACE Hello world! SPACE SPACE
```

Using a naive approach, the string needs to be resized once for each leading and trailing character, or four times in the example string. This approach potentially requires 4th Dimension to move memory each time the string is resized. While memory may not have to be moved, and moving matter may not make a meaningful performance difference in many cases, it's not difficult to write a trimming function that only resizes the string once. Instead of continually modifying the string, the function takes three steps:

1. Scan the string from the front to find the first non-whitespace character, if any.
2. Scan the string from the end to find the last non-whitespace character, if any.
3. Extract the non-whitespace characters.

The diagram below identifies the key sections of the `SPACE SPACE Hello world! SPACE SPACE` sample string:



Notice that the space character within the string `Hello world!` is not whitespace. Characters are only considered whitespace when they are leading or trailing.

Trimming Function Implementation

The sample database included with this technical note contains a function named *XML_CleanWhitespace* that implements a trimming function using the three step strategy described above. To simplify the operation, an interprocess array including the four XML whitespace characters is initialized at startup by a call to *XML_InitWhitespaceCharacters*, listed below:

```
ARRAY  STRING(1;<>XML_WhitespaceCharacters_as;4)
<>XML_WhitespaceCharacters_as{1}:=Char(Tab  )
<>XML_WhitespaceCharacters_as{2}:=Char(Line feed )
<>XML_WhitespaceCharacters_as{3}:=Char(Carriage return )
<>XML_WhitespaceCharacters_as{4}:=Char(Space  )
```

The *XML_CleanWhitespace* function is listed below:

```
C_TEXT($0;$result_t)
C_TEXT($1;$source_t)
$source_t:=$1
$result_t:=""

C_LONGINT($firstCharacter_index)
C_LONGINT($lastCharacter_index)
$firstCharacter_index:=0
$lastCharacter_index:=0

`------------------------------------------------------
`  1) Find first non-whitespace character.
`------------------------------------------------------
C_LONGINT($length)
C_LONGINT($index)
C_LONGINT($element)
$length:=Length($source_t)
$index:=0
$element:=0
C_BOOLEAN($done)
$done:=False
```

```
Repeat
   $index:=$index+1

   If ($index>$length)
     $done:=True
   Else  ` Look for the current character in the whitespace array.
   $element:=Find in array(<>XML_WhitespaceCharacters_as;$source_t$index)
     If ($element<0)` The character being tested is not whitespace, so this is what we're looking
for.
       $firstCharacter_index:=$index
       $done:=True
     End if
   End if
Until ($done)

`---------------------------------------------------------
` 2) Find last non-whitespace character.
`---------------------------------------------------------
C_LONGINT($index)
C_LONGINT($element)
$index:=Length($source_t)+1
$element:=0
C_BOOLEAN($done)
$done:=False

Repeat  ` Step backwards through string looking for last non-whitespace character.
   $index:=$index-1

   If ($index=0)
     $done:=True
   Else
     ` Look for the current character in the whitespace array.
     $element:=Find in array(<>XML_WhitespaceCharacters_as;$source_t$index)
     If ($element<0)
       ` The character being tested is not whitespace, so this is what we're looking for.
       $lastCharacter_index:=$index
       $done:=True
     End if

   End if

Until ($done)

`---------------------------------------------------------
`  3) Extract non-whitespace characters.
`---------------------------------------------------------
C_LONGINT($result_length)
$result_length:=$lastCharacter_index-$firstCharacter_index+1

Case of
   : ($lastCharacter_index=0)
     $result_t:=""

   : ($firstCharacter_index=0)
     $result_t:=""

   : ($result_length<1)
```

```
      $result_t:=""
   Else
      $result_t:=Substring($source_t;$firstCharacter_index;$result_length)
End case
$0:=$result_t
```

Additional Comments on the Implementation

The *XML_CleanWhitespace* function tests if a character is whitespace or not by checking the contents of the <>XML_WhitespaceCharacters_as array. Why not simply test the character directly within the code? For example, consider the code fragment below (assume <>Tab, <>LINE_FEED, <>CARRIAGE_RETURN, and <>SPACE have been initialized at startup):

```
If ($index=0)
   $done:=True
Else
   Case of
     : ($source_t$index=<>TAB)
     : ($source_t$index=<>LINE_FEED)
     : ($source_t$index=<>CARRIAGE_RETURN)
     : ($source_t$index=<>SPACE)
     Else ` The character being tested is not whitespace, so this is what we're looking for.
       $lastCharacter_index:=$index
       $done:=True
   End case
End if
```

The advantage of the hard-coding shown above is that it's easy to read and understand. On the other hand, the array-based system used by *XML_CleanWhitespace* is also convenient because the same code can be reused to process other string trimming rules. Using this strategy, changing the array changes the behavior of the routine while leaving the methods logic and code intact and unmodified. For example, it is easy to define a different array that includes a list of unwanted control characters that should also be removed from the front and back of a string. While the same result could be achieved by adding new case statements to a hard-coded trimming routine, it's simpler to define new data and leave the trimming function as generic trimming engine.

Summary

4th Dimension's XML element value reading commands, such as **DOM GET XML ELEMENT VALUE**, **DOM Get parent XML element**, **DOM Get first child XML element** and **SAX GET XML ELEMENT VALUE**, don't support trimming leading and trailing whitespace. Fortunately, this missing functionality is easy to add. This technical note describes an efficient implementation of a generalized trimming function included in the accompanying sample database.