Validating XML Element Names

By David Adams

TN 05-40

Overview

このテクニカルノートの目的は、XML 要素の命名規則に関する情報を提供し、要素名の妥当性を検証するためのメソッドを記述することです。これらの情報は、4th Dimension でカスタム XML を生成する際に有用なものです。

XML Names: Overview of Rules

XML 要素名は、XML の規格で定められている命名規則に従わなくてはなりません。XML の命名規則は 4th Dimension の変数名やメソッド名の制限に似ていますが、同じではありありません。XML 要素名の主な要件をまとまてみました。

- XML 要素名では、4th Dimension の変数名等とは違い、大文字と小文字を区別します。したがって次はいずれも異なるものです。customer、Customer、および CUSTOMER。
- XML 要素名では、ハイフン、ピリオド、表意記号を冒頭に使用することができます。
- XML 要素名は、表意記号、文字、数字、ハイフン、ピリオド、コロン、アンダースコアを含むことができます。
- コロンは、名前空間のために予約されており、その他の意味合いで要素名に使用すべきではありません。
- XML 要素名には、スペース、ノンブレークスペース、タブ、ラインフィード、キャレッジリターンを含むあらゆる種類のスペースを使用してはなりません。
- XML から始まる要素名は、大文字・小文字のすべての組み合わせにおいて、現在また将来の XML 規格のために予約されています。
- XML 要素名の長さに制限はありません。

Case-Sensitivity in 4th Dimension

すでに触れたように、XML の要素名や属性名は大文字と小文字を区別して扱われます。4th Dimension のプログラム言語とデータベースエンジンは、基本的に文字の大小を区別しないので、この点はよく考慮に入れる必要があります。4th Dimension と文字の区別については、テクニカルノート 05-41 Case-Sensitive Operations in 4th Dimension を参照してください。

XML Names: Examples of Good and Bad Names

命名規則の理解を助けるために具体的な例を幾つかここに挙げておきます。

名前	説明
Species_Name	問題ありません。
1Species_Name	数字で始まるので不正な名前です。
Species_Name_1	問題ありません。名前の中には数字を含めることができます。
:Species_Name	冒頭のコロンは名前空間用に予約されています。避けてください。
Species Name	スペースを含んでいるので不正な名前です。
xml_ Species_Name	XML としては問題ありませんが、カスタム XML としては不正な名前です。XML で始まる名前は、大文字・小文字のすべての組み合わせにおいて、現在また将来の XML 規格のために予約されています。

XML Names: Simplifies Rules for 4th Dimension

多くの 4th Dimension プログラマの場合、前述の規則は、以下の注意事項として要約することができます。

- 大文字と小文字を区別する。
- z また A-Z から始まるべき。
- 数字、文字、アンダースコア、ピリオドだけを使用する。
- その他の記号は用途を目的をしっかり理解しているのでない限り、使用しない。

Simplified Rules: Legal First Character

XML 要素名の最初の一文字に使用できるのは、次に挙げた文字のいずれかです。

ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char
58	:	73	I	82	R	95	_	105	i	114	r
65	A	74	J	83	S	97	a	106	j	115	s
66	B	75	K	84	T	98	b	107	k	116	t
67	C	76	L	85	U	99	c	108	l	117	u
68	D	77	M	86	V	100	d	109	m	118	v
69	E	78	N	87	W	101	e	110	n	119	w
70	F	79	O	88	X	102	f	111	o	120	x
71	G	80	P	89	Y	103	g	112	p	121	y
72	H	81	Q	90	Z	104	h	113	q	122	z

: (ASCII 58)記号は、名前空間のコンテキストで使用されるべきであり、その他の場合は使用すべきでない点に留意してください。

Simplified Rules: Legal Body Characters

XML 要素名に使用できるのは、次に挙げた文字のいずれかです。

ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char	ASCII	Char
45	-	57	9	74	J	85	U	101	e	112	p
46	.	58	:	75	K	86	V	102	f	113	q
48	0	65	A	76	L	87	W	103	g	114	r
49	1	66	B	77	M	88	X	104	h	115	s
50	2	67	C	78	N	89	Y	105	i	116	t
51	3	68	D	79	O	90	Z	106	j	117	u
52	4	69	E	80	P	95	_	107	k	118	v
53	5	70	F	81	Q	97	a	108	l	119	w
54	6	71	G	82	R	98	b	109	m	120	x
55	7	72	H	83	S	99	c	110	n	121	y
56	8	73	I	84	T	100	d	111	o	122	z

: (ASCII 58)記号は、名前空間のコンテキストで使用されるべきであり、その他の場合は使用すべきでない点に留意してください。

Name Validation Code: *xmlNameIsValid*

XML の名前に使用されている文字列の妥当性を検証する方法は無数にありますが、このテクニカルノートでは、これまで論じてきた点を単純に実践している *xmlNameIsValid* メソッドを紹介します。メソッドのコードは以下のとおりです。

```
C_BOOLEAN($0;$nameIsValid_b)
C_TEXT($1;$nameSource_t)

$nameSource_t:=$1

$nameIsValid_b:=False

C_TEXT($nameCleaned_t)
$nameCleaned_t:=""
$nameCleaned_t:=xmlNameClean($nameSource_t)

Case of
	¥ ($nameCleaned_t="")
	 $nameIsValid_b:=False
	¥ ($nameSource_t#$nameCleaned_t)
	 $nameIsValid_b:=False

	Else
	 $nameIsValid_b:=True

End case

$0:=$nameIsValid_b
```

*xmlNameIsValidm*メソッドは、調べるべき名前を*xmlNameClean*という別のメソッドに渡してから結果を調べています。結果が空、または元のデータと違っている場合は、名前が妥当ではありません。XMLの名前を検証する処理は、*xmlNameClean*メソッドの中にすべてまとめられており、*xmlNameIsValidm*メソッドは*xmlNameClean*メソッドの呼び出しを簡単にするために用意されています。

Name Utility Code: xmlNameClean

xmlNameIsValidメソッドの中で直接XML要素名の検証をするという方法もあります。しかし、命名規則に関わる部分を別のメソッドに取り分けておけば、再利用がしやすくなります。要項なXML名を生成するために文字列を検証する処理は、色々な場面で必要になってきます。以下は*xmlNameClean*メソッドが使用できるシチュエーションの例です。

End User XML Editor

ジョアンは、ユタ州にあるACME Black Dot社でAcmeDeskを担当しているプログラマです。エンドユーザから、既存のフィールドデータおよび計算式に基づいてカスタムXMLを生成できるようにして欲しいという要望が出されています。ジョアンは、フィールドや計算式を選択するとエクスポートができるようなエディタを開発することにしました。ユーザは自由に要素名を設定することができ、生成されたXMLを色々なプログラムで使用したいと考えています。プロトタイプテストの結果、ユーザはしばしば不正なXMLを生成してしまうことが分かりました。そのような問題を回避するため、ジョアンは不正な文字を除去する次のようなコードが有用であることに気づきました。

```
xml_name_entry_area:=XMLNameClean(xml_name_entry_area)
```

Field-to-Element Export

ブルースは、ACME Black Dot社でジョアンの部下として働いています。ジョアンからは、データベースに保存されたすべてのレコードをXMLファイルに書き出すようなコードを作るタスクを割り当てられました。XMLが理解しやすいように、元のフィールド名やテーブル名をXMLの要素名に反映させるようにと指示されています。ブルースは、XMLには慣れていないものの、4th Dimensionの経験は豊富です。残念ながら、生成したXMLは、多くのXMLツールにはじかれてしまいます。ACME Black Dot社は後からXMLを導入したため、XMLの命名規則に従っていなかったことが原因でした。ジョアンに相談してみると、XMLNameCleanルーチンを使ってみてはどうかと勧められました。ブルースは次のコードを記述して問題っを解決しました。

```
$element_name:=xmlNameClean (Field   name($field_pointer))
```

xmlNameCleanメソッドのコードは次のようなものです。

```
C_TEXT($0;$result_t)
C_TEXT($1;$source_t)

$source_t:=$1
$result_t:=""

C_LONGINT($source_length)
$source_length:=Length($source_t)

C_BOOLEAN($continue_b)
$continue_b:=True

  ` Verify that source name is not empty.
If ($source_length=0)  ` Test for empty string (invalid)
 $result_t:=""
 $continue_b:=False
End if
  ` Verify that first character is legal.
If ($continue_b)
   ` Valid first characters for an XML name are (Inclusive)
   ` A-Z a-z _ :
   ` The colon has a very specific use and meaning so don't use it unless you
know how and why.
 C_LONGINT($firstChar_ascii)
 $firstChar_ascii:=Ascii($source_t[[1]])
 Case of
  ¥($firstChar_ascii>=65) & ($firstChar_ascii<=90)  ` A-Z are valid first charact
ers.
   $result_t:=$source_t[[1]]
  ¥($firstChar_ascii>=97) & ($firstChar_ascii<=122)  ` a-z are valid first charac
ters.
   $result_t:=$source_t[[1]]
  ¥($firstChar_ascii=95) | ($firstChar_ascii=58)  ` Underscore (95) and Colon
(58) may be valid.
   $result_t:=$source_t[[1]]
  Else
   $result_t:=""
   $continue_b:=False  ` Bad first character, terminate.
 End case
End if

  ` Remove any illegal following characters.
If ($continue_b)
   ` Valid body characters for an XML name are (Inclusive)
   ` A-Z a-z _ : - . 0-9
   ` The colon has a very specific use and meaning so don't use it unless you
know how and why.
 C_LONGINT($index)
 C_LONGINT($ascii)
 For ($index;2;$source_length)
  $ascii:=Ascii($source_t[[$index]])
```

```
   Case of
    ¥ ($ascii>=65) & ($ascii<=90)  ` A-Z are valid body characters.
     $result_t:=$result_t+$source_t[[$index]]
    ¥ ($ascii>=97) & ($ascii<=122)  ` a-z are valid body characters.
     $result_t:=$result_t+$source_t[[$index]]
    ¥ ($ascii=95) | ($ascii=58)  ` Underscore (95) and Colon (58) are valid bod
y characters.
     $result_t:=$result_t+$source_t[[$index]]
    ¥ ($ascii=45) | ($ascii=46)  ` Hyphen (45) and Period (46) are valid body c
haracters.
     $result_t:=$result_t+$source_t[[$index]]
    ¥ ($ascii>=48) & ($ascii<=57)  ` 0-9 are valid body characters.
     $result_t:=$result_t+$source_t[[$index]]
    Else
       ` Bad character, don't include it.
   End case
  End for
End if
   ` Verify that name doesn't start with XML.
If (Length($result_t)>=3)
    ` Note: 4D's string comparisons are not case-sensitive, so the same result
    ` could be achieved with this line of code:
    ` If ($result_t="xml@")
    ` I've taken this more pedantic approach to make the rule explicit.
  C_LONGINT($charOne_ascii)
  C_LONGINT($charTwo_ascii)
  C_LONGINT($charThree_ascii)
  $charOne_ascii:=Ascii($result_t[[1]])
  $charTwo_ascii:=Ascii($result_t[[2]])
  $charThree_ascii:=Ascii($result_t[[3]])
  If ($charOne_ascii=88) | ($charOne_ascii=120)  ` 'X' or 'x'
   If ($charTwo_ascii=77) | ($charTwo_ascii=109)  ` 'M' or 'm'
    If ($charThree_ascii=76) | ($charThree_ascii=108)  ` 'L' or 'l'
     $result_t:=""  ` Name is bad. Would need to call routine recursively after t
rim if didn't clear here.
     $continue_b:=False
    End if
   End if
  End if
End if

$0:=$result_t
```

Implementation Notes: xmlNameClean

xmlNameClean メソッドに含まれている機能を応用する方法は無数にあります。明快なのは、ハードコーディングされているキャラクターコードテストをデータドリブンソリューションで置き換えることです。そのような場合においては、最初の文字に使用できる有効なアスキーコードは、メソッドを実行する前に配列などに納められます。最初の文字および本体を検証するコードは、次の再帰メソッドのようになります。

```
  ` Verify that first character is legal.
If ($continue_b)
       C_LONGINT($firstChar_ascii)
       $firstChar_ascii:=Ascii($source_t[[1]])
       C_LONGINT($element)
       $element:=Find in array(array_of_legal_first_character_ascii_codes;$firstCh
ar_ascii)
       If ($element>0)  ` Good character.
        $result_t:=$source_t[[1]]
       Else
        $result_t:=""
        $continue_b:=False  ` Bad first character, terminate.
       End if
End if
  ` Remove any illegal following characters.
If ($continue_b)
       C_LONGINT($index)
       C_LONGINT($ascii)
       For ($index;2;$source_length)
        $ascii:=Ascii($source_t[[$index]])
        C_LONGINT($element)
        $element:=Find in array(array_of_legal_body_character_ascii_codes;$ascii)
        If ($element>0)
          $result_t:=$result_t+$source_t[[$index]]
        Else
           ` Bad character, don't include it.
        End if
       End for
End if
```

上の再帰メソッドが実際そうであるように、データドリブンソリューションには、メリットとデメリットがあります。

短所　配列の内容を知らない限り、コードを完全に理解することはできない。

長所　データがハードコードされていないため、独立したテスト、印刷、表示、エクスポートなどの目的のために読み込むことができる。

長所　許されるキャラクターが変わった場合、データを変更するだけで済む。コードの編集は不要であり、余計なバグも発生しない。

長所　メソッドは短く、簡潔で、読みやすい。

長所　制御フローが少なく、論理的な不整合によるミスの可能性が少ない。

プログラマの中には、両者の実行速度の違いやメモリ使用量の差を気にする人がいるかもしれませんが、XML の名前文字列の長さと今日のマシンスペックを考えれば、それらは実世界においてほとんど意味を持たない程度の差に過ぎないと言わざるを得ません。

Summary

このテクニカルノートでは、XML 要素の名前を決める際に考慮すべき基本的なルールについて振り返り、4th Dimension プログラマ向けの弔意事項をまとめ、要素名を検証するためのサンプルコードを提供しました。このコード、もしくは類似したコードを追加することによって、XML を生成する際に直面しかねない問題を未然に防ぐことができるはずです。