Title: **Removing Attribute Value Normalization from XMLAttributes**

Author: Jan-Eike Michels

Source: U.S.A.

Status: SQL:2003 TC and SQL:200x WD change proposal

Date: March 8, 2004

Abstract

This paper points out an inconsistency between SQL/XML on one side and XQuery and other XML-related specifications on the other side with respect to whether values for an XML attribute should be normalized as the current rules of XMLAttributes prescribe. This inconsistency is fixed by aligning SQL/XML with the other XML-related specification and removing the attribute value normalization.

TC changes are needed but currently not included.

References

[Infoset]   "XML Information Set", W3C Recommendation dated 24 October, 2001, available at http://www.w3.org/TR/2001/REC-xml-infoset-20011024

[SQL/XML:2003]   "ISO International Standard (IS) Database Language SQL - Part 14: SQL/XML", ISO/IEC 9075-14:2003

[SQL:2003 TC]   Stephen Cannan (ed), to be published

[SQL/XML WD]   Jim Melton (ed), "Working Draft (WD) Database Language SQL - Part 14: SQL/XML", ISO/IEC JTC1/SC32 WG3:SIA-010 = ANSI INCITS H2-2003-427

[XML 1.0 SE]   "Extensible Markup Language (XML) Version 1.0 (second edition)", W3C Recommendation dated 2 October 2000, available at http://www.w3.org/TR/REC-xml

[XML 1.0 TE]   "Extensible Markup Language (XML) Version 1.0 (Third Edition)", W3C Recommendation dated 4 February 2004, available at http://www.w3.org/TR/REC-xml

[XQuery 1.0]   "XQuery 1.0: an XML query language", W3C Working Draft dated 12 November 2003, available at http://www.w3.org/TR/2003/WD-xquery-20031112/

[XSLT 1.0]   "XSL Transformations (XSLT) Version 1.0", W3C Recommendation dated
                         16 November 1999, available at http://www.w3.org/TR/xslt

1. Introduction

1.1 The current situation in [SQL/XML WD]

The General Rules in Subclause 6.10, <XML element>, of [SQL/XML WD] that govern the generation of an XML attribute can be summarized (and simplified) as follows (GR 3):

1. The value *AV* that the user specified (through a <value expression>) is mapped according to the GRs of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", yielding *CAV*. Subclause 9.16 is responsible for converting *AV* to Unicode. Additionally, if the most specific type of *AV* is a character string, then GR 8)a)ii) of Subclause 9.16, requires that each instance of "&" (U+0026) be replaced by "&", each instance of "<" (U+003C) be replaced by "<", each instance of ">" (U+003E) be replaced by ">", and each instance of Carriage Return (U+000D) be replaced by "".

2. An XML attribute information item is created with the usual properties, among others the [normalized value], which is set to the result of applying the rules of [XML 1.0 SE], section 3.3.3, "Attribute Value Normalization", to *CAV*.

Step two seems to be in sync with [Infoset], which says this about the [normalized value] property:

> **[normalized value]** The normalized attribute value (see 3.3.3 Attribute-Value Normalization [XML]).

Looking at section 3.3.3, "Attribute Value Normalization", of [XML 1.0 SE], one can see that the algorithm for attribute value normalization, among other things, undoes the escaping that was performed in step 1 above. For the convenience of the reader, we quote here the algorithm from [XML 1.0 SE]:

> Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize the attribute value by applying the algorithm below, or by using some other method such that the value passed to the application is the same as that produced by the algorithm.
>
> 1. All line breaks must have been normalized on input to #xA as described in **2.11 End-of-Line Handling**, so the rest of this algorithm operates on text normalized in this way.
>
> 2. Begin with a normalized value consisting of the empty string.
>
> 3. For each character, entity reference, or character reference in the unnormalized attribute value, beginning with the first and continuing to the last, do the following:

- For a character reference, append the referenced character to the normalized value.

- For an entity reference, recursively apply step 3 of this algorithm to the replacement text of the entity.

- For a white space character (#x20, #xD, #xA, #x9), append a space character (#x20) to the normalized value.

- For another character, append the character to the normalized value.

If the attribute type is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

[...]

All attributes for which no declaration has been read should be treated by a non-validating processor as if declared **CDATA**.

Please note that the algorithm speaks of the "XML processor" performing the normalization (i.e., during the parsing of an XML document). But in [SQL/XML WD], the normalization algorithm is used in the context of attribute constructors.

Let the following examples illustrate the above theoretical explanation.

1. `XMLElement(Name "e", XMLAttributes('J&E' as "att"))`
   The & in the attribute value is first mapped to & (according to step 1 above)
   => J&E
   Then this is normalized (according to step 2 above) to
   => J&E
   and stored as the normalized attribute value.
   Serialization would return:
   `<e att="J&E"></e>`

2. `XMLElement(Name "e", XMLAttributes('J&E' as "att"))`
   Again the & is mapped to & (according to step 1 above)
   => J&amp;E
   Then this is normalized (according to step 2 above) to
   => J&E
   and stored as the normalized attribute value.
   Serialization would return:
   `<e att="J&amp;E"></e>`

As the examples show, it does not seem necessary to first map certain characters which is immediately undone by the attribute value normalization algorithm. Serialization is independent of those two steps and would always guarantee that valid XML (with entity/character references where needed) is generated.

Let's see what other recommendations/specification have to say on this topic.

1.2 [XSLT 1.0] and attribute construction

[XSLT 1.0] does not perform attribute normalization as is evidenced by section 7.1.3, "Creating Attributes with xsl:attribute". Partly quoted here with emphasis added by the present author.

```
<!-- Category: instruction -->
<xsl:attribute
      name = { qname }
      namespace = { uri-reference }>
      <!-- Content: template -->
</xsl:attribute>
```

[...]

NOTE: When an xsl:attribute contains a text node with a newline, then the XML output must contain a character reference. For example,

<xsl:attribute name="a">x

y</xsl:attribute>

will result in the output

a="x
y"

(or with any equivalent character reference). The XML output cannot be

a="x

y"

This is because XML 1.0 requires newline characters in attribute values to be normalized into spaces but requires character references to newline characters not to be normalized. **The attribute values in the data model represent the attribute value after normalization.** If a newline occurring in an attribute value in the tree were output as a newline character rather than as character reference, then the attribute value in the tree created by reparsing the XML would contain a space not a newline, which would mean that the tree had not been output correctly.

1.3 [XQuery 1.0] and attribute construction

Section 3.7.1.1, "Attributes", in [XQuery 1.0] deals with the construction of attributes in direct element constructors. Partly quoted here with emphasis added by the present author.

[...]

Conceptually, an attribute (other than a namespace declaration attribute) in a direct element constructor is processed by the following steps:

1. **Predefined entity references and character references in the attribute content are expanded into their referenced strings, as described in 3.1.1 Literals.**

2. Each consecutive sequence of literal characters in the attribute content is treated as a string containing those characters. **Whitespace in attribute content is normalized according to the rules for "Attribute Value Normalization" in [XML 1.0] (each whitespace character is replaced by a space (#x20) character.)**

3. **Each enclosed expression is converted to a string as follows:**

   a. Atomization is applied to the value of the enclosed expression, converting it to a sequence of atomic values.

   b. If the result of atomization is an empty sequence, the result is the zero-length string. Otherwise, each atomic value in the atomized sequence is cast into a string. If any of these atomic values cannot be cast into a string, a dynamic error [err:XQ0052] is raised.

   c. The individual strings resulting from the previous step are merged into a single string by concatenating them with a single space character between each pair.

4. Adjacent strings resulting from the above steps are concatenated with no intervening blanks. The resulting string becomes the string value of the attribute node.

[...]

So, it seems as if at least some normalization is performed for literal content of an attribute value in a direct attribute constructor (step 2 above). However, no normalization is applied to the result of expressions in direct attribute constructors (step 3 above). This is illustrated by the following two examples:

Attribute normalization only applies to literal content of the direct attribute constructor:

```
<new attr="x         y"/>
->
<new attr="x y"/>
```

Whereas the result of expressions within a direct attribute constructor is not normalized:

```
<new attr="{'x        y'}"/>
->
<new attr="x        y"/>
```

In addition to direct element constructors, XQuery also has computed attribute constructors (3.7.3.2, "Computed Attribute Constructors", in [XQuery 1.0]). Again, partly quoted here with emphasis added by the present author.

> The content expression of a computed attribute constructor is processed as follows:
>
> 1. **<u>Atomization</u> is applied to the value of the content expression**, converting it to a sequence of atomic values.
>
> 2. If the result of atomization is an empty sequence, the value of the attribute is the zero-length string. Otherwise, **each atomic value in the atomized sequence is cast into a string**. If any of these atomic values cannot be cast into a string, a <u>dynamic error</u> [err:XQ0052] is raised.
>
> 3. **The individual strings resulting from the previous step are merged into a single string by concatenating them with a single space character between each pair. The resulting string is the string value of the attribute**.

As can be seen, no mentioning of normalization is made in computed attribute constructors in [XQuery 1.0], as the following example illustrates:

```
attribute attr {"x        y"}
->
attr="x         y"
```

Since [XQuery 1.0] provides three options (direct attribute constructors with literal content, direct attribute constructors with expressions, and computed attribute constructors), the questions is, which of these is closest in behavior to the SQL/XML XMLAttributes pseudo-function. Given that there is no direct attribute constructor in SQL/XML but only a pseudo-function that generates (computes) an attribute information item given an attribute name and a value expression determining the attribute value, we conclude that XQuery's computed attribute constructor is the best guidance if we were to follow [XQuery 1.0].

1.4 What should be done for SQL/XML?

We have at least three possibilities (two of which lead to the same result) of how we could handle attribute value normalization in SQL/XML:

1. Leave everything as is (as described in Section 1.1, "The current situation in [SQL/XML WD]", on page 2)

2. Change it to not performing any normalization (as outlined in Section 1.2, "[XSLT 1.0] and attribute construction", on page 4)

3. Change it to be aligned with XQuery (as outlined in Section 1.3, "[XQuery 1.0] and attribute construction", on page 4), which is (if we follow the behavior of computed attribute constructor) equivalent to not performing any normalization.

It seems strange that SQL/XML mandates the mapping of special characters (*i.e.*, &, <, >, and CR) followed by attribute value normalization, while two specifications ([XSLT 1.0] and [XQuery 1.0]) closely tied to XML do not do this.

So why should SQL/XML be different? We think that this difference is a bug in SQL/XML and it should be fixed by not mandating or even mentioning the normalization of constructed attribute values (as well as the mapping of certain characters before the normalization takes place).

1.5 Proposed Solution for [SQL/XML WD]

Since Subclause 6.10, <XML element>, maps the attribute value to Unicode by invoking the GRs of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types" and then normalizes the value, we propose to parameterize the invocation of Subclause 9.16 to indicate that certain characters ("&", "<", ">", and CR) shall not be mapped to their character references, when Subclause 9.16 is invoked from Subclause 6.10. Additionally, attribute value normalization will be removed from Subclause 6.10.

Since Subclause 9.16 is also invoked from other Subclauses that need the mapping of certain characters to their character references to be performed, the rules that invoke Subclause 9.16 from those Subclauses are changed to indicate that the mapping should take place.

1.6 Changes to [SQL:2003 TC]

Since the previously described problem is also a bug in [SQL/XML:2003], we propose equivalent changes to the ones described above for the [SQL:2003 TC] as well.

2. Proposal conventions

This proposal uses the following conventions:

1. SMALLCAPS	denote numbered editorial instructions;
~~strikeout~~	denotes existing text to be deleted;
boldface	denotes new text to be inserted;
plain	denotes existing text to be retained,
[Note:...]	brackets enclose italicized notes to the proposal reader
	boxes surround "editing tags," which are part of the document (not instructions to the editor) and may be deleted, inserted, modified or retained, depending on the typeface within the box

3. Proposal for [SQL/XML WD]

> *[Note to the Editor and Reader: The author of the
> present paper is aware of at least one other paper
> (but maybe more) that also touches the same
> Subclauses and potentially the same rules as the
> present one. To help the Editor in applying those
> proposals it might be necessary to produce one
> paper that takes all those changes into account.]*

3.1 Changes to Subclause 6.10, <XML element>.

1. MODIFY GENERAL RULE 3) AS SHOWN HERE:

   3) Let *i* range from 1 (one) to *n*. If the value of AV_i is not null, then:

   a) Let CAV_i be the result of applying the General Rules of Subclause 9.16,
      "Mapping values of SQL data types to values of XML Schema data types",
      to AV_i, ~~and~~ *ENC* as the *ENCODING*, **and _False_ as *CHARMAPPING*,**
      resulting in a character string CAV_i of Unicode characters.

   b) Let AII_i be an XML attribute information item having the following
      properties:

      i) The [local name] property is the XML QName local part of ANC_i.

      ii) The [prefix] property is the XML QName prefix of ANC_i, if any;
          otherwise, "no value".

      iii) The [namespace name] property is $NSURI_i$.

      iv) The [normalized value] is ~~the result of applying the rules of [XML],
          section 3.3.3, "Attribute Value Normalization", to~~ CAV_i.

      v) The [specified] property indicates that the attribute was specified rather
         than defaulted.

      vi) The [attribute type] property is "CDATA".

      vii) The [references] property is "no value".

      viii) The [owner element] property is *EI*.

3.2 Changes to Subclause 6.11, <XML forest>.

1. MODIFY GENERAL RULE 3) "CASE:" B) "OTHERWISE:" II) "FOR EACH *I*..." 2) "CASE:" B) "OTHERWISE:" AS SHOWN HERE:

> B) Otherwise, let CS_i be result of applying the General Rules of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", to V_i and *ENC* as the *ENCODING***, and _True_ as the *CHARMAPPING*. Let C_i be the result of
>
> ```
> XMLPARSE (CONTENT CS_i PRESERVE WHITESPACE)
> ```

3.3 Changes to Subclause 9.12, Mapping an SQL table to an XML element or an XML forest.

1. MODIFY GENERAL RULE 6) "FOR *I* RANGING..." B) "FOR *J* RANGING..." V) "CASE:" 3) "OTHERWISE:" A) "LET $XMLV_j$..." AS SHOWN HERE:

> A) Let **XMLVj** be the result of applying the mapping defined in Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", using V_j as the SQL data value *DV*, *NULLS* as the choice of whether to map null values to absent elements (absent) or to elements that are marked with `xsi:nil="true"` (nil), ~~and~~ *ENCODING* as the choice of whether to encode binary strings in base64 or in hex**, and _True_ as *CHARMAPPING*.

3.4 Changes to Subclause 9.16, Mapping values of SQL data types to values of XML Schema data types.

1. INSERT A NEW SYNTAX RULE AFTER SR 3) "LET *ENCODING* BE..." AS SHOWN HERE:

> **3.1)Let *CHARMAPPING* be the choice of whether to replace certain characters ("&" (U+0026), "<" (U+003C), ">" (U+003E), and Carriage Return (U+000D)) with their character references (_True_) or not (_False_).**

2. MODIFY GENERAL RULE 8) AS SHOWN HERE:

> 8) Case:
>
> > a) If *SQLT* is a character string type, then:
> >
> > > i)   Let *CS* be the character set of *SQLT*. Let **XMLVRAW** be the result of mapping *SQLV* to Unicode using the implementation-defined mapping of character strings of *CS* to Unicode. If any Unicode code point in **XMLVRAW** does not represent a valid XML character, then an exception condition is raised: *SQL/XML mapping error — invalid XML character.*
> > >
> > > ii) **Case:**
> > >
> > > > 1) **If *CHARMAPPING* is _True_, then let** ~~Let~~ *XMLV* be *XMLVRAW,* with each instance of "&" (U+0026) replaced by "&", each

instance of "<" (U+003C) replaced by "<", each instance of ">" (U+003E) replaced by ">", and each instance of Carriage Return (U+000D) replaced by "".

2) **Otherwise, let *XMLV* be *XMLVRAW.***

b) [...]

3.5 Changes to Subclause 10.9, Construction of an XML element.

1. MODIFY GENERAL RULE 4) "CASE:" C) "OTHERWISE," III) "FOR EACH *J*..." 1) "CASE:" B) "OTHERWISE,..." AS SHOWN HERE:

B) Otherwise, let $XMLV_j$ be the result of applying the General Rules of Subclause 9.16, "Mapping values of SQL data types to values of XML Schema data types", to V_j **and *True* as the *CHARMAPPING*.** $XMLV_j$ is a character string of Unicode characters. Let CEC_j be the result of

```
XMLPARSE (CONTENT XMLVj PRESERVE WHITESPACE)
```

4. Proposal for [SQL:2003 TC]

4.1 TBD.

5. Checklist

Concepts	n/a
Access Rules	n/a
Conformance Rules	n/a
Lists of SQL-statements by category	n/a
Table of identifiers used by diagnostics statements	n/a
Collation coercibility for character strings	n/a
Closing Possible Problems	n/a
Any new Possible Problems clearly identified	no
Reserved and non-reserved keywords	no
SQLSTATE tables and Ada package	n/a
Information and Definition Schemas	n/a
Implementation-defined and –dependent Annexes	no
Incompatibilities Annex	no
Embedded SQL and host language implications	n/a
Dynamic SQL issues: including descriptor areas	n/a
CLI issues	no
MED issues	no
SQL/XML issues	yes

- End of paper -