Title: BLOBs and XMLParse/XMLSerialize

Author: Jan-Eike Michels
Source: U.S.A.
Status: SQL/XML:200x WD change proposal
Date: March 8, 2004

Abstract

We propose new functionality that allows:

1. to serialize an XML value as a BLOB value, and
2. to parse the contents of a BLOB value as an XML value.

References

[SQL/XML:2003] Jim Melton (ed), “ISO International Standard (IS) Database Language SQL - Part 14: SQL/XML”, ISO/IEC 9075-14:2003

[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 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

1. Introduction

1.1 Rationale

SQL/XML currently features the XMLParse pseudo-function that allows a character string to be parsed as an XML value and the XMLSerialize pseudo-function that allows an XML value to be serialized as a character string. Though these functions are very useful, they do have some shortcomings.

For example, if those functions are used to transfer a serialized XML value to/from a client, implicit character set conversion could occur with the consequence that the character set of the

string value (also referred to as external encoding) could be different from the internal encoding[1] that can be specified in the XML value itself which could lead to data corruption.

Another drawback of the current approach is that the serialized XML value has to be represented in a character set that an implementation explicitly supports.

To make the XMLParse and XMLSerialize pseudo-functions more useful, we propose to enhance both of these functions by allowing binary large objects (BLOBs) as input and output, respectively.

BLOBs have the advantage that they do not have an external encoding (i.e., a character set) associated with them, which could contradict the internal encoding present in the serialized version of the XML value. Also, BLOBs allow an implementation to support encodings for XMLParse and XMLSerialize that are not explicitly supported as character sets for character string types.

1.2 Proposed Solution

In [SQL/XML:2003], there are two features for each of XMLParse and XMLSerialize and the conversion from/to character strings as follows:

1. X060, “XMLParse: CONTENT option”,
2. X061, “XMLParse: DOCUMENT option”,
3. X070, “XMLSerialize: CONTENT option”, and
4. X071, “XMLSerialize: DOCUMENT option”

We propose to mirror these features for the new functionality. I.e., we propose four new features for [SQL/XML WD] as follows:

1. X065, “XMLParse: BLOB input and CONTENT option”,
2. X066, “XMLParse: BLOB input and DOCUMENT option”,
3. X072, “XMLSerialize: BLOB serialization and CONTENT option”, and
4. X073, “XMLSerialize: BLOB serialization and DOCUMENT option”

To avoid confusions between the existing features and the new ones, we also propose to rename the existing features as follows:

1. X060, “XMLParse: **Character string input and** CONTENT option”,
2. X061, “XMLParse: **Character string input and** DOCUMENT option”,
3. X070, “XMLSerialize: **Character string serialization and** CONTENT option”, and

---

1. An internal encoding can be specified in the prolog of an XML document using an *Encoding Declaration* (see also production rule [80] EncodingDecl in [XML 1.0 TE]); e.g., <?xml encoding='UTF-8'?> or <?xml encoding='EUC-JP'?>.

4. X071, "XMLSerialize: **Character string serialization and** DOCUMENT option"

As mentioned earlier, an XML value serialized as a BLOB carries only an internal encoding. For XMLSerialize and character strings, the current rules derive this internal encoding from the character set associated with the character string type of the character string. For BLOB the internal encoding cannot be derived in the same way. Therefore, we propose to give the user an option to specify which encoding to use. If this option is not specified, then an implementation-defined encoding is the default. It will be implementation-defined which encodings are supported.

Following are a few examples of the new functionality:

1. Assume that *bv* is a BLOB value that contains a sequence of octets representing a well-formed XML document (the syntax of XMLParse does not change).
   `XMLParse(DOCUMENT bv PRESERVE WHITESPACE)`
2. Assume that *xv* is an XML value. The result is a BLOB value with the (implementation-defined) default size for BLOBs, containing the serialized version of the XML value with an implementation-defined internal encoding (by not explicitly specifying an encoding).
   `XMLSerialize(CONTENT xv AS BLOB)`
3. Assume that *xv* is an XML value containing a well-formed XML document. The result is a BLOB value with a size of one MB, containing the serialized version of the XML value with UTF-8 as the internal encoding (since UTF8[2] was explicitly specified, assuming that the implementation supports UTF-8).
   `XMLSerialize(DOCUMENT xv AS BLOB(1M) ENCODING UTF8)`
4. Assume that *xv* is an XML value. The result is a BLOB value with a size of two MBs, containing the serialized version of the XML value with UTF-16 as the internal encoding (since UTF16 was explicitly specified, assuming that the implementation supports UTF-16).
   `XMLSerialize(CONTENT xv AS BLOB(2M) ENCODING UTF16)`

1.3 Issues not addressed

The changes in this paper do have some implications on the host language bindings of the XML type and on parameters and return values of type XML of external routines, which under the covers use the rules of XMLParse and XMLSerialize. These implications are not addressed by the present paper, but are left for a follow-on paper.

2. Note that the name for the encoding that follows the ENCODING keyword is an <SQL language identifier> which places certain restrictions on the allowed characters. Therefore the allowed encoding names have to be implementation-defined; e.g., UFT-8 would not be an allowed <SQL language identifier>, but UTF8 is. An implementation might additionally also allow other identifiers to denote the UTF-8 encoding.

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 5.1, <token> and <separator>

1. MODIFY THE FORMAT OF <NON-RESERVED WORD> AS SHOWN HERE:

```
<non-reserved word> ::=
       !! All alternatives from ISO/IEC 9075-2
     | ABSENT | ACCORDING
     | ...
     | EMPTY
     | ENCODING
     | HEX
     | ...
```

3.2 Changes to Subclause 6.5, <string value function>.

1. MODIFY THE FORMAT AS SHOWN HERE:

```
<character value function> ::=
       !! All alternatives for ISO/IEC 9075-2
     | <XML character string serialization> ~~serialize>~~
```

```
<XML character string serialization> ~~serialize>~~ ::=
     XMLSERIALIZE <left paren> <document or content> <XML
       value expression> AS <data type> <right paren>

<document or content> ::=
       DOCUMENT
     | CONTENT

<blob value function> ::=
       !! All alternatives for ISO/IEC 9075-2
     | <XML binary string serialization>

<XML binary string serialization> ::=
     XMLSERIALIZE <left paren> <document or content> <XML
       value expression> AS <data type> [ ENCODING <XML
       encoding specification> ] <right paren>

<XML encoding specification> ::= <XML encoding name>

<XML encoding name> ::= <SQL language identifier>
```

2. MODIFY THE SYNTAX RULES AS SHOWN HERE:

1) | Replace SR 2) | The declared type of <character value function> is the declared type of the immediately contained <character substring function>, <regular expression substring function>, <fold>, <transcoding>, <character transliteration>, <trim function>, <character overlay function>, <normalize function>, <specific type method>, or <XML **character string serialization>** ~~serialize>~~.

2) | Insert this SR | If <XML **character string serialization>** ~~serialize>~~ is specified, then <data type> shall be a character string type. The declared type of <XML **character string serialization>** ~~serialize>~~ is <data type>.

3) | Replace SR 15) | The declared type of <blob value function> is the declared type of the immediately contained <blob substring function>, <blob trim function>, <blob overlay function>, or <XML binary string serialization>.

4) | Insert this SR | If <XML binary string serialization> is specified, then:

a) <data type> shall be a binary string type.

b) The declared type of <XML binary string serialization> is <data type>.

c) If <XML encoding specification> is not specified, then an implementation-defined <XML encoding name> is implicit.

d) The supported <XML encoding name>s are implementation-defined.

3. MODIFY THE GENERAL RULES AS SHOWN HERE:

   1) | Replace GR 2) | The result of <character value function> is the result of the immediately contained <character substring function>, <regular expression substring function>, <fold>, <transcoding>, <character transliteration>, <trim function>, <character overlay function>, <normalize function>, <specific type method>, or <XML **character string serialization>** ~~serialize>~~.

   2) | Insert this GR | If <XML **character string serialization>** ~~serialize>~~ is specified, then let *DC* be the <document or content>, let *XMLV* be the value of the <XML value expression>, ~~and~~ let *DT* be the <data type>**, and let *CS* be the character set of *DT***. The result of <XML **character string serialization>** ~~serialize>~~ is the result determined by applying the General Rules of Subclause 10.14, "Serialization of an XML value", with *DC* as *SYNTAX*, *XMLV* as *VALUE*, ~~and~~ *DT* as *TYPE***, and *CS* as *ENCODING***.

   3) | Replace GR 11) | The result of <blob value function> is the result of the simply contained <blob substring function>, <blob trim function>, <blob overlay function>, or <XML binary string serialization>.

   4) | Insert this GR | If <XML binary string serialization> is specified, then let *DC* be the <document or content>, let *XMLV* be the value of the <XML value expression>, let *DT* be the <data type>, and let *XEN* be the <XML encoding name>. The result of <XML binary string serialization> is the result determined by applying the General Rules of Subclause 10.14, "Serialization of an XML value", with *DC* as *SYNTAX*, *XMLV* as *VALUE*, *DT* as *TYPE*, and *XEN* as *ENCODING*.

4. MODIFY THE CONFORMANCE RULES AS SHOWN HERE:

   1) **| Insert this CR |** Without Feature X070, "XMLSerialize: **Character string serialization and** CONTENT option", ~~in~~ conforming SQL language~~,~~ **shall not contain an** <XML **character string serialization>** ~~serialize> shall not~~ **that** immediately **contains** ~~contain~~ a <document or content> that is CONTENT.

   2) **| Insert this CR |** Without Feature X071, "XMLSerialize: **Character string serialization and** DOCUMENT option", ~~in~~ conforming SQL language~~,~~ **shall not contain an** <XML **character string serialization>** ~~serialize> shall not~~ **that** immediately **contains** ~~contain~~ a <document or content> that is DOCUMENT.

   3) | Insert this CR | Without Feature X072, "XMLSerialize: BLOB serialization and CONTENT option", conforming SQL language shall not contain an <XML binary string serialization> that immediately contains a <document or content> that is CONTENT.

   4) | Insert this CR | Without Feature X073, "XMLSerialize: BLOB serialization and DOCUMENT option", conforming SQL language shall

not contain an <XML binary string serialization> that immediately contains a <document or content> that is DOCUMENT.

3.3 Changes to Subclause 6.13, <XML parse>.

1. MODIFY THE FUNCTION DESCRIPTION AS SHOWN HERE:

   Perform a non-validating parse of a ~~character~~ string to produce an XML value.

2. DELETE SYNTAX RULE 1):

   ~~1) The declared type of <string value expression> shall be a character type.~~

   [NOTE to the proposal reader: The semantics of this SR is now captured by CRs.]

3. MODIFY THE CONFORMANCE RULES AS SHOWN HERE:

   1) Without Feature X060, "XMLParse: **Character string input and** CONTENT option", in conforming SQL language, **the declared type of the <string value expression> immediately contained in <XML parse> shall not be a character string type and** <XML parse> shall not immediately contain a <document or content> that is CONTENT.

   2) Without Feature X061, "XMLParse: **Character string input and** DOCUMENT option", in conforming SQL language, **the declared type of the <string value expression> immediately contained in <XML parse> shall not be a character string type and** <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

   3) Without Feature X062, "XMLParse: explicit WHITESPACE option", in conforming SQL language, <XML parse> shall not contain <XML whitespace option>.

   4) Without Feature X065, "XMLParse: BLOB input and CONTENT option", in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a binary string type and <XML parse> shall not immediately contain a <document or content> that is CONTENT.

   5) Without Feature X066, "XMLParse: BLOB input and DOCUMENT option", in conforming SQL language, the declared type of the <string value expression> immediately contained in <XML parse> shall not be a binary string type and <XML parse> shall not immediately contain a <document or content> that is DOCUMENT.

3.4 Changes to Subclause 10.14, Serialization of an XML value

1. MODIFY GENERAL RULE 1) AS SHOWN HERE:

    1) Let *V* be the XML value *VALUE* in an application of this Subclause. Let *DC* be the *SYNTAX* in an application of this Subclause (*i.e.*, either DOCUMENT or CONTENT). Let *DT* be the ~~character~~ string ~~data~~ type *TYPE* in an application of this Subclause. Let *CS* be the ***ENCODING* in an application of this Subclause.** ~~character set of *DT*.~~

2. MODIFY GENERAL RULE 2) AS SHOWN HERE:

    2) Case:

        a) If *V* is the null value, then the result is the null value.

        b) Otherwise:

            i) If *DT* is a character string type, then:

                ~~i~~**1)** Let *RII* be the XML root information item of *V*.

                ~~ii~~**2)** Case:

                    [...]

                    [Note to the Editor: Please renumber subsequent rules accordingly.]

            ii) Otherwise (*DT* is a binary string type):

                1) Let *RII* be the XML root information item of *V*.

                2) Case:

                    A) If *DC* is DOCUMENT, then:

                        I) If the [children] property of *RII* does not contain exactly one XML element information item, then an exception condition is raised: *data exception — not an XML document*.

                        II) Case:

                            1) If *CS* is UTF8 or UTF16, then let *CEC* be the zero-length string.

                            2) If *CS* is UTF32, then let *CEC* be

                            `encoding="UTF-32"`

                            3) Otherwise, let *EN* be the implementation-defined name for *CS*. Let *CEC* be

`encoding="`*`EN`*`"`

III) Case:

1) **If the [standalone] property of *RII* is "yes", then let *SA* be a binary string equivalent to**

   `standalone="yes"`

2) **If the [standalone] property of *RII* is "no", then let *SA* be**

   `standalone="no"`

3) **Otherwise, let *SA* be the zero-length string.**

IV) Let *VP* be the [version] property of *RII*.

Case:

1) **If *VP* is not "no value", then let *VA* be**

   `version="`*`VP`*`"`

2) **If either *CEC* or *SA* is not the zero-length string, then let *VA* be**

   `version="1.0"`

3) **Otherwise, let *VA* be the zero-length string.**

V) If *VA* is not the zero-length string, then let *CV1* be a binary string equivalent to

`<?xml `*`VA CEC SA`*` ?>`

B) Otherwise, let *CV1* be the zero-length string.

3) Case:

A) If *CS* is UTF16, then let *BOM* be a binary string equivalent to `U&'\FEFF'`.

B) Otherwise, it is implementation defined whether *BOM* is a binary string equivalent to `U&'\FEFF'` or the zero-length string.

Note x - The purpose of the previous rule is to ensure that a Byte Order Mark (BOM) can be prepended to the serialized value when needed due to the specified encoding.

[Note to the Editor: please assign the appropriate value to x.]

4) Let *CV2* be an implementation-dependent binary string such that the result of

`XMLPARSE (DC (BOM || CV1 || CV2) PRESERVE WHITESPACE)`

is identical to *V*, except possibly for the [version] and [standalone] properties of the XML root information item. If there is no such value *CV2*, then an exception condition is raised: *data exception — character not in repertoire*.

5) The serialization of *V* is the result of the assignment of

`BOM || CV1 || CV2`

to a target of type *DT* according to the rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075-2.

3.5 Changes to Subclause 10.15, Parsing a character string as an XML value.

1. MODIFY THE TITLE OF THIS SUBCLAUSE AS SHOWN HERE:

   Parsing a ~~character~~ string as an XML value

2. MODIFY THE FUNCTION DESCRIPTION AS SHOWN HERE:

   Parse a character string **or binary string** to obtain an XML value.

3. MODIFY GENERAL RULES 1) - 3) AS SHOWN HERE:

   1) Let *DC* be the *SYNTAX* (either DOCUMENT or CONTENT), let *V* be the value of the ~~character~~ string *TEXT*, and let *WO* be the value of the ~~<XML whitespace option>~~ *OPTION* **(either PRESERVE WHITESPACE or STRIP WHITESPACE)** in an application of this Subclause.

   2) A ~~character~~ string is called a *textual XML document* if the ~~character~~ string conforms to the definition of a well-formed XML document as defined in [XML 1.0], as modified by [Namespaces].

   3) A ~~character~~ string is called a *textual XML content* if the following are all true:

      a) The ~~character~~ string matches the following production as an extension to the grammar of [XML 1.0], as modified by [Namespaces]:

         `XMLvalue ::= XMLDecl? content`

      b) The character string meets all the well-formedness constraints of applicable productions of [XML 1.0], as modified by [Namespaces].

      c) Each of the parsed entities, as defined by [XML 1.0], that is referenced directly or indirectly within the textual XML content is well-formed, as defined by [XML 1.0].

3.6 Changes to Subclause 24.1, Claims of conformance to SQL/XML.

1. MODIFY LIST ENTRY 5) AS SHOWN HERE:

5) Claim conformance to at least one of the following:

a) Feature X070, "XMLSerialize: **Character string serialization and** CONTENT option".

b) Feature X071, "XMLSerialize: **Character string serialization and** DOCUMENT option".

c) [...]

3.7 Changes to Subclause 24.3, Implied feature relationships of SQL/XML.

1. MODIFY TABLE 14 "IMPLIED FEATURE RELATIONSHIPS OF SQL/XML" AS SHOWN HERE:

Feature ID	**Feature Name**	**Implied Feature ID**	**Implied Feature Name**
...			
X060	XMLParse: **Character string input and** CONTENT option	X010	XML type
X061	XMLParse: **Character string input and** DOCUMENT option	X010	XML type
X065	**XMLParse: BLOB input and CONTENT option**	**X010**	**XML type**
X066	**XMLParse: BLOB input and DOCUMENT option**	**X010**	**XML type**
X070	XMLSerialize: **Character string serialization and** CONTENT option	X010	XML type
X071	XMLSerialize: **Character string serialization and** DOCUMENT option	X010	XML type
X072	**XMLSerialize: BLOB serialization and CONTENT option**	**X010**	**XML type**
X073	**XMLSerialize: BLOB serialization and DOCUMENT option**	**X010**	**XML type**
...			

3.8 Changes to Annex A, SQL Conformance Summary.

1. AUTOMATICALLY GENERATE THE CHANGES RESULTING FROM THIS PAPER FOR ANNEX A.

3.9 Changes to Annex B, Implementation-defined elements.

1. ADD A NEW LIST ENTRY AS SHOWN HERE:

6.1)Subclause 6.5, <string value function>

a) The <XML encoding name>s supported for <XML binary string serialization> are implementation-defined.

b) If an <XML encoding specification> is not specified, then the implicit <XML encoding name> is implementation-defined.

2. MODIFY LIST ENTRY 21) AS SHOWN HERE:

21)Subclause 10.14, "Serialization of an XML value":

a) The encoding name for any character set other than UTF8**,** ~~or~~ UTF16**, or UTF32** is implementation-defined.

[NOTE to the proposal reader: This is a bug fix in passing.]

b) The encoding name for any encoding other than UTF8, UTF16, or UTF32 is implementation-defined.

3.10 Changes to Annex C, Implementation-dependent elements.

1. MODIFY LIST ENTRY 5) AS SHOWN HERE:

5) Subclause 10.14, "Serialization of an XML value":

a) The result is implementation-dependent, but shall be such that reparsing with XMLPARSE with the same choice of DOCUMENT or CONTENT specified in <XML **character string serialization>** ~~serialize>~~ **or <XML binary string serialization>, respectively**, and with the PRESERVE WHITESPACE option, will yield a value identical to the original value.

3.11 Changes to Annex E, SQL feature taxonomy.

1. MODIFY TABLE 15 - "FEATURE TAXONOMY FOR OPTIONAL FEATURES" AS SHOWN HERE:

	Feature ID	Feature Name
...		
32	X060	XMLParse: **Character string input and** CONTENT option
33	X061	XMLParse: **Character string input and** DOCUMENT option
...		
a	**X065**	**XMLParse: BLOB input and CONTENT option**
b	**X066**	**XMLParse: BLOB input and DOCUMENT option**
35	X070	XMLSerialize: **Character string serialization and** CONTENT option
36	X071	XMLSerialize: **Character string serialization and** DOCUMENT option
c	**X072**	**XMLSerialize: BLOB serialization and CONTENT option**
d	**X073**	**XMLSerialize: BLOB serialization and DOCUMENT option**
...		

[Note to the Editor: please assign appropriate values to a, b, c and d.]

4. Checklist

Concepts	n/a
Access Rules	n/a
Conformance Rules	yes
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	no
Any new Possible Problems clearly identified	yes, see Section 1.3, “Issues not addressed”, on page 3 (but this will be addressed in a different paper).
Reserved and non-reserved keywords	no
SQLSTATE tables and Ada package	no
Information and Definition Schemas	no
Implementation-defined and –dependent Annexes	yes
Incompatibilities Annex	no
Embedded SQL and host language implications	no (to be dealt with in a different paper)
Dynamic SQL issues: including descriptor areas	no
CLI issues	no
MED issues	no
SQL/XML issues	yes

- End of paper -