Faster XML data validation in a programming language with XML datatypes

Kurt Svensson
Inobiz AB
Kornhamnstorg 61, 103 12 Stockholm, Sweden
kurt.svensson@inobiz.se

Abstract

EDI-C is a programming language used in B2B (EDI, WebServices) applications. It is possible to declare XML variables in the language. XML variables are declared in a way similar to structures in C and C++. These variables are accessed through dot-notation. Validation information (facets) can also be declared. The XML structures and their validation values are, at compile time, optimized for fast evaluation and execution in the runtime environment. Usually XSD schema are read and evaluated when an XML document loads – our method loads and evaluates the XSD schema at compile time. Thus, when executing a program, the XSD schema has already been loaded and evaluated – i.e., it's no longer necessary to load and evaluate each time the program is executed. This method increases XML processing speed, which is vital for WebServices messaging.

Keywords

B2B, WebServices, format conversion, XML, XML data declarations, XSD schema, XSD schema facets, data validation.

1 Introduction

EDI-C is a special purpose language developed for programming data format conversions in B2B (EDI, WebServices) applications. A program written in EDI-C is compiled into a compact binary file containing a statement tree and various structures such as symbol table, string areas, DOM trees etc. DOM trees can represent XML, EDI (X12, EDIFACT) structures and transaction files. The compiled binary file is interpreted by the EDI-C Runtime virtual machine.

The design goal was to make the language suitable for programming data format conversions – no more, no less.

1) The syntax has to be as close to C and C++ as possible to make it easy for developers to learn the language.

2) Dynamic type conversion between the basic data types: integer and string.

3) No function prototypes have to be declared. Function calls are done with "call by value" parameter passing. Recursive function calls are not allowed.

4) Ability to declare variables that describe the structures of XML, X12, EDIFACT and transaction files. These variables must be directly addressable – no function calls.

5) Embedded SQL as defined in the standard SQL/92 [1].

6) Possibility to call COM objects, .NET components, EXE files, DLLs and other EDI-C programs

The complete language is described in [2].

2 Overview of the programming language EDI-C

The syntax for EDI-C follows the general rules applicable for C and C++.

```
main ()
{
print (“Hello world”);
}
```

The basic data types are: string and integer. Automatic type conversion is done when these data types are used in expressions and function calls.

```
integer i = ”0075” + 25;
// i now contains 100
```

The data type string is similar to the string type in Java and C#. The & character is used as the concatenation operator instead of +.

```
string Text = ”Today is ” & date ();
```

Statements are `if ... else`, `switch`, `for`, `while`, `do ... while`, `break`, `continue` and `return`. The `goto` statement is omitted since it is considered harmful.

Approximately 400 standard built-in functions are available. Many of them are quite “high-level”.

```
// Read a file and convert its data from EBCDIC to ASCII
string s = convertm (readfile (“File.txt”), 2, 0);
```

The language supports embedded SQL and the SQL engine in the EDI-C Runtime uses ODBC, which means that all database systems with an ODBC driver can be used.

```
EXEC SQL SELECT seller, buyer
  INTO :sSellersId, :sBuyersId
  FROM orderhead
  WHERE orderno = :sMyOrderNumber;
```

COM components are accessible.

```
object MyObj;
MyObj = createobject (”xTradeSystem.xtServer”);
MyObj.CreateParse (sender, receiver);
string s = MyObj.mRefIds;
MyObj.RemoteName = computername ();
++MyObj.Counter;
```

3 Declarations of XML variables

Assume we have a (very simple) XML file such as:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Order>
  <Header>
```

```
 <Buyer>The Shoestore</Buyer>
    <Seller>ACME Inc</Seller>
  </Header>
  <Article ArticleNumber="123456">
    <Type>Walking Shoe</Type>
    <Color>Black</Color>
    <Size>7.5</Size>
    <Quantity>12</Quantity>
  </Article>
  <Article ArticleNumber="234567">
    <Type>Golf Shoe</Type>
    <Color>Brown</Color>
    <Size>8</Size>
    <Quantity>25</Quantity>
  </Article>
</Order>
```

The XML file above represents an order for 12 pairs of black walking shoes and 25 pairs of brown golf shoes.

The structure of this XML file can be described in an EDI-C program as:

```
xmlpcdata pcdata;
xmlelem Buyer { pcdata; };
xmlelem Seller { pcdata; };
xmlelem Header { Buyer; Seller; };
xmlelem Type { pcdata; };
xmlelem Color { pcdata; };
xmlelem Size { pcdata; };
xmlelem Quantity { pcdata; };
xmlelem Article
  attribute (name="ArticleNumber" use="required")
  {
  Type;
  Color;
  Size;
  Quantity;
  };
xmldoc Order
  {
  Header;
  Article [9999];
  };
```

If an element or attribute name contains special characters, then the name should be placed within double quotes.

```
xmlelem "Säljare" { pcdata; }; // Swedish for seller
```

Namespace can be specified.

```
xmlelem dt:Seller { pcdata; };
```

So-called virtual namespaces are used for elements with the same name but different content. Recursive definitions also use virtual namespaces, which are specified as integers.

```
xmlelem dt:A {cac:B; cac:C; cs:D; };
xmlelem 1:dt:A {cac:B; cs:D; cac:E; };
```

Groups (sequence, choice, all) are declared as:

```
xmlelem SomeGroup type="choicegroup" { Abc; Def; };
```

In real B2B applications, XML structures are far more complicated than our example. By using the Inobiz Development System, the developer can import XSD schema files. The Development System can then automatically generate the necessary XML declarations. However, it is not the scope of this text to describe the Development System [3].

4 Addressing XML variables

Addressing the various attributes and data fields is done through dot-notation.

```
Order.Header.Seller.pcdata = "ACME Inc";
```

If any of the element names contain special characters, put the name within double quotes.

```
Order.Header."Säljare".pcdata = "ACME Inc";
```

Indexing is relative 1, not 0.

```
Order.Article[2].#ArticleNumber = "234567";
Order.Article[2].Quantity.pcdata = 25;
```

Note that attributes and pcdata have the `string` data type.

A variable of the type `xmlptr` can be set to point to an element node in the XML tree.

```
// Set p to point to first Article
xmlptr p = Order.Article[1];

// Same effect as above
xmlptr p = Order.Article;
```

Loop trough all articles and insert the data into an SQL database.

```
for (xmlptr p = Order.Article; xmldata (p); ++p)
  {
  EXEC SQL INSERT INTO ArticleTable (artnum, type, color, qty)
    VALUES(:p.#ArticleNumber, :p.Type.pcdata,
      :p.Color.pcdata, p.Quantity.pcdata);
  }
```

5 Reading and writing XML files

An XML file is read and parsed by calling the built-in `xmlread` function.

```
xmlread (Order, "SomeFile.xml");
```

If the XML document is in a string, not in a file, an extra parameter to the function is needed.

```
string s = readfile ("SomeFile.xml");
xmlread (Order, s, "m");
```

The XML syntax is checked when parsing the document. The XML document must exactly match the declared XML structure. Any parsing errors are written to the current error log file as:

```
ERROR: 'SomeFile.xml' (7,12) Undefined attribute 'ArticleNum'
  in element 'Article' near
  'ArticleNum="123456">\r\n   <Type>Walking Shoe</'
```

XML files are written using the `xmlwrite` function.

```
// Write the XML document. Append CR/LF at the end of each element and
// indent elements for better readability
xmlwrite (Order, "SoutOutputFile.xml", "n", "2");
```

6 The xmlfacets variable

Pcdata and attributes in elements contain data. In the current implementation, the data have the string type and the length of the data could be between 0 and 40000. In XSD schemas it is possible to restrict the properties of element and attribute data [4] [5]. An EDI-C variable of the type `xmlfacets` is used to define these restrictions.

The syntax of the `xmlfacets` variable declaration is:

```
xmlfacets name
  type="value"
  minLength="value"
  maxLength="value"
  minExclusive="value"
  maxExclusive="value"
  minInclusive="value"
  maxInclusive="value"
  totalDigits="value"
  fractionDigits="value"
  pattern="regular-expression"
  enumeration="value", "value", "value" . . . ;
```

At least one of the facets attributes must be present.

The type attribute value can be any of the following:

```
string, normalizedString, token, byte, unsignedByte, base64Binary, hexBinary,
integer, positiveInteger, negativeInteger, nonNegativeInteger,
nonPositiveInteger, int, unsignedInt, long, unsignedLong, short,
unsignedShort, decimal, float, double, boolean, time, dateTime, duration,
date, gMonth, gYear, gYearMonth, gDay, gMonthDay, Name, QName, NCName,
anyURI, language, ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN,
NMTOKENS, anyType, anySimpleType
```

Value restrictions can be defined for the various elements for the XML Order document example. Restrictions can be added to the XML structure by declaring `xmlfacets` variables. The `xmlfacets` variables are referred to in the pcdata/attribute specifications. The complete declaration of the XML Order document may now be:

```
xmlfacets NameProperties
  type="string" // Default
  minLength="3" maxLength="35";
```

```
xmlfacets ArtNumRestriction
  type=”integer” minInclusive=”100000”
  maxInclusive=”999999”;
xmlfacets TypeRestriction
  minLength=”4” maxLength=”20”;
xmlfacets ColorRestr
  pattern=”Black|Brown|White”;
xmlfacets PossibleSizes
  type=”decimal”
  enumeration=”6.5”, “7”, “7.5”, “8”, “8.5”;
xmlfacets Quantites
  type=”integer” minExclusive=”0” maxExcluive=”1000”;
xmlpcdata pcdata;
xmlelem Buyer { pcdata facets=”NameProperties”; };
xmlelem Seller { pcdata facets=”NameProperties”; };
xmlelem Header { Buyer; Seller; };
xmlelem Type { pcdata facets=”TypeRestriction”; };
xmlelem Color { pcdata facets=”ColorRestr”; };
xmlelem Size { pcdata facets=”PossibleSizes”; };
xmlelem Quantity { pcdata facets=”Quantities”; };
xmlelem Article attribute (name=”ArticleNumber”
  facets=”ArtNumRestriction” use=”required”)
  {
  Type;
  Color;
  Size;
  Quantity;
  };
xmldoc Order
  {
  Header;
  Article [9999];
  };
```

When the program is compiled, all of the facets values, including the enumeration constants, will be inserted into the compilation unit.

7 Facets validation

Facet validation can be made in three ways:

a) When an XML document is read and parsed.

```
// Parameter “v” == validate
xmlread (Order, “SomeFile.xml”, “v”);
```

b) When an XML document is written.

```
// Parameter “v” == validate
xmlwrite (Order, “SomeOutputFile.xml”, “v”);
```

c) Anywhere in a program.

```
// Example 1. Validate the whole XML tree
xmlvalidate (Order);
```

```
// Example 2. Validate each article prior
// to inserting data into a database
string ValErrors = "";

for (xmlptr p = Order.Article; xmldata (p); ++p)
  {
  if (xmlvalidate (p) == true)
    EXEC SQL INSERT INTO ArticleTable (artnum, type, color, qty)
      VALUES (:p.#ArticleNumber, :p.Type.pcdata, :p.Color.pcdata,
       :p.Quantity.pcdata);
  else
    // Pick up errors for later processing
    ValErrors &= xmlvalidateerrors ();
  }
```

A validation error message could be written as:

```
'Article[47].Color.pcdata' contains 'Light-brown'. Does not match 'pattern'
value 'Black|Brown|White'
```

8 Conclusion

A special DOM tree will be constructed at compile time according to the XML declarations. This tree is optimized in such a way that the EDI-C Runtime XML reader can parse and validate XML documents at high speed. The facets and their values are, at compile time, stored in a data structure which will be accessed by the validation mechanism at runtime. Calling the validation mechanism is done from within the XML parser, just as the data is stored in a DOM tree leaf (pcdata/attribute). The same method is used when writing an XML file. The EDI-C XML reader is fast; it parses XML at a speed of 6-7 MB/second on a 3 GHz Windows-PC. If "normal" facets validation is used, the speed will be slowed down to approximately 3-5 MB/second. The XML writer process data at a speed of about 7 MB/second. Writing with validation will slow down the speed.

While speed is perhaps not a crucial factor for ordinary EDI-applications, it is important when using WebServices as a messaging method. Tests have shown that our approach of reading/writing and especially validating has significantly increased the processing speed of WebService applications.

References

[1] Jim Melton, Alkan R. Simon, *Understanding the new SQL*, Morgan Kaufman, 1993
[2] Inobiz AB, *The EDI-C Programming Language, Reference Manual*, 2002-2007.
[3] Inobiz AB, *Development System Users Guide*, 2002-2007.
[4] W3C, *XML Schema Part 2: Datatypes Second Edition*, 28 Oct 2004.
[5] Priscilla Walmsley, *Definitive XML Schema*, Prentice Hall, 2002