PhUSE 2008

Paper TU03

XML by eXaMpLe

Raymond Ebben, OCS Consulting, the Netherlands

ABSTRACT
This document is a basic tutorial on the practical use of XML with examples tailored to the pharmaceutical sector.

INTRODUCTION
This tutorial is written to provide a basic understanding of what XML is all about. Supported by practical examples the tutorial will demonstrate how to make use of XML as a SAS programmer. For this tutorial a basic understanding of SAS base is required. Prior knowledge of XML is not required.

1 SHORT INTRODUCTION TO XML

1.1 WHAT IS XML?

XML is an abbreviation for eXtensible Mark-up Language and finds its origin in SGML (Standard Generalized Mark-up Language) which is an ISO Standard meta language in which mark-up languages for documents can be defined.

XML was originally designed as a mark-up language for documents containing structured information. Since then it has become a standard for many applications to exchange data between systems. XML has a strong resemblance to the HTML language.

1.2 WHY USE XML?

As stated above, XML is a standard for many applications to exchange data between systems and of course there are reasons for that. In this paragraph we will name the most important reasons for our sector.

Unicode
XML is based on Unicode. This is a character set with a much broader range than the most commonly used ASCII character set. Next to international character sets such as Latin, Greek and Cyrillic, Unicode also includes mathematical symbols which we, in the pharmaceutical industry, often need to use.

XML is structured.
The structure of an XML file can be defined. This enables the creation of structured files of which both the content and the structure can be easily validated and the integrity of the data can be ensured.

XML is self describing data storage.
When defining the structure of an XML file almost any type of data can be stored in such a way that XML files are not only machine readable, but humans can also understand the information stored inside XML files.

2 XML STRUCTURE

The most basic building blocks of an XML file are elements, attributes and comments.

Elements

Elements define sections of information within an XML file and are defined using so called “XML tags”. XML tags consist of an opening tag and a closing tag which together mark the borders of the information section. An opening tag represents the name of the element enclosed by “<” and “>” and a closing tag would contain the same name enclosed by “</” and “>”, where the “/” indicates that this is a closing tag. Below is an example of an element called “PatientHeight” with the information representing the patient height, 181, enclosed in the “PatientHeight” XML tags.

eXaMpLe 1 : PatientHeight element

```
<PatientHeight> 181 </PatientHeight>
```

If for some reason the element contains no information, the element will be described in a single tag which represents a merged start and end tag as the “/” is now moved to the end of the element as shown in the example below.

eXaMpLe 2 : PatientHeight element containing no information

```
<PatientHeight/>
```

Elements within an XML document can be nested. This allows the information to be bundled. The following example shows the nesting of patient information such as height and weight nested within the “PatientInformation” element. This nesting then indicates that the height and weight belong to the same patient.

eXaMpLe 3 : Nesting of elements

```
<PatientInformation>
  <PatientHeight> 181 </PatientHeight>
  <PatientWeight> 78 </PatientWeight>
</PatientInformation>
```

Attributes

When we look at the element above, the number 181 gives a good indication of the unit of measurement. But what if the Patients' height was 71.2? This could mean either 71.2 inch or 71.2 cm. We would have to guess. This is where attributes can be applied. Attributes allow for additional descriptive information to be added to the element tag. So given the example above the attribute for "PatientHeight" could be "Unit". The attribute is placed after the name of the element but within the start tag. The syntax for an attribute is the name of the attribute, in this case "Unit" followed by an equal sign and the value of the attribute in quotes. Please note that the use of an element "<Unit> Inch </Unit>" could be used as an alternative to an attribute, but that would beat the purpose of this example.

eXaMpLe 4 : Attribute Unit

```
<PatientHeight Unit="Inch"> 71.2 </PatientHeight>
```

If the element contains no information the element will be described in a single tag as shown in example 2. Below is an example of an empty element with an attribute.

eXaMpLe 5 : Attribute on an element containing no information

```
<PatientHeight Unit="Inch"/>
```

Comments

As described in paragraph 1.2 "Why use XML?", one of the advantages of XML is that it is self describing data. It is not only machine readable but also human readable. Therefore it can be of added value to place comments within XML documents. These comments can provide additional information for a programmer, but is ignored by programs that process XML files (XML parsers). The syntax of a comment also consists of a start and an end tag. The start tag is defined as "<!--"; the end tag is defined as "-->". The following example shows the comment "This is an example comment".

eXaMpLe 6 : Comments

```
<!-- This is an example comment -->
```

2.1 THE DEFAULT STRUCTURE OF AN XML DOCUMENT

As described in paragraph 1.2 “Why use XML?”, XML is a structured language. The default structure of any XML file is defined in the XML specification for creating XML files. These specifications are defined and maintained by W3C (http://www.w3.org/XML/) which is an international consortium responsible for developing protocols and guidelines to define web standards.

XML files that adhere to this standard are referred to as “a well-formed XML”. To verify that the XML file is a “well-formed XML” simply open the XML file with a computer program capable of verifying XML files, such as Internet Explorer (version 5 and higher).

Below, the most basic rules are mentioned. For the complete list see the website of the W3C.

- The XML file must contain a single “root” element (to nest the content of the XML file).
- Every element must be correctly nested.
- Non-empty elements must have begin and end tags.
- Empty elements are denoted by a single tag ending with a slash (/).
- All attribute values must be enclosed in double quotation marks or single quotation marks.
- The content of the XML file is case sensitive.
- Special characters in the content must be “escaped”.
  For example the special characters <, &, and > are represented as <, &, >. A double quotation mark is represented as " and a single quotation mark is represented as '.

eXaMpLe 7 : Example of a well-formed XML file

```
<?xml version='1.0' encoding='utf-8'?>
<Study_ABC>
  <PatientInformation>
    <PatientID> SUBJ001 </PatientID>
    <PatientHeight Unit="CM"> 181 </PatientHeight>
    <PatientWeight Unit="KG"> 78 </PatientWeight>
    <PatientLabValue> 0.789 </PatientLabValue>
  </PatientInformation>

  <PatientInformation>
    <PatientID> SUBJ002 </PatientID>
    <PatientHeight Unit="CM"/>
    <PatientWeight Unit="KG"/>
    <PatientLabValue/>
  </PatientInformation>

  <PatientInformation>
    <PatientID> SUBJ003 </PatientID>
    <PatientHeight Unit="CM"> 173 </PatientHeight>
    <PatientWeight Unit="KG"> 66 </PatientWeight>
    <PatientLabValue> &lt; 0.434 </PatientLabValue>
  </PatientInformation>
</Study_ABC>
```

If you examine the example closely, you may notice the element *<?xml version='1.0' encoding='utf-8'?>* which does not meet the requirements for an element as described above. This is the mandatory element to indicate the file type. The attributes of this element describe the version of the XML specifications to which this document must adhere and the encoding type, which we briefly touched upon in paragraph 1.2 “Why use XML?” where we discussed the benefits of XML.

The element “Study_ABC” is the root element of the XML file. As stated above, each XML document must have a single root element for nesting the information in the file.

2.2 THE CUSTOM STRUCTURE OF AN XML DOCUMENT

Next to the default structure described in the previous paragraph it is possible to define a custom structure for your XML file by creating a separate file in which the structure of the XML file is defined, a so called “definition document”. There are a few flavours of definition documents, but the two most commonly used are DTD and XSD.

DTD stands for Document Type Definition and originates from the same SGML language as XML. This type of structure definition file is the older version and has limitations with respect to describing rules for the data.

XSD stands for XML Schema Definition and is developed more recently than DTD. XSD does not have the limitations of DTD and is recommended by W3C. Therefore the XSD is better suited to describe the custom structure of the XML file. As this is a basic tutorial on XML we will not go into detail on the topic of XSD, but rather touch upon the subject briefly to provide an understanding of what an XSD is and what it is used for.

Within an XSD file the custom structure is described in the form of an XML file. Therefore this file should also adhere to the guidelines for a well-formed XML file. To create an understanding of what type of information can be stored in an XSD file, we will take a closer look at example 7 and see if we can define the contours of a custom structure.

If we look at the structure of the XML file we can see that the element “Study_ABC“ is the root element of the file, which nests the content of the XML file. Within this nesting the element “PatientInformation” occurs multiple times. The element “PatientInformation” nests the elements “PatientID”, “PatientHeight”, “PatientWeight” and “PatientLabValue”. Translated into plain English this means: “Study_ABC has multiple patients and for each patient the ID, height, weight and lab value are stored”.

When examining the individual elements, we can see that the elements “PatientHeight” and “PatientWeight” contain an attribute called “Unit”.

Furthermore we can analyse the type of data in the elements, we can see that the element “PatientID” must be a character value, that the elements “PatientHeight” and “PatientWeight” are (most likely) numeric values, and that the “PatientLabValue” must be character, as we can see the “<” sign in the last value.

So, by examining the XML file in example 7 we have described the structure, the elements, the additional information of the elements in the form of attributes, and we described the type of data to be stored in the different elements. This type of information and more can be described in an XSD. In example 8 on the next page we can see the XSD containing the description of example 7.

eXaMpLe 8 : XSD describing the custom structure of example 7

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Study_ABC">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="PatientInformation">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="PatientID" type="xs:string" />
              <xs:element name="PatientHeight">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:decimal">
                      <xs:attribute name="Unit" type="xs:string" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="PatientWeight">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:decimal">
                      <xs:attribute name="Unit" type="xs:string" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
              <xs:element name="PatientLabValue" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

When we have both the XML file and the XSD file we can use an XML parser to validate the content. This will verify that the XML document is well structured, adheres to the custom structure and has the correct data types. This means that when you store character data into a numeric element, this would result in a validation error.

An example of an application capable of parsing XML and validating XML files according to their XSD is “XML Notepad 2007” which can be downloaded from the Microsoft website.

3 READING AND WRITING XML FILES WITH SAS

Now that we have obtained a basic understanding of an XML file, let's start with reading and writing XML files from SAS. One way of reading and writing XML files with SAS would be to make use of the XML libname engine, which translates SAS datasets to XML and vice versa. This way the XML files can be used as if they were datasets.

3.1 WRITING XML FILES WITH THE SAS XML LIBNAME ENGINE

The syntax of the XML libname statement is `libname libref XML 'file location' <Options>;`
When the SAS XML libname reads XML data, this data must be in a structure suitable for the XML libname engine to read. To understand how SAS expects the structure of such an XML file to be, let's start by writing an XML file with SAS, and examine the XML output file. When we execute the following SAS code:

SAS Code example 1

```
001  data work.study_abc;
002    length patientid $10 patientheight 8 patientweight 8;
003    infile cards;
004    input patientid $ patientheight patientweight;
005  cards;
006  subj001 181 78
007  subj002 . .
008  subj003 173 66
009  ;
010  run;
011
012  libname phuse xml 'c:\temp\study_abc.xml' ;
013
014  data phuse.study_abc;
015    set work.study_abc;
016  run;
```

The result XML looks like this:

Result of example 1

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <STUDY_ABC>
      <patientid> subj001 </patientid>
      <patientheight> 181 </patientheight>
      <patientweight> 78 </patientweight>
   </STUDY_ABC>
   <STUDY_ABC>
      <patientid> subj002 </patientid>
      <patientheight Missing="." />
      <patientweight Missing="." />
   </STUDY_ABC>
   <STUDY_ABC>
      <patientid> subj003 </patientid>
      <patientheight> 173 </patientheight>
      <patientweight> 66 </patientweight>
   </STUDY_ABC>
</TABLE>
```

As you can see SAS names the root element of the XML file "TABLE". The elements defining the observations of the datasets contain the name of the SAS dataset "Study_ABC". The elements nested in the observations are the names of the variables within the dataset. The value for each observation and variable is stored within a variable element. Please note that missing values are presented with the "missing" attribute. To generate just an empty tag the option `tagset=sasxmiss` can be added to the SAS XML libname statement. The example shown uses the SAS datastep to create an XML file. But the SAS XML libname engine can be applied to most SAS procedures.

3.2 READING XML FILES WITH THE SAS XML LIBNAME ENGINE

When reading XML files with the SAS XML libname engine, SAS expects the XML files to be in the following format:

- One root element (as is mandatory on a well formed XML).
- An element bundling the variable information per observation. The name of this element will be used to name the dataset.
- Element(s) within the observation element. These will become the SAS variables.

This was illustrated in the previous example (Results of example 1), where we analysed the XML file written by SAS.

The following code will read in the XML file into a SAS dataset:

SAS Code example 2

```
001 libname phuse xml 'c:\temp\study_abc.xml';
002
003 data work.study_abc;
004   set phuse.study_abc;
005 run;
```

3.3 WRITING XML FILES TO EXCHANGE DATA

Earlier in this tutorial we mentioned that XML is a standard for many applications to exchange data between systems. These systems require the XML file to adhere to a custom structure as defined for the system.

For some of these systems SAS has provided a standard solution which can be applied by using the `XMLTYPE` option. This option specifies the output type of the XML file. This option is added to the SAS XML libname statement and assigned with a type value. The default value is the value GENERIC, which will write a generic XML file as shown in SAS code example 1. Other values include ORACLE, HTML, MSACCESS and CDISCODM.

As you may have noticed, the last value mentioned refers to the CDISC Operational Data Model, which will help you to create XML files adhering to the CDISC ODM standard. Currently version 1.2 of the CDICS ODM is supported. As this is a basic tutorial we will not go into the specifics of using the CDISCODM `XMLTYPE`.

An alternative approach to create an XML file adhering to specific system standards would be to use a specific `TAGSET`, designed to write the XML file in the custom structure required. The `TAGSET` option is added to the SAS XML libname statement and assigned with a `TAGSET` name. An example of an existing `TAGSET` would be “MSOFFICE2K”. This would create an XML file readable by Microsoft Office 2000.

Finally there is an option to determine the type of XML building block to be used. By default SAS will write the data into XML elements, but when the options `XMLDATAFORM` is added to the SAS XML libname statement with the value attribute specified, SAS will write the values to the XML file using attributes instead of elements. For example the element `<patientid> subj001 </patientid>` from Result of example 1, would then be written as: `<column name="patientid" value="subj001"/>`.

3.4 WRITING XML FILES AND SCHEMAS

In paragraph 2.2 “The custom structure of an XML document”, we discussed that the custom structure of an XML file is described via an XSD. This XSD is referred to by SAS as the XML schema. With the SAS XML libname statement SAS offers the possibility to write the schema and the data. To utilise this functionality the option XMLMETA must be specified. This option can have the values SCHEMA, DATA and SCHEMADATA, where SCHEMA will write just the schema, DATA will write just the XML data, and SCHEMADATA will write both the SCHEMA and the XML data. The following example shows the SAS code to write both the schema and the data. The creation of the work dataset Study_ABC has been omitted in this example, as it is the same in SAS Code example 1.

SAS Code example 3

```
001 libname phuse xml 'c:\temp\study_abc.xml' xmlmeta=schemadata;
002
003 data phuse.study_abc;
004   set work.study_abc;
005 run;
```

The result XML looks like this:

Result of example 3

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:od="urn:schemas-microsoft-com:officedata">
    <xs:element name="TABLE">
      <xs:complexType> <xs:sequence>
        <xs:element ref="STUDY_ABC"  minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence> </xs:complexType>
    </xs:element>
    <xs:element name="STUDY_ABC">
      <xs:complexType> <xs:sequence>
        <xs:element name="patientid" minOccurs="0" od:jetType="text"
         od:sqlSType="nvarchar">
          <xs:simpleType><xs:restriction base="xs:string">
            <xs:maxLength value="10" />
          </xs:restriction></xs:simpleType>
        </xs:element>
        <xs:element name="patientheight" minOccurs="0" od:jetType="double"
         od:sqlSType="double" type="xs:double" />
        <xs:element name="patientweight" minOccurs="0" od:jetType="double"
         od:sqlSType="double" type="xs:double" />
      </xs:sequence> </xs:complexType>
    </xs:element>
  </xs:schema>
   <STUDY_ABC>
      <patientid>subj001</patientid>
      <patientheight>181</patientheight>
      <patientweight>78</patientweight>
   </STUDY_ABC>
   <STUDY_ABC>
      <patientid>subj002</patientid>
      <patientheight/>
      <patientweight/>
   </STUDY_ABC>
   <STUDY_ABC>
      <patientid>subj003</patientid>
      <patientheight>173</patientheight>
      <patientweight>66</patientweight>
   </STUDY_ABC>
</TABLE>
```

Please note that the indention of the result file has been manipulated to fit the example on the page.
By default SAS will write the information into one file; the file referenced in the SAS XML libname statement. There

are situations or applications that require a separate schema file, which I personally prefer as I believe it is better to separate the content from the structure. To achieve this, the option XMLSCHEMA can be added to the SAS XML libname statement. The SAS code would then look like this.

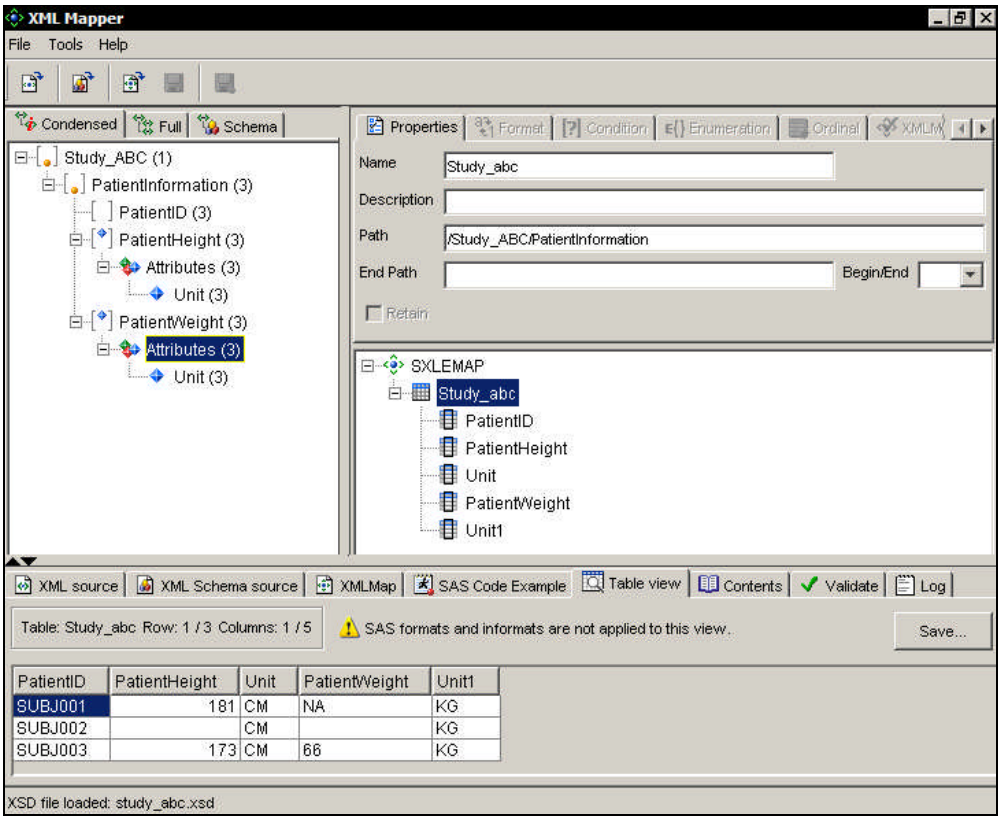
SAS Code example 4

```
001  filename xsd 'c:\temp\study_abc.xsd';
002
003  libname phuse xml 'c:\temp\study_abc.xml'
004       xmlmeta=schemadata
005       xmlschema=xsd;
006
007  data phuse.study_abc;
008   set work.study_abc;
009  run;
```

The output of this would be two separate files: an XML file containing the data and an XSD file containing the schema.

3.5 READING XML FILES THAT ARE NOT IN THE EXPECTED FORMAT

As described in paragraph 3.2 “Reading XML files with the SAS XML libname engine”, SAS expects the XML file to be in a specific format. But what if the XML file you want to read does not adhere to this format? The solution here is to create an “XMLMAP”, which literally maps the XML elements and attributes to the columns of a SAS dataset. To create these XML maps suitable for use with the SAS XML libname engine, SAS provides the tool “SAS XML mapper”, which can be found on the SAS installation discs.



Without discussing the SAS XML mapper in detail, the tool allows the user to open an XML file and/or an XSD file. This will result in a tree structure of the data on the left hand pane. By dragging the elements and attributes to the right hand pane, the SAS dataset is described, as is shown in the bottom result pane. To actually use this mapping in a SAS program, save the mapping to a location on the disk and on the “SAS Code Example” tab in the XML mapper, copy the SAS code and if required modify the location of the XMLMAP.

3.6 USING THE XML LIBNAME ENGINE TO READ MICROSOFT EXCEL DATA

Personally, one of the things I find a nuisance is reading Excel data. There is always the hassle with the variable types, the potential "locking" of the file and the additional SAS license required. Of course there are ways to get around this and this is one of them! Please note that the functionality used is only available for Microsoft Office Professional Edition 2003 and Microsoft Office Excel 2003, or newer.

The method described here is a workaround type of method which will do nicely when an Excel spreadsheet (or different spreadsheets with the same type of content) is read by SAS more than once. As an example let's imagine that the data of the Study_ABC, which has been used in this tutorial, was originally delivered in an Excel spreadsheet.

It all starts by creating a dummy dataset in SAS reflecting the data you want to read from Excel. For the example we are going to create an empty dataset defining the structure of the data we want to read. Please note that it would not matter if the dataset contained any data. The next step is to set up a SAS XML libname statement to write the schema of the dataset to an XSD file. This is illustrated in the example below.

SAS Code example 5

```
001  data work.study_abc;
002    length patientid $10
003           patientheight 8
004           patientweight 8;
005    stop;
006  run;
007
008  libname phuse xml 'c:\temp\study_abc_xls.xsd' xmlmeta=schema;
009
010  data phuse.study_abc;
011    set work.study_abc;
012  run;
```

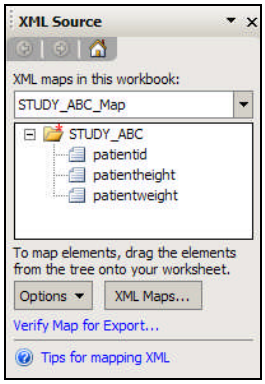
Executing this code will result in an XSD file containing the schema of our target dataset. The next step is to open the source spreadsheet in Excel. The data would look something like shown below.

	A	B	C
1	**PatientID**	**PatientHeight**	**PatientWeight**
2	Subj001	181	78
3	Subj002		
4	Subj003	173	66

The next step is to go to the Data menu and click on "XML". Then click "XML Source" to open the XML Source task pane which will be displayed on the right hand side of the window.

On the XML Source task pane, click on the button "XML Maps…" to open the map selection dialog box. On this dialog box, select "Add.." and browse to the XSD schema created with SAS Code example 5, (c:\temp\study_abc_xls.xsd) and click open.

When prompted to select the root, select the value "TABLE". As seen in the result of example 1, this is the root element of the XML file. The XML source pane will now look something like this:



The next step is to drag the elements “PatientID”, “PatientHeight” and “Patientweight” from the XML source pane, to the corresponding header in the Excel file. When finished, save the Excel spreadsheet for future use.

As a final step to export the data, go to the File menu and click “Save as…”. In the “Save as type:” dropdown select XML Data (*.xml) and choose a filename and location to save the file. When finished we can import this data into SAS by using the SAS XML Libname engine as shown in SAS Code example 2.

CONCLUSION

XML is a very powerful mechanism to store data or to exchange data between systems. With standards like CDISC, and the increasing availability of EDC tools, PDA’s, tablet PC’s et cetera, the necessity for the pharmaceutical SAS programmer to have at least a basic understanding of XML will increase.

ACKNOWLEDGEMENTS

I would like to thank OCS Consulting for providing me with the means to write and present this paper, and in particular my colleague Jules van der Zalm for reviewing my paper.

RECOMMENDED READING

For further information I would recommend the following resources:

http://www.w3.org/XML/
Read this for more background information on XML and XML specifications

http://www.w3schools.com/xml/
Read this for a tutorial on XML

SAS® 9.1.3 XML LIBNAME Engine User guide, by the SAS institute.
This guide provides more detailed information on the SAS XML libname engine and can be obtained from the SAS support website.

CONTACT INFORMATION

Your comments and questions are valued and encouraged, contact the author at:

Author Name	:	Raymond Ebben
Company	:	OCS Consulting
Address	:	P.O. Box 490 5240 AL Rosmalen The Netherlands
Work Phone	:	+31(0)73 523 6000
Facsimile	:	+31(0)73 523 6600
Email	:	sasquestions@ocs-consulting.com
Web	:	www.ocs-consulting.nl