**PhUSE 2009**

**Paper TS01**

# XML management using SAS®

Marco Bertolino, Actelion Pharmaceuticals Ltd, Imperia, Italy

## ABSTRACT
This document is a basic tutorial on the practical use of XML through SAS programs.

## INTRODUCTION
XML (the e**X**tended **M**arkup **L**anguage) is an open standard for the definition, transmission, validation, and interpretation of data. The standard was developed by the Worldwide Web Consortium (W3C) in order to provide a simple and efficient way to manage self-documenting data files. SAS® Software includes a number of useful tools for creating and parsing XML data. These include:

- The *XML libname engine* is used to import and export documents in XML format into or from a SAS dataset, optionally using an XMLMap. The new XML92 engine now supports XML Maps on output as well as input.
- The *XML Mapper* application is a graphical interface to analyze the structure of an XML document or an XML schema and generate XML syntax for the XMLMap.
- The ODS MARKUP output destination is used to create XML from SAS output; ODS MARKUP creates but does not read XML documents.

## 1 SHORT INTRODUCTION TO XML
XML is an abbreviation for eXtensible Mark-up Language and finds its origin in SGML (Standard Generalized Markup 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.

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.

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 XSD (*XML Schema Definition*) and DTD (*Document Type Definition*). A schema (XSD) is a way of self-documenting an XML file. It's like a PROC CONTENTS of XML files. A DTD is much the same, but schemas are newer and will eventually replace DTDs. Creating a schema is much like planning to create a database.

Within an XSD file the custom structure is described in the form of an XML file. 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 or better you can define which elements are mandatory or optional and more you can define the list of acceptable value for each element.

It would certainly be possible to write your own XML parser in a SAS DATA step (the XML libname engine supports DTDs and XSDs on output but not, unfortunately, on input; the engine just assumes that input documents are valid and well-formed), but the general feeling is that given the availability of well tested ones, why bother? XML parsers are available for downloading from Sun, Microsoft and Oracle among other locations; a full list of sources is available at http://www.xml.com/pub/rg/XML_Parsers.

Figure 1 illustrates a possible XML representation of some clinical trial specimen data.
In this case, all clinical trial data are grouped under a single study which is identified by a StudyID. A study may have

multiple sites from which data are collected and each site is identified by a unique SiteID. A site has multiple participants who may visit the site on multiple visits.
Each visit has a date and VisitID and may include the collection of specimens. In turn, each specimen has a type and it is collected in one or more tubes, all of which share the same tube type.

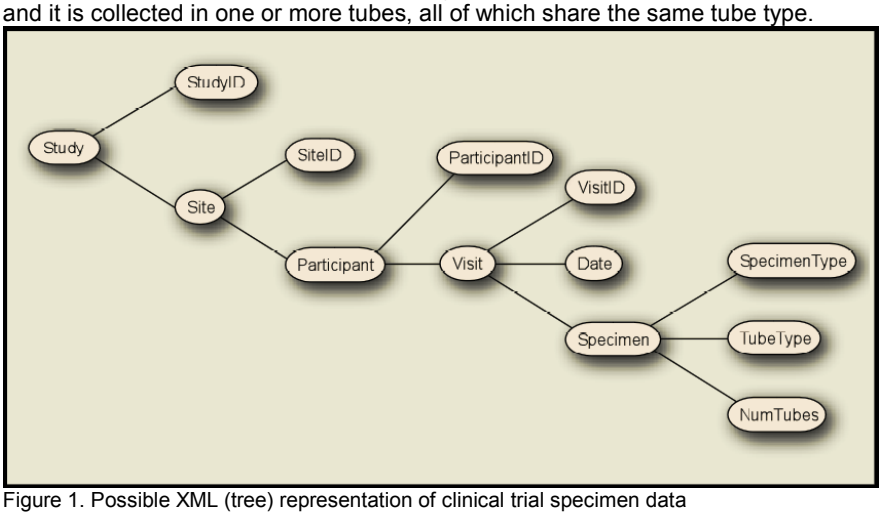



Figure 1. Possible XML (tree) representation of clinical trial specimen data

Figure 2 diagrams a possible relational view of the same data displayed in Figure 1. The relational diagram stores the same information as the hierarchical diagram in Figure 1 preserving both the data and the relationships between the data entities.
In this case, the names of the tables can be extracted by determining the critical path among the nodes of the tree in Figure 1. At each level of the tree, there is a single node that acts as a root for a sub-tree (i.e. Study,Site).

Each one of these critical path nodes becomes a table in the relational diagram while the rest of the nodes become column names within the table named after its parent node. Some of these subordinate nodes become primary keys within their tables if they can uniquely identify a table entry. For instance, the table Site uses the column SiteID as a primary key since it is a unique identifier for all site records.
The primary keys are also reproduced in related tables as foreign keys to establish relationships among tables. For example, SiteID is the primary key to the Site table and it is also a foreign key within the Participant table reflecting the relationship between Site and Participant. Additionally, the links among the tables preserve the cardinality of the relationships.

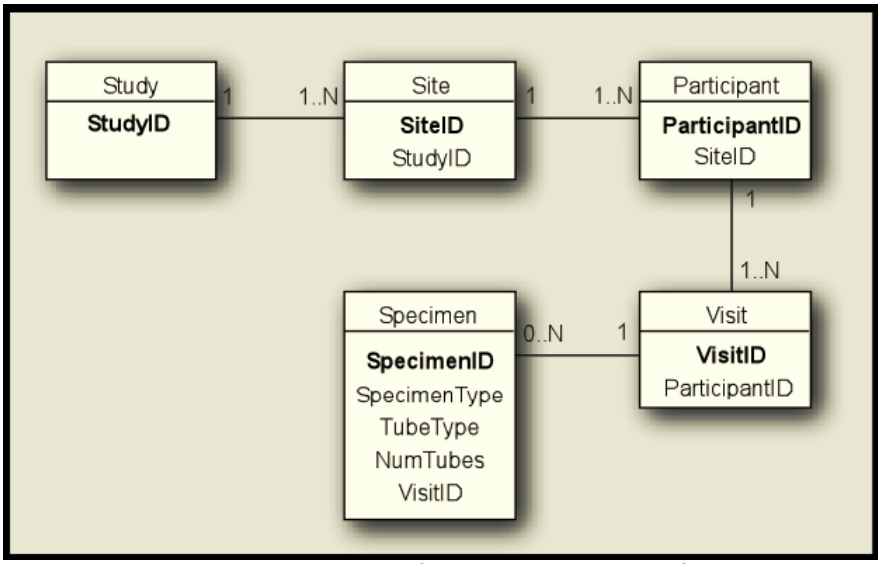



Figure 2. Possible relational representation of clinical trial specimen data from Figure 1

Transferring data from a hierarchical structure (XML) to a relational structure (SAS) is a non-trivial problem. It requires good understanding of the data to be able to identify primary keys, foreign keys, and dataset variables. Data transfer from a relational structure (SAS) to a hierarchical structure (XML) is also difficult. You need to identify which data elements take precedence in the hierarchy and establish an order relationship. In either case, the programmer must have a solid understanding of the source data to be transferred.

### 1.1 BASIC RULES OF XML

While SAS now has tools for you to read in XML documents without your needing to become an XML guru, it is important to familiarize oneself with the following basic rules of XML before attempting to convert XML files to SAS datasets and vice versa:

• The basic building block of XML is an element. Elements begin with an open tag <element_name>
• Elements can either end with an end tag </element_name> or terminate with '/>' in their open tag <element_name/>
• All tags must be fully nested – no overlapping tag boundaries are allowed.
• A tag may optionally include a value.
• A tag may optionally include attributes, which are values enclosed in double quotes.
• A comment is text delimited by '<!--' and '-->'.

## 2 WORKING WITH XML

SAS intersects with markup languages in two ways: through the SAS XML Libname Engine (SXLE) to import from XML into SAS format or to export from SAS data set format to XML format or with ODS MARKUP to create various types of markup files from procedure or data step output. The method that you use to create markup files depends on the disposition of the file. If you are publishing reports and document content, then you will probably use ODS MARKUP to create marked up content, such as HTML or CSV files. However, if you are dealing with XML as part of a data base or data exchange system, and you need to either create markup files from SAS data sets for data exchange or to import XML files into SAS data set format, you will use the SAS XML Libname Engine.

### 2.1 VIEWING AN XML FILE

Since XML documents are text files, they can be viewed in virtually any text editor. In addition, there are XML editors available that simplify the process of writing and editing XML documents, including the SAS XML Mapper tool provided with SAS 9.1 and SAS 9.2.

An easy to be used freeware XML editor is XMLFox ® editor (http://www.xmlfox.com), in Figure 3 and Figure 4 is presented how the same XML file can be displayed on XMLFox editor.

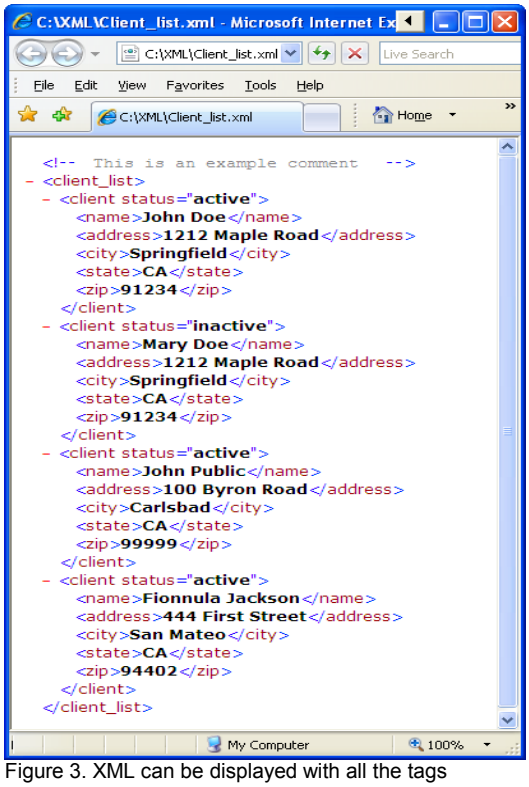


Figure 3. XML can be displayed with all the tags

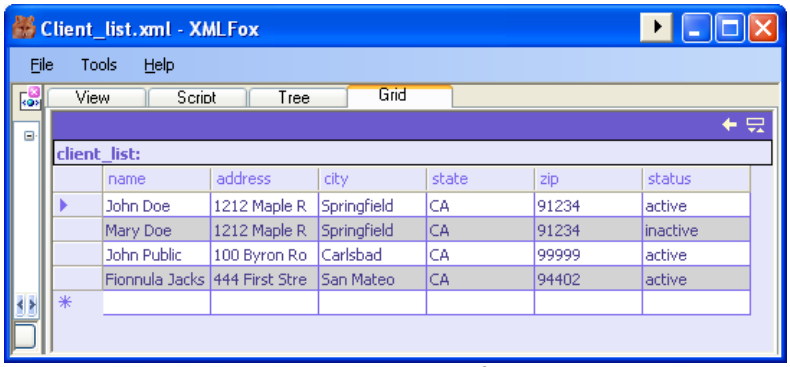


Figure 4. XML can be displayed looking at the data first

## 2.2 READING AN XML DOCUMENT INTO A SAS DATASET

The easy way to access and write XML from SAS is to use the XML engine on a libname statement (versions 8.1 and above). However, using this statement without any other options requires that the XML file structure contain only rectangular datasets (the usage of custom tagsets is not covered by this paper, for a complete treatment of this topic look at http://www.lexjansen.com/pharmasug/2006/technicaltechniques/tt24.pdf). In other words, the hierarchy for the source XML Document can only be three levels deep, containing a root element, second-level elements corresponding to the table to be read in, and third-level elements corresponding to the variables. For this example, we use the XML engine to parse the incoming file called hosp_discharge.xml, which contains an excerpt of a hospital discharge data file.

The syntax of the XML libname is similar to that of a standard SAS libname, but (1) xml is placed after the libname keyword and before the location to indicate that the XML engine is to be used, and (2) the location in quotes refers to the specific XML file, not just a directory path.

Before reading in an XML file into SAS, it is a good idea to examine it. Figure 5 shows a screenshot of the hosp_discharge.xml file in Internet Explorer. The file contains a root-level node called <hosp_discharge> which in turn contains a number of <discharge> elements. Each <discharge> element has 5 children elements <patient_ID>, <SEX>, <admit_date>, <DOB>, and <discharge_date>.

**THE SAS CODE TO READ THE XML FILE IS AS FOLLOWS:**

```
title1 'XML TEST Read';
libname in xml 'C:\XML\hosp_discharge.xml';
DATA discharge;  set in.discharge;
run;
title2 'check parsing of file';
proc print DATA=discharge;
run;
proc contents DATA=discharge;
run;
```

**THE PROC PRINT OF THE RESULTING SAS DATASET IS REPRODUCED BELOW:**

| Obs | DISCHARGE_DATE | DOB | ADMIT_DATE | SEX | PATIENT_ID |
|---|---|---|---|---|---|
| 1 | 2009-09-01 | 1940-12-27 | 2009-08-01 | F | 2121229 |
| 2 | 2009-10-10 | 1946-08-25 | 2009-09-01 | F | 3359900 |
| 3 | 2009-10-10 | 1910-06-30 | 2009-05-01 | M | 4245123 |

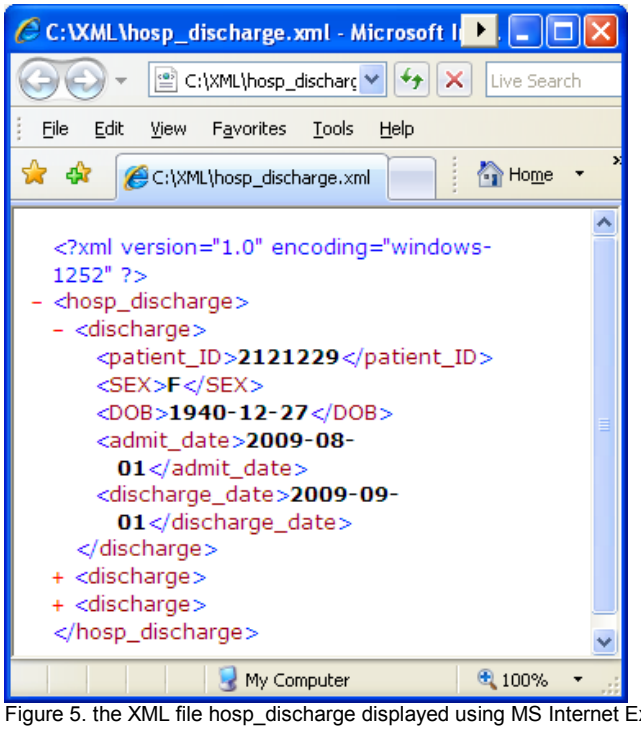


Figure 5. the XML file hosp_discharge displayed using MS Internet Explorer

**AN EXCERPT FROM THE PROC CONTENTS IS BELOW:**

| # | Variable | Type | Len | Format | Informat | Label |
|---|---|---|---|---|---|---|
| 3 | ADMIT_DATE | Num | 8 | IS8601DA. | YYMMDD. | ADMIT_DATE |
| 1 | DISCHARGE_DATE | Num | 8 | IS8601DA. | YYMMDD. | DISCHARGE_DATE |
| 2 | DOB | Num | 8 | IS8601DA. | YYMMDD. | DOB |
| 5 | PATIENT_ID | Num | 8 | F8. | F8. | PATIENT_ID |
| 4 | SEX | Char | 1 | $1. | $1. | SEX |

If we examine the SAS code and the XML file on the previously mentioned we notice that the name of the SAS dataset is the same as the name of the second-level element in the XML file (<discharge>). Additionally, the variable names in the PROC PRINT output match the name of the third-level elements in the XML file. However, the case of the element names is not preserved, as SAS converts all characters in a name to upper case. Finally, the XML engine formats dates using the IS8601DA. (yyyy-mm-dd) format. (Former releases transformed such dates to character fields.)

## 2.3 CONVERTING A SAS DATASET TO XML

To convert a SAS dataset to XML, create an XML libname and then write to it. The following lines of code illustrate how to write out the dataset created in the previous point to an XML file. The program adds in a SAS date variable, xfer_date, to record the date we transferred the file back to XML format.

```
title1 ' XML TEST Write';
libname out xml 'C:\XML\discharge_from_SAS.xml';  xmltype=generic;
DATA out.discharge;
set discharge;
***let's add in the date converted as today's date and format it in ISO8601
***(yyyy-mm-dd) format;
xfer_date=today();
format xfer_date is8601da10.;
run;
```

The resulting XML file as seen through Internet Explorer (partially collapsed) is displayed in Figure 6, below. Notice the following about the resulting XML file in Figure 6:

• The root element is <LIBRARY>. This is a convention SAS 9.2 uses when writing out XML files using the XML libname engine. (former SAS versions was using the root element <TABLE>).
• Tags for all the original fields are all caps (this conversion was done during the original parsing for reading data).
• The tags for the xfer_date field are in lower case, since we created that variable using lower case in SAS.
• All the SAS date fields are displayed in SAS internal numeric data storage (# of days since January 1, 1960). This is because formats are not preserved in general when copying to non-native engines such as XML.

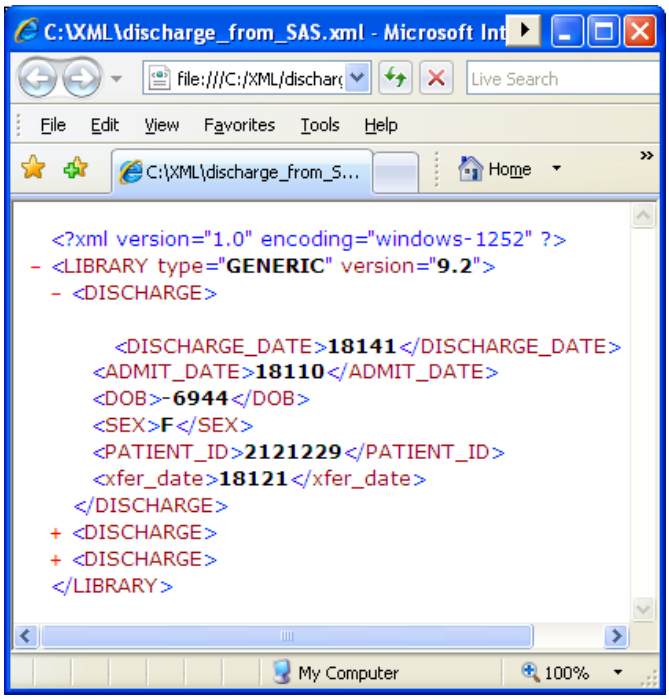



Figure 6

## 2.4 CREATING XML FILES FROM SAS PROCS USING Output Delivery System

SAS Release 8 introduced the `ODS XML` driver as an experimental driver, with no guarantee that the output would be valid XML. There was only one DTD available, which produced a document in a single standard format. With Release 9.1, SAS provided the new `ODS MARKUP` statement, allowing the user to export a variety of markup languages, including HTML, XML, CSV, DTD, CSS and XSL. It is important to note that ODS cannot be used to parse XML, only generate it.

```
ods xml
file='C:\XML\discharge_print.xml';
proc print noobs data=discharge;
run;
ods xml close;
```

```
ods markup
body='C:\xml\discharge_print.xml';
proc print noobs data= discharge;
run;
ods markup close;
```

If you examine the resulting file, discharge_print.xml in IE, you will notice that it follows a very rigid data structure that specifies every detail of the PROC PRINT issued including the title, the name of the data set, the descriptions of every column, and the data of the entire table on a row by row basis. This XML format carries a lot of detail which is unrelated to the data being presented. This makes it impractical as a data transport mechanism because its verbose nature makes the resulting XML files quite large and unwieldy.

# 3 READING IN MORE COMPLICATED XML FILES

As mentioned earlier, using the XML libname to read in XML files to SAS requires that the file be structured in a rectangular (non hierarchical) manner. Specifically, this means that each observation to be read into SAS must be appear within a second-level element, and each variable value must appear as a third-level element.
By default, XML attributes are dropped in the conversion process, and SAS determines the data type, format, and informat of all variables. SAS 9.1/9.2 include a production version of the XMLMAP option, which can read in XML files that don't map directly to rectangular data. The SAS XML Libname engine shipped with SAS 8.2 cannot handle nonrectangular input files.

## 3.1 USING THE XMLMAP OPTION REQUIRES THE CREATION OF A MAP FILE

The map file is an XML file that specifies how SAS is to map the source XML document to specific datasets, variables and observations. Documentation for this option can be found at http://support.sas.com/91doc/docMainpage.jsp. See also Anthony Friebel's paper presented at SUGI 28 which includes an extensive discussion of the XMLMAP engine (in the References section at the end of this paper).
SAS 9.1/9.2 also includes a production version of SAS XML Mapper, a stand-alone application that can be used as an XML Editor or to create XML map files with a drag and drop interface. SAS XML Mapper is located on the 9.1 client component CDs. SAS XML Mapper should also be available via web download (http://www.sas.com/apps/demosdownloads/92_SDL_sysdep.jsp?packageID=000513&jmpflag=N).
The structure of the basic XMLMAP schema is displayed in Figure 7
and described further below:

- The root element of the XMLMAP file is the <SXLEMAP> element. It can have one or more <TABLE> children elements. Each <TABLE> element identifies a SAS dataset by name using the name= attribute.
- A <TABLE> element has a single <TABLE-PATH> child element which specifies the point in the XML document where the table data begins. A <TABLE> element also has one or more <COLUMN> elements that identify the columns of the dataset. A <COLUMN> element will have several children:
  - a <DATATYPE> element that specifies the XML data type of the source data
  - an optional <FORMAT> element that specifies a format to be associated with the variable
  - an optional <INFORMAT> element that specifies an informat to be associated with the variable
  - a <LENGTH> element that specifies the data length
  - a <TYPE> element that indicates the SAS data type (character or numeric) used for the column
  - a <PATH> element that specifies where to look for column data within the XML input file's hierarchy

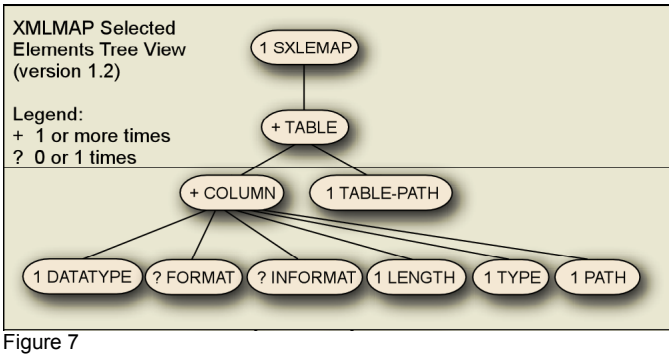
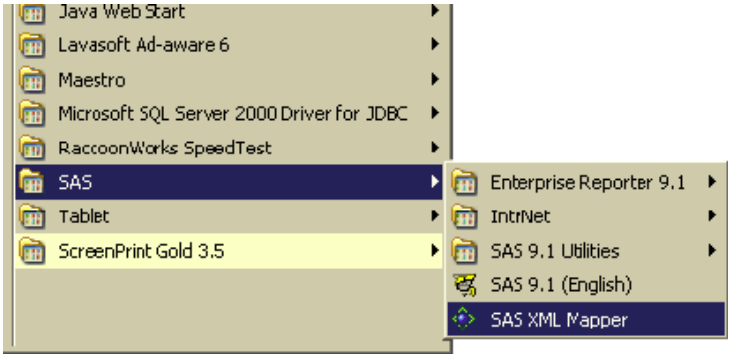


Figure 7

## 3.2 Use the XMLMAP

In this section it will be presented the option to read all client_list.xml elements (viewed in 2.1) and map (1) the status= attribute to a character variable named status and (2) the zipcode element to a character variable.
In general, reading in more complicated XML files into SAS can be accomplished using a five-step process:
1. View the Data/Understand the structure.
2. Decide what you want to extract.
3. Use SAS XML Mapper to create the map file.
4. Use SAS XML Mapper to generate the SAS code to read in the file and reference the map file.
5. Submit your program and check that the conversion worked.

First, we launch SAS XML Mapper by selecting **All Programs** from the **Start** button in Windows and picking the "SAS" Program Group. Within it, click on the **SAS XML Mapper** item to launch the application.




The XML Mapper user interface uses a single window divided in 3 panes (see Figure 8, below). The top-left pane named the **XML pane** and displays the tree structure of an XML file. The top-right pane is named the **XML Map pane** and displays a graphical representation of the table and columns specified for the target SAS dataset(s) and also allows the user to specify the properties and formats of each dataset column. The bottom pane is named the **Source Pane** and includes multiple tabs which in turn display the XML document source code, the source code for the automatically generated XML Map, the automatically generated SAS code used to read the XML file, a Map file validation test pane, and a Log file.

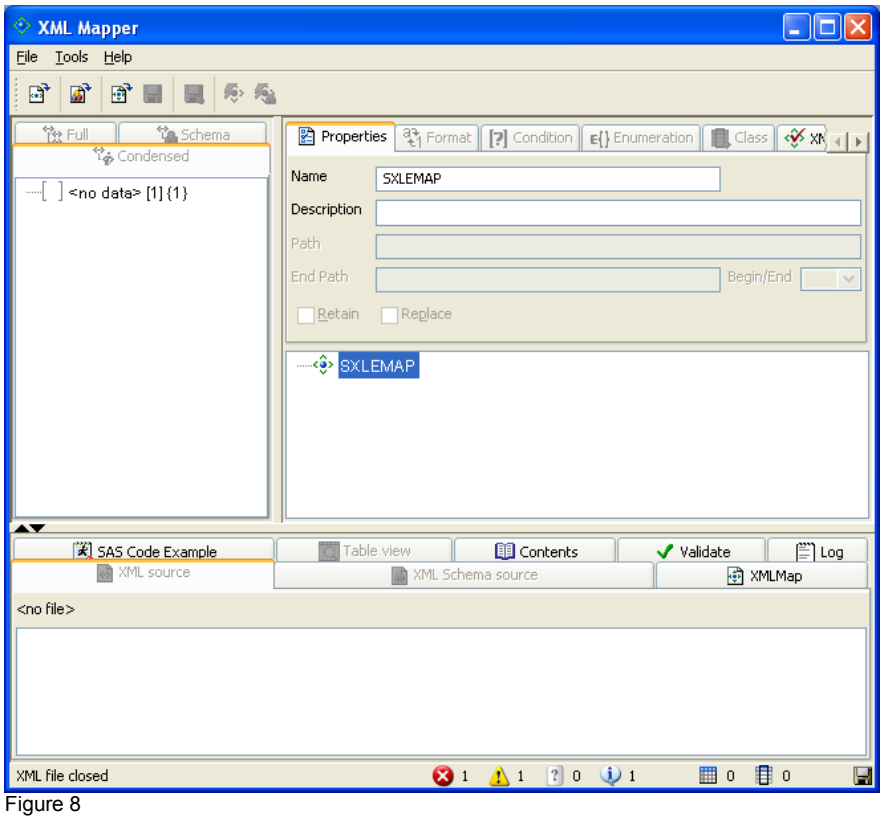



Figure 8

STEP 1: OPEN THE XML TO BE IMPORTED

The normal sequence of actions in XML Mapper involves opening an existing XML document (or the corresponding XSD file), creating an XML Map from the elements of the XML document, saving the resulting XML Map file, and saving the automatically generated SAS code that reads the XML document.

To open an XML document simply select the "**Open XML…**" menu item from the **File** menu, then select the name of the file to open by navigating through the directory structure. If we open the document client_list.xml, the XML Mapper interface appears as in Figure 9:

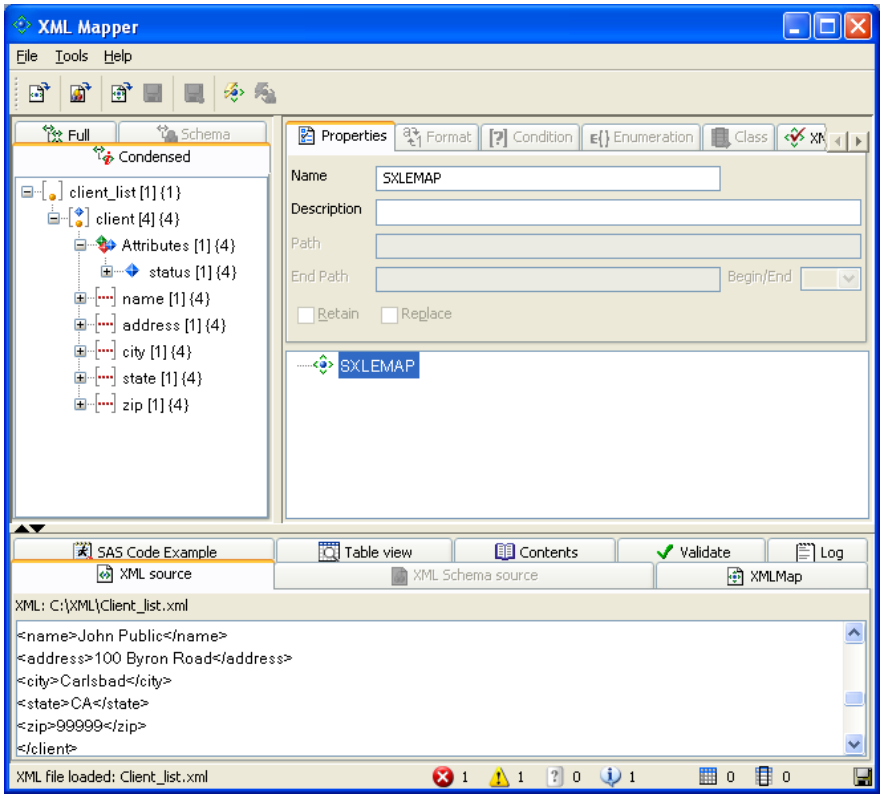



Figure 9

STEP 2: DECIDE WHAT YOU WANT TO EXTRACT

We want to extract all the client information from the XML document. Specifically, we want the name, address, city, state, zip sub-elements of the client element as well as the active attribute from the client element. This means that we will use the client element of the XML document as the basis for our target dataset, and its sub-elements and attribute will make up the variables of the target dataset.

STEP 3: USE SAS XML MAPPER TO CREATE THE MAP FILE

It is time to create the map file. The **XML Pane** displays a tree view of the XML document where each node or attribute is a node in the tree. At the same time, the **Source Pane** displays the text of the XML document while **the XML Map Pane** is initially empty. To build an XML Map file we must drag the elements that we intend to read in from **the XML Pane** into the **XML Map Pane**. The first step is to drag the element that will constitute the basis for our entire dataset. In the case of the client_list.xml file the element that makes the logical choice for a table name is the <client> element.

To select the client element simply click and drag it from the **XML Pane** into **the XML Map Pane** dropping it over the SXLEMAP node. This action creates a table node in the XML Map pane view with the name “client”. The elements and attributes selected for the dataset columns must be dragged and dropped over the “client” node in the XML Map **Pane**. The resulting tree is displayed in Figure 10. Notice how, the **Source Pane** displays the code of the automatically generated map file.

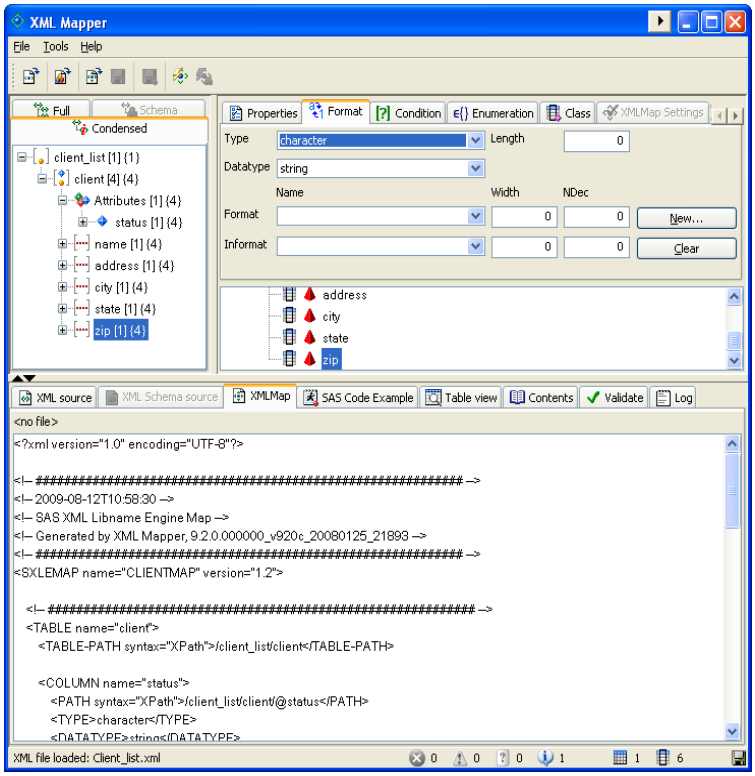



Figure 10

Finally, we change the default name of the XML Map (SXLEMAP) to a name meaningful to our program by clicking on the SXLEMAP node of the **XML Map Pane** and entering a new name in the **Properties** tab. A partial listing of the generated XML Map file can be seen in Figure 11.

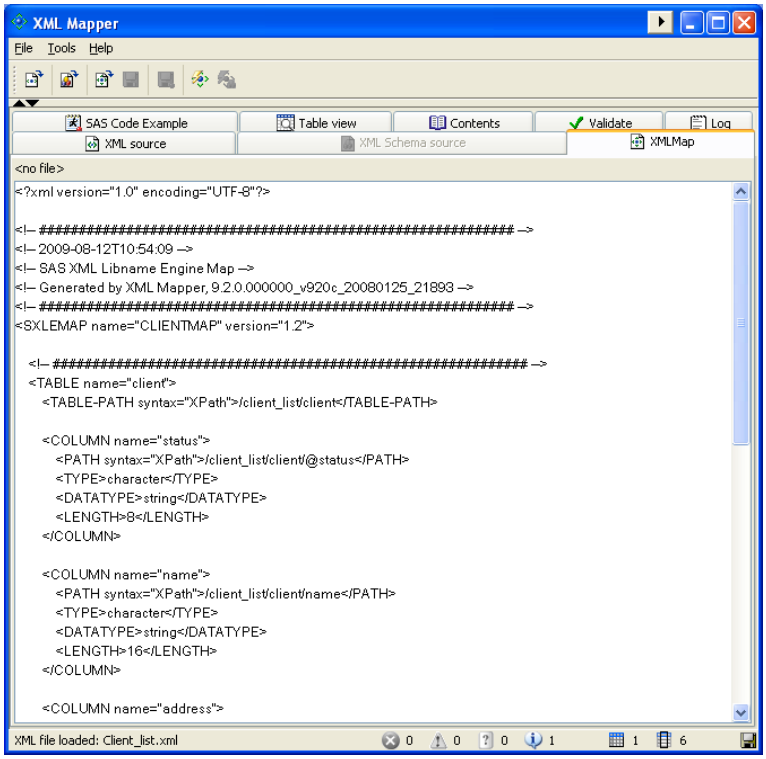



Figure 11

We then save the XML Map file by selecting the **Save XMLMap As…** item from the **File** menu.

STEP 4: USE SAS XML MAPPER TO GENERATE THE SAS CODE TO READ IN THE FILE AND REFERENCE THE MAP FILE

After we save the XML Map file, the automatically generated SAS code in the **SAS Code Example** tab of the **Source Pane** will be complete (see Figure 12). The SAS code can be saved to a .sas file by selecting the **Save SAS As…** item from the **File** menu. As you can see in Figure 12, the example code references the .map file we created in the XML libname xmlmap= option. The SAS code includes a PROC CONTENTS and PROC PRINT on a subset of observations. The PROC CONTENTS and PROC PRINT are excellent tools to check that the source data was mapped properly.

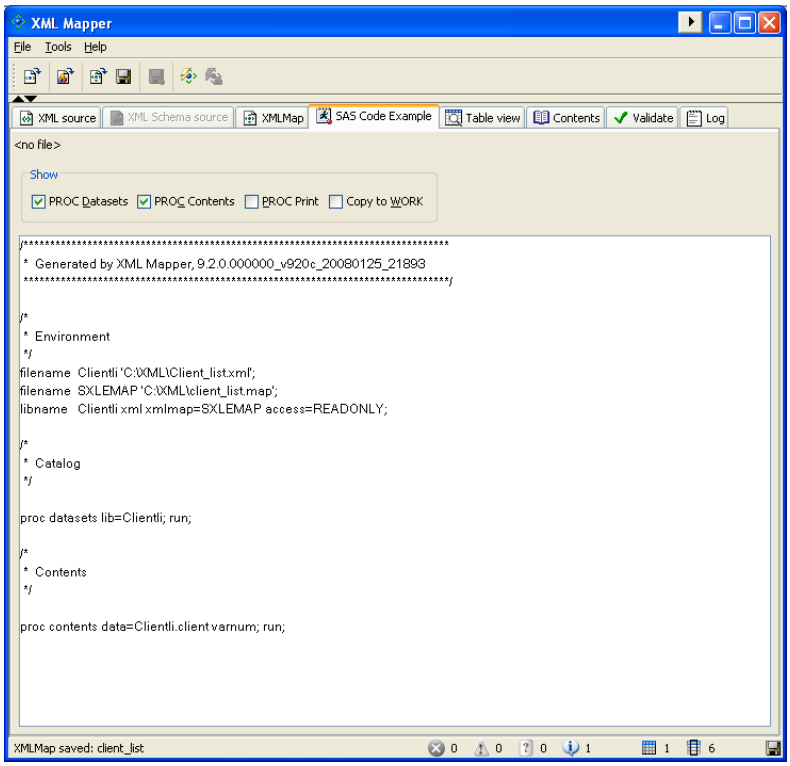



Figure 12

STEP 5: SUBMIT YOUR PROGRAM AND CHECK THAT YOUR CONVERSION WORKED

An excerpt from PROC CONTENTS on the converted file appears below. We successfully read in all elements as well as the status attribute, and mapped the zip element to a character variable.

Variables in Creation Order

| # | Variable | Type | Len | Format | Informat | Label |
|---|---|---|---|---|---|---|
| 1 | status | Char | 8 | $8. | $8. | status |
| 2 | name | Char | 16 | $16. | $16. | name |
| 3 | address | Char | 16 | $16. | $16. | address |
| 4 | city | Char | 11 | $11. | $11. | city |
| 5 | state | Char | 2 | $2. | $2. | state |
| 6 | zip | Char | 5 | $5. | $5. | zip |

STEP 6: WRITE XML FILE USING XMLMAP FILES.
The new XML92 engine (available on SAS 9.2) now supports XML Maps on output as well as input. Below an example on how to read the XML file using XMLMAP file, performing updates to the data and then write again an other XML.

```
filename  Clientli 'C:\XML\Client_list.xml';
filename  SXLEMAP 'C:\XML\client_list.map';
libname   Clientli xml xmlmap=SXLEMAP access=READONLY;

data client; set clientli.client; run;  /*loads data into a dataset  */
……   /* data modification  */

filename  Clientlo 'C:\XML\Client_list_output.xml';
filename  SXLEMAP 'C:\XML\client_list.map';
libname   Clientlo xml92 xmlmap=SXLEMAP;
data clientlo.client; set client; run;
```

## CONCLUSION

This paper has presented the basic processes for reading and writing XML data to and from SAS. SAS offers several ways to handle XML files depending on the structure of the XML data. If the data comes in a rectangular format, XML libname can process it directly. If the XML data has a complex hierarchical structure, you will need to create a map to inform SAS which elements you wish to extract and point to their location in the hierarchy of the source XML document. The new production-version XML Mapper tool, provided as a stand-alone application in SAS 9.1, greatly simplifies this process.

## REFERENCES

Castro, Elizabeth. XML for the World Wide Web. Peachpit Press: Berkeley, CA. 2001.

Friebel, Anthony. “XML? We do that!”. Paper 173-28. SUGI 28 Proceedings. SAS: Cary, NC. 2003.

SAS Institute Inc. 2003. SAS OnlineDoc® 9.1. SAS: Cary, NC. http://support.sas.com/91doc/docMainpage.jsp

“Using ODS to Export Markup Languages” SAS: Cary, NC 2002.
http://www.sas.com/rnd/base/topics/odsmarkup/index.html

Paper 119-29 SUGI29 Reading and Writing XML files from SAS®
Miriam Cisternas, Ovation Research Group, Carlsbad, CA
Ricardo Cisternas, MGC Data Services, Carlsbad, CA

## CONTACT INFORMATION

Marco Bertolino
Actelion Pharmaceuticals Ltd
Via delle Valli 26
Imperia/18100 Italy
Email: marco.bertolino@actelion.com
WEB: www.actelion.com