THE POWER TO KNOW®

SAS® 9.2 XML LIBNAME Engine

User's Guide

Second Edition

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2010. *SAS® 9.2 XML LIBNAME Engine: User's Guide, Second Edition*. Cary, NC: SAS Institute Inc.

SAS® 9.2 XML LIBNAME Engine: User's Guide, Second Edition

ISBN 978-1-60764-447-7

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st electronic book, May2010
1st printing, May 2010

Contents

What's New

Overview

SAS 9.2 provides two versions for XML LIBNAME engine functionality.

- □ By specifying the engine nickname **XML**, you access the SAS 9.1.3 XML engine functionality.
- □ By specifying the engine nickname **XML92**, you access the new SAS 9.2 XML engine functionality with enhancements and changes.

  Note: In the third maintenance release for SAS 9.2, XML92 functionality is production, except for the WSDL markup type, which is preproduction. △

  Note: In the z/OS environment, XML92 functionality is preproduction. △

The **XML92** engine nickname provides the following enhancements and changes:

- □ an enhanced LIBNAME statement
- □ new XMLMap functionality
- □ diagnostics of obsolete syntax

See "Understanding the XML LIBNAME Engine Versions: XML and XML92" on page 85.

Enhanced LIBNAME Statement

The "LIBNAME Statement Syntax" on page 87 has the following enhancements for the **XML92** engine nickname. Some functionality is for SAS 9.2 Phase 2 and later, and some functionality is for the third maintenance release for SAS 9.2.

- □ In addition to assigning a libref to a specific XML document, you can assign a libref to the physical location of a SAS library in a directory-based environment.
- □ The XMLTYPE= option now supports the XMLMAP markup type. The XMLMAP markup type specifies that the XML markup is determined by an XMLMap, which tells the XML engine how to interpret an XML document that is being imported or a SAS data set that is being exported.

- For SAS 9.2 Phase 2 and later, the XMLTYPE= option supports the WSDL markup type. The WSDL markup type supports the markup standards of the Web Services Description Language (WSDL). Specifying the WSDL markup type enables the XML engine to invoke a Web service and import the Web service results. The PARMS= data set option is provided to specify the input parameters SAS data set when importing Web service results. See "Importing Web Service Results Using the WSDL Markup Type (Preproduction)" on page 35 and "PARMS= Data Set Option" on page 117.
- For the third maintenance release for SAS 9.2, there are several new options for the WSDL markup type. For a list of the new options, see Table 7.1 on page 88.

For both the `XML` and `XML92` engine nicknames, the value RELAX for the XMLPROCESS= option has changed to PERMIT. XMLPROCESS=PERMIT accepts character data that does not conform to World Wide Web Consortium (W3C) specifications. Use XMLPROCESS=PERMIT cautiously; this value is provided as a convenience. If an XML document has non-escaped characters, the content is not standard XML construction.

New XMLMap Functionality

XMLMap functionality has the following enhancements for the `XML92` engine nickname:

- You can now export an XML document from a SAS data set using the XMLMap that was created to import the XML document. The XMLMap tells the XML engine how to map the SAS format (variables and observations) into the specific XML document structure. See Chapter 4, "Exporting XML Documents Using an XMLMap," on page 41.
- The "XMLMap Syntax Version 1.9" on page 100 has the following enhancements:
  - The new OUTPUT, HEADING, ATTRIBUTE, and TABLEREF elements enable you to include file attribute information in the exported XML document.
  - The COLUMN element now supports the replace= attribute, which controls concatenation of data. The COLUMN element also supports the class= attribute, which determines the type of generated variable.
  - The new DECREMENT-PATH element defines when to decrement the accumulated value for a counter variable.

Obsolete Syntax

For the `XML92` engine nickname, the following syntax is obsolete:

- The OIMDBM and HTML markup types for the XMLTYPE= option in the LIBNAME statement are obsolete.
- The OIMSTART= option in the LIBNAME statement is obsolete.
- You can no longer specify the XMLMAP= option as a data set option. You must specify the XMLMap in the LIBNAME statement for the XML engine.

PART 1

Usage

CHAPTER

1

Getting Started with the XML Engine

What Does the XML Engine Do?

The XML engine processes an *XML document*. The engine can:

- export (write to an output location) an XML document from a SAS data set of type DATA by translating the SAS proprietary file format to XML markup. The output XML document can then be:
    - used by a product that processes XML documents.
    - moved to another host for the XML engine to process by translating the XML markup back to a SAS data set.
- import (read from an input location) an external XML document. The input XML document is translated to a SAS data set.

Understanding How the XML Engine Works

Assigning a Libref

The XML engine works much like other SAS engines. That is, you execute a LIBNAME statement in order to assign a libref and specify an engine. You use that libref throughout the SAS session where a libref is valid.

A libref for the XML engine can be associated with either a specific XML document or, for the XML92 engine nickname, the physical location of a SAS library in a directory-based environment. When you use the libref, SAS either translates the data in a SAS data set into XML markup, or translates the XML markup into SAS format.

Importing an XML Document

To import an XML document as a SAS data set, the following LIBNAME statement assigns a libref to a specific XML document and specifies the XML engine:

```
libname myxml xml 'C:\My Files\XML\Students.xml';
```

Executing the DATASETS procedure shows that SAS interprets the XML document as a SAS data set:

```
proc datasets library=myxml;
```

Output 1.1 PROC DATASETS Output for MYXML Library

```
                                             Directory

          Libref         MYXML
          Engine         XML
          Physical Name  C:\My Files\XML\Students.xml
          XMLType        GENERIC
          XMLMap         NO XMLMAP IN EFFECT


                                                    Member
                                     #  Name      Type

                                     1  STUDENTS  DATA
```

The PRINT procedure results in the following output:

```
proc print data=myxml.students;
run;
```

Output 1.2 PROC PRINT Output of SAS Data Set MYXML.STUDENTS

```
                               The SAS System

     Obs    STATE     CITY          ADDRESS           NAME                 ID

       1    Texas     Huntsville    1611 Glengreen    Brad Martin        1755
       2    Texas     Houston       11900 Glenda      Zac Harvell        1522
```

Exporting an XML Document

To export an XML document from a SAS data set, the LIBNAME statement for the XML engine assigns a libref to the XML document to be created.

In the following code, the first LIBNAME statement assigns the libref MYFILES to the SAS library that contains the SAS data set Singers. The second LIBNAME statement assigns the libref MYXML to the physical location of the XML document that is to be exported from Myfiles.Singers:

```
libname myfiles 'C:\My Files\';

libname myxml xml 'C:\My Files\XML\Singers.xml';
```

Executing these statements creates the XML document named Singers.XML:

```
data myxml.Singers;
   set myfiles.Singers;
run;
```

Output 1.3 XML Document Singers.XML

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <SINGERS>
      <FirstName> Tom </FirstName>
      <Age> 62 </Age>
   </SINGERS>
   <SINGERS>
      <FirstName> Willie </FirstName>
      <Age> 70 </Age>
   </SINGERS>
   <SINGERS>
      <FirstName> Randy </FirstName>
      <Age> 43 </Age>
   </SINGERS>
</TABLE>
```

SAS Processing Supported by the XML Engine

The XML engine supports the following processing:

- □ The XML engine supports input (read) and output (create) processing. The XML engine does not support update processing.
- □ The XML engine is a *sequential access* engine in that it processes data one record after the other. The engine starts at the beginning of the file and continues in sequence to the end of the file. The XML engine does not provide random (direct) access, which is required for some SAS applications and features. For example, with the XML engine, you cannot use the SORT procedure or ORDER BY in the SQL procedure. If you request processing that requires random access, a message in the SAS log notifies you that the processing is not valid for sequential access. If this message occurs, put the XML data into a temporary SAS data set before you continue.

Transferring an XML Document Across Environments

When you transfer an XML document across environments (for example using FTP), you must be aware of the document's content in order to determine the appropriate transfer mode. If the document contains either an encoding attribute in the XML declaration or if a byte order mark precedes the XML declaration, transfer the file in binary mode. If the document contains neither criteria and you are transferring the document across similar hosts, transfer the file in text mode.

When you export an XML document using the XML engine, by default, the XML document contains an encoding attribute in the XML declaration from the SAS data set's encoding (for example, **<?xml version="1.0" *encoding="windows-1252"* ?>**).

You can override the SAS data set's encoding when you export the XML document by specifying the XMLENCODING= LIBNAME statement option.

Frequently Asked Questions

Is the XML Engine a DOM or SAX Application?

The XML engine uses a Simple API for XML (SAX) model, not a Document Object Model (DOM). SAX does not provide a random-access lookup to the document's contents; it scans the document sequentially and presents each item to the application one item at a time.

Does the XML Engine Validate an XML Document?

The XML engine does not validate an input XML document. The engine assumes that the data passed to it is in valid, well-formed XML markup. Because the engine does not use a DTD (Document Type Definition) or SCHEMA, there is nothing to validate against.

What Is the Difference between Using the XML Engine and the ODS MARKUP Destination?

Typically, you use the XML engine to transport data, while the ODS MARKUP destination is used to create XML from SAS output. The XML engine creates and reads XML documents; ODS MARKUP creates but does not read XML documents.

Why Do I Get Errors When Importing XML Documents Not Created with SAS?

The XML engine reads only files that conform to the markup types supported in the XMLTYPE= LIBNAME statement option. Attempting to import free-form XML documents that do not conform to the specifications required by the supported markup types will generate errors. To successfully import files that do not conform to the XMLTYPE= markup types, you can create a separate XML document, called an XMLMap. The XMLMap syntax tells the XML engine how to interpret the XML markup into a SAS data set or data sets, variables (columns), and observations (rows).

See Chapter 3, "Importing XML Documents," on page 21, Chapter 5, "Importing XML Documents Using an XMLMap," on page 45, "LIBNAME Statement Syntax" on page 87, and Chapter 8, "XMLMap Files for the XML Engine," on page 99.

What Are the XML Engine Nicknames?

SAS 9.2 provides two versions of XML LIBNAME engine functionality by supporting the engine nicknames `XML` and `XML92` in the LIBNAME statement. See "Understanding the XML LIBNAME Engine Versions: XML and XML92" on page 85.

CHAPTER

2

Exporting XML Documents

Understanding How to Export an XML Document

Exporting an XML document is the process of writing a SAS data set of type DATA to an output XML document. The XML engine exports an XML document by translating SAS proprietary format to XML markup.

To export an XML document, you execute the LIBNAME statement for the XML engine in order to assign a libref to the physical location of an XML document to be created. Then, you execute SAS code that produces output such as a DATA step or the COPY procedure.

Exporting an XML Document for Use by Oracle

This example exports an XML document from a SAS data set for use by Oracle. By specifying the ORACLE markup type, the XML engine generates tags that are specific to Oracle standards.

The following output shows the SAS data set MYFILES.CLASS to be exported to Oracle.

Output 2.1 SAS Data Set MYFILES.CLASS to Be Exported for Use by Oracle

```
                 Obs    Name       Gender Age    Height    Weight

                   1    Alfred      M      14     69.0      112.5
                   2    Alice       F      13     56.5       84.0
                   3    Barbara     F      13     65.3       98.0
                   4    Carol       F      14     62.8      102.5
                   5    Henry       M      14     63.5      102.5
                   6    James       M      12     57.3       83.0
                   7    Jane        F      12     59.8       84.5
                   8    Janet       F      15     62.5      112.5
                   9    Jeffrey     M      13     62.5       84.0
                  10    John        M      12     59.0       99.5
                  11    Joyce       F      11     51.3       50.5
                  12    Judy        F      14     64.3       90.0
                  13    Louise      F      12     56.3       77.0
                  14    Mary        F      15     66.5      112.0
                  15    Philip      M      16     72.0      150.0
                  16    Robert      M      12     64.8      128.0
                  17    Ronald      M      15     67.0      133.0
                  18    Thomas      M      11     57.5       85.0
                  19    William     M      15     66.5      112.0
```

The following SAS program exports an XML document from the SAS data set MYFILES.CLASS:

```
libname myfiles 'SAS-library'; ❶

libname trans xml 'XML-document' xmltype=oracle; ❷

data trans.class; ❸
   set myfiles.class;
run;
```

1 The first LIBNAME statement assigns the libref MYFILES to the physical location of the SAS library that stores the SAS data set CLASS. The V9 engine is the default.

2 The second LIBNAME statement assigns the libref TRANS to the physical location of the file (complete pathname, filename, and file extension) that will store the exported XML document and specifies the XML engine. The engine option XMLTYPE=ORACLE produces tags that are equivalent to the Oracle 8i XML implementation.

3 The DATA step reads the SAS data set MYFILES.CLASS and writes its content in ORACLE XML markup to the specified XML document.

Here is the resulting XML document.

Output 2.2 XML Document Exported from MYFILES.CLASS to Be Used by Oracle

```
<?xml version="1.0" encoding="windows-1252" ?>
<ROWSET>
 <ROW>
  <Name> Alfred </Name>
  <Gender> M </Gender>
  <Age> 14 </Age>
  <Height> 69 </Height>
  <Weight> 112.5 </Weight>
 </ROW>
 <ROW>
  <Name> Alice </Name>
  <Gender> F </Gender>
  <Age> 13 </Age>
  <Height> 56.5 </Height>
  <Weight> 84 </Weight>
 </ROW>
.
.
.
 <ROW>
  <Name> William </Name>
  <Gender> M </Gender>
  <Age> 15 </Age>
  <Height> 66.5 </Height>
  <Weight> 112 </Weight>
 </ROW>
</ROWSET>
```

Exporting an XML Document Containing SAS Dates, Times, and Datetimes

This example exports an XML document from a SAS data set that contains datetime, date, and time values. The XML document is generated for the GENERIC markup type.

First, the following SAS program creates a simple SAS data set and prints the contents of the data set. The variable DateTime contains a datetime value, Date contains a date value, and Time contains a time value.

```
data test;
   DateTime=14686;
   format DateTime datetime.;
   Date=14686;
   format Date date9.;
   Time=14686;
   format Time timeampm.
;

proc print data=test;
run;
```

Output 2.3 PROC PRINT of SAS Data Set WORK.TEST Containing SAS Dates, Times, and Datetimes

```
                              The SAS System                              1

               Obs        DateTime              Date        Time

                1     01JAN60:04:04:46    17MAR2000    4:04:46 AM
```

The following code exports an XML document for the GENERIC markup type that includes the SAS date, time, and datetime information:

```
libname trans xml 'XML-document' xmltype=generic; ❶

data trans.test; ❷
   set work.test;
run;
```

1 The LIBNAME statement assigns the libref TRANS to the physical location of the file (complete pathname, filename, and file extension) that will store the exported XML document and specifies the XML engine. XMLTYPE= specifies the GENERIC markup type, which is the default.

2 The DATA step reads the SAS data set WORK.TEST and writes its content in XML markup to the specified XML document.

Here is the resulting XML document.

Output 2.4 XML Document Using GENERIC Markup

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <TEST>
      <DateTime> 1960-01-01T04:04:46.000000 </DateTime>
      <Date> 2000-03-17 </Date>
      <Time> 04:04:46 </Time>
   </TEST>
</TABLE>
```

Exporting Numeric Values

This example uses a small SAS data set, with a numeric variable that contains values with a high precision. The following SAS program creates the data set with an assigned user-defined format, then exports two XML documents to show the difference in output:

```
libname format xml 'C:\My Documents\format.xml'; ❶

libname prec xml 'C:\My Documents\precision.xml' xmldouble=internal; ❷

data npi; ❸
   do n=1 to 10;
      n_pi = n*3.141592653589793;
      output;
   end;
format n_pi f14.2;
run;

data format.dbltest; ❹
   set npi;
run;

data prec.rawtest; ❺
   set npi;
run;
```

```
title 'Drops the Precision'; ❻
proc print data=format.dbltest;
   format n_pi f14.10;
run;

title 'Keeps the Precision'; ❼
proc print data=prec.rawtest;
   format n_pi f14.10;
run;
```

1 The first LIBNAME statement assigns the libref FORMAT to the file that will store the generated XML document FORMAT.XML. The default behavior for the engine is that an assigned SAS format controls numeric values.

2 The second LIBNAME statement assigns the libref PREC to the file that will store the generated XML document PRECISION.XML. The XMLDOUBLE= option specifies INTERNAL, which causes the engine to retrieve the stored raw values.

3 The DATA step creates the temporary data set NPI. The data set has a numeric variable that contains values with a high precision. The variable has an assigned user-defined format that specifies two decimal points.

4 The DATA step creates the data set FORMAT.DBLTEST from WORK.NPI.

5 The DATA step creates the data set PREC.RAWTEST from WORK.NPI.

6 From the data set FORMAT.DBLTEST, PROC PRINT generates the XML document FORMAT.XML, which contains numeric values controlled by the SAS format.

Output 2.5 XML Document FORMAT.XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<TABLE>
   <DBLTEST>
      <n> 1 </n>
      <n_pi>               3.14 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 2 </n>
      <n_pi>               6.28 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 3 </n>
      <n_pi>               9.42 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 4 </n>
      <n_pi>              12.57 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 5 </n>
      <n_pi>              15.71 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 6 </n>
      <n_pi>              18.85 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 7 </n>
      <n_pi>              21.99 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 8 </n>
      <n_pi>              25.13 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 9 </n>
      <n_pi>              28.27 </n_pi>
   </DBLTEST>
   <DBLTEST>
      <n> 10 </n>
      <n_pi>              31.42 </n_pi>
   </DBLTEST>
</TABLE>
```

For the PRINT procedure output, a format was specified in order to show the precision loss. In the output, the decimals after the second digit are zeros. Here is the procedure output.

Output 2.6 PRINT Procedure Output for FORMAT.DBLTEST

```
                    Drops the Precision            1

                 Obs              N_PI           N

                   1      3.1400000000           1
                   2      6.2800000000           2
                   3      9.4200000000           3
                   4     12.5700000000           4
                   5     15.7100000000           5
                   6     18.8500000000           6
                   7     21.9900000000           7
                   8     25.1300000000           8
                   9     28.2700000000           9
                  10     31.4200000000          10
```

7 From the data set PREC.RAWTEST, PROC PRINT generates the XML document PRECISION.XML, which contains the stored numeric values.

Output 2.7 XML Document PRECISION.XML

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<TABLE>
   <RAWTEST>
      <n rawvalue="QRAAAAAAAAA="> 1 </n>
      <n_pi rawvalue="QTJD9qiIWjA=">               3.14 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QSAAAAAAAAA="> 2 </n>
      <n_pi rawvalue="QWSH7VEQtGA=">               6.28 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QTAAAAAAAAA="> 3 </n>
      <n_pi rawvalue="QZbL4/mZDpA=">               9.42 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QUAAAAAAAAA="> 4 </n>
      <n_pi rawvalue="QckP2qIhaMA=">              12.57 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QVAAAAAAAAA="> 5 </n>
      <n_pi rawvalue="QftT0UqpwvA=">              15.71 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QWAAAAAAAAA="> 6 </n>
      <n_pi rawvalue="QhLZfH8zIdI=">              18.85 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QXAAAAAAAAA="> 7 </n>
      <n_pi rawvalue="QhX9u+m7p3U=">              21.99 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QYAAAAAAAAA="> 8 </n>
      <n_pi rawvalue="Qhkh+1RELRg=">              25.13 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QZAAAAAAAAA="> 9 </n>
      <n_pi rawvalue="QhxGOr7Msrs=">              28.27 </n_pi>
   </RAWTEST>
   <RAWTEST>
      <n rawvalue="QaAAAAAAAAA="> 10 </n>
      <n_pi rawvalue="Qh9qeilVOF4=">              31.42 </n_pi>
   </RAWTEST>
</TABLE>
```

For the PRINT procedure output, a format was specified in order to show the retained precision. Here is the procedure output.

Output 2.8 PRINT Procedure Output from PREC.RAWTEST

```
                      Keeps the Precision           2

                  Obs             N_PI           N

                    1      3.1415926536          1
                    2      6.2831853072          2
                    3      9.4247779608          3
                    4     12.5663706144          4
                    5     15.7079632679          5
                    6     18.8495559215          6
                    7     21.9911485751          7
                    8     25.1327412287          8
                    9     28.2743338823          9
                   10     31.4159265359         10
```

Exporting an XML Document with Separate Metadata

This example exports an XML document from a SAS data set and specifies a separate file to contain metadata-related information. The example illustrates using the XMLMETA= option and XMLSCHEMA= option and uses a SAS data set that was created from a Microsoft Access database.

The following SAS program exports an XML document from the SAS data set MYFILES.SUPPLIERS:

```
libname input 'c:\My Documents\myfiles'; ❶

filename xsd 'c:\My Documents\XML\suppliers.xsd'; ❷

libname output xml 'c:\My Documents\XML\suppliers.xml' xmltype=msaccess
   xmlmeta=schemadata xmlschema=xsd; ❸

data output.suppliers; ❹
   set input.suppliers;
run;
```

1 The first LIBNAME statement assigns the libref INPUT to the physical location of the SAS library that stores the SAS data set SUPPLIERS.

2 The FILENAME statement assigns the fileref XSD to the physical location of the separate external file that will contain the metadata-related information.

3 The second LIBNAME statement assigns the libref OUTPUT to the physical location of the file (complete pathname, filename, and file extension) that will store the exported XML document and specifies the XML engine. The engine options

- □ XMLTYPE=MSACCESS supports the markup standards for a Microsoft Access database.
- □ XMLMETA=SCHEMADATA specifies to include both data content and metadata-related information in the exported markup.
- □ XMLSCHEMA= specifies the fileref that is assigned, in the previous FILENAME statement, to the separate external file that will contain the metadata-related information.

4 The DATA step reads the SAS data set INPUT.SUPPLIERS and writes its data content in Microsoft Access database XML markup to the XML document Suppliers.XML, and then writes the metadata information to the separate external file Suppliers.XSD.

Here is part of the resulting XML document.

Output 2.9 XML Document Suppliers.XML

```
<?xml version="1.0" encoding="windows-1252" ?>
   <dataroot  xmlns:xs="http://www.w3.org/2001/XMLSchema"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:od="urn:schemas-microsoft-com:officedata"
              xsi:noNamespaceSchemaLocation="suppliers.xsd">
      <SUPPLIERS>
         <HOMEPAGE/>
         <FAX/>
         <PHONE>(272) 444-2222</PHONE>
         <COUNTRY>UK</COUNTRY>
         <POSTALCODE>EC1 4SD</POSTALCODE>
         <REGION/>
         <CITY>London</CITY>
         <ADDRESS>49 Franklin St.</ADDRESS>
         <CONTACTTITLE>Purchasing Manager</CONTACTTITLE>
         <CONTACTNAME>Charlotte Smith</CONTACTNAME>
         <COMPANYNAME>Exotic Flowers</COMPANYNAME>
         <SUPPLIERID>1</SUPPLIERID>
      </SUPPLIERS>
      <SUPPLIERS>
         <HOMEPAGE>#MYCAJUN.HTM#</HOMEPAGE>
         <FAX/>
         <PHONE>(512) 284-3677</PHONE>
         <COUNTRY>USA</COUNTRY>
         <POSTALCODE>70117</POSTALCODE>
         <REGION>LA</REGION>
         <CITY>New Orleans</CITY>
         <ADDRESS>P.O. Box 78934</ADDRESS>
         <CONTACTTITLE>Order Administrator</CONTACTTITLE>
         <CONTACTNAME>Shelley Martin</CONTACTNAME>
         <COMPANYNAME>New Orleans Cajun Foods</COMPANYNAME>
         <SUPPLIERID>2</SUPPLIERID>
      </SUPPLIERS>
          .
          .
          .
   </dataroot>
```

And here is the separate metadata information.

Output 2.10 Separate Metadata Information Suppliers.XSD

```
<?xml version="1.0" encoding="windows-1252" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:od="urn:schemas-microsoft-com:officedata">
   <xs:element name="dataroot">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="SUPPLIERS"  minOccurs="0" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="SUPPLIERS">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="HOMEPAGE" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="94" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="FAX" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="15" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="PHONE" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="15" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="COUNTRY" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="11" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="POSTALCODE" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="8" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="REGION" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="8" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
```

```
<xs:element name="CITY" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="13" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="ADDRESS" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="45" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="CONTACTTITLE" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="28" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="CONTACTNAME" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="26" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="COMPANYNAME" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="38" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="SUPPLIERID" minOccurs="0"
                        od:jetType="double" od:sqlSType="double" type="xs:double" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Exporting an XML Document in CDISC ODM Markup

This example exports the SAS data set that was imported in “Importing a CDISC ODM Document” on page 32 back to an XML document that is in CDISC ODM markup. Because the CDISCODM markup type is specified, the XML engine generates tags that are specific to the CDISC Operational Data Model.

The following SAS program exports an XML document from the SAS data set ODM.AE:

```
filename output 'C:\myoutput.xml';❶
libname  output xml xmltype=CDISCODM formatactive=yes;❷

data output.AE2;
   set odm.AE;
run;
```

1 The FILENAME statement assigns the fileref OUTPUT to the physical location of the external file (complete pathname, filename, and file extension) to which the exported information will be written.

2 The LIBNAME statement specifies the fileref OUTPUT as the output location and specifies the XML engine. It includes the following engine options:

- □ XMLTYPE=CDISCODM supports the markup standards for CDISC ODM 1.2.
- □ FORMATACTIVE=YES specifies to convert SAS formats to the corresponding CDISC ODM CodeList elements.

The output is the same as the XML document that is shown in “Example CDISC ODM Document” on page 121.

CHAPTER 3

Importing XML Documents

Understanding How to Import an XML Document

Importing an XML document is the process of reading an external XML document as a SAS data set. The XML engine translates the input XML document to the SAS proprietary file format.

To import an XML document, you execute the LIBNAME statement for the XML engine in order to assign a libref to the physical location of an existing XML document. Then, you execute SAS code to access the XML document as a SAS data set.

Importing an XML Document Using the GENERIC Markup Type

This example imports the following XML document, which conforms to the physical structure for the GENERIC markup type. For information about the required physical structure, see “Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type” on page 45.

```
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
   <CLASS>
      <Name> Alfred </Name>
      <Gender> M </Gender>
      <Age> 14 </Age>
      <Height> 69 </Height>
      <Weight> 112.5 </Weight>
   </CLASS>
   <CLASS>
      <Name> Alice </Name>
      <Gender> F </Gender>
      <Age> 13 </Age>
      <Height> 56.5 </Height>
      <Weight> 84 </Weight>
   </CLASS>
.
.
.
   <CLASS>
      <Name> William </Name>
      <Gender> M </Gender>
      <Age> 15 </Age>
      <Height> 66.5 </Height>
      <Weight> 112 </Weight>
   </CLASS>
</TABLE>
```

The following SAS program translates the XML markup to SAS proprietary format:

```
libname trans xml 'XML-document'; ❶

libname myfiles 'SAS-library'; ❷

data myfiles.class; ❸
   set trans.class;
run;
```

1 The first LIBNAME statement assigns the libref TRANS to the physical location of the XML document (complete pathname, filename, and file extension) and specifies the XML engine. By default, the XML engine expects GENERIC markup.

2 The second LIBNAME statement assigns the libref MYFILES to the physical location of the SAS library that will store the resulting SAS data set. The V9 engine is the default.

3 The DATA step reads the XML document and writes its content in SAS proprietary format.

Issuing the following PRINT procedure produces the output for the data set that was translated from the XML document:

```
proc print data=myfiles.class;
run;
```

Output 3.1 PROC PRINT Output for MYFILES.CLASS

```
                         The SAS System                         1

          Obs    WEIGHT     HEIGHT        AGE    GENDER NAME

            1     112.5       69.0         14    M      Alfred
            2      84.0       56.5         13    F      Alice
            3      98.0       65.3         13    F      Barbara
            4     102.5       62.8         14    F      Carol
            5     102.5       63.5         14    M      Henry
            6      83.0       57.3         12    M      James
            7      84.5       59.8         12    F      Jane
            8     112.5       62.5         15    F      Janet
            9      84.0       62.5         13    M      Jeffrey
           10      99.5       59.0         12    M      John
           11      50.5       51.3         11    F      Joyce
           12      90.0       64.3         14    F      Judy
           13      77.0       56.3         12    F      Louise
           14     112.0       66.5         15    F      Mary
           15     150.0       72.0         16    M      Philip
           16     128.0       64.8         12    M      Robert
           17     133.0       67.0         15    M      Ronald
           18      85.0       57.5         11    M      Thomas
           19     112.0       66.5         15    M      William
```

Importing an XML Document with Numeric Values

This example imports the XML document PRECISION.XML, which was exported in "Exporting Numeric Values" on page 10. This example illustrates how you can change the behavior for importing numeric values.

The first SAS program imports the XML document using the default behavior, which retrieves PCDATA from the element:

```
libname default xml 'C:\My Documents\precision.xml';

title 'Default Method';
proc print data=default.rawtest;
   format n_pi f14.10;
run;
```

The result of the import is the SAS data set DEFAULT.RAWTEST.

Output 3.2 PROC PRINT of Data Set DEFAULT.RAWTEST

```
                    Default Method             1

              Obs               N_PI           N

                1       3.1400000000           1
                2       6.2800000000           2
                3       9.4200000000           3
                4      12.5700000000           4
                5      15.7100000000           5
                6      18.8500000000           6
                7      21.9900000000           7
                8      25.1300000000           8
                9      28.2700000000           9
               10      31.4200000000          10
```

The second SAS program imports the XML document using the XMLDOUBLE= option on page 93 in order to change the behavior, which retrieves the value from the rawdata= attribute in the element:

```
libname new xml 'C:\My Documents\precision.xml' xmldouble=internal;

title 'Precision Method';
proc print data=new.rawtest;
   format n_pi f14.10;
run;
```

The result of the import is SAS data set NEW.RAWTEST.

Output 3.3 PROC PRINT of Data Set NEW.RAWTEST

```
                  Precision Method            2

              Obs               N_PI           N

                1       3.1415926536           1
                2       6.2831853072           2
                3       9.4247779608           3
                4      12.5663706144           4
                5      15.7079632679           5
                6      18.8495559215           6
                7      21.9911485751           7
                8      25.1327412287           8
                9      28.2743338823           9
               10      31.4159265359          10
```

Importing an XML Document with Non-Escaped Character Data

W3C specifications (section 4.6 Predefined Entities) state that for character data, certain characters such as the left angle bracket (<), the ampersand (&), and the apostrophe (') must be escaped using character references or strings like **&lt;**, **&amp;**, and **&apos;**. For example, to allow attribute values to contain both single and double quotation marks, the apostrophe or single-quotation character (') can be represented as **&apos;** and the double-quotation character (") as **&quot;**.

To import an XML document that contains non-escaped characters, you can specify the LIBNAME statement option XMLPROCESS=PERMIT in order for the XML engine to accept character data that does not conform to W3C specifications. That is, non-escaped characters like the apostrophe, double quotation marks, and the ampersand are accepted in character data.

Note: Use XMLPROCESS=PERMIT cautiously. If an XML document consists of non-escaped characters, the content is not standard XML construction. The option is provided for convenience, not to encourage invalid XML markup. △

This example imports the following XML document named Permit.XML, which contains non-escaped character data:

```
<?xml version="1.0" ?>
<PERMIT>
   <CHARS>
      <accept>OK</accept>
      <status>proper escape sequence</status>
      <ampersand>&amp;</ampersand>
      <squote>&apos;</squote>
      <dquote>&quot;</dquote>
      <less>&lt;</less>
      <greater>&gt;</greater>
   </CHARS>
   <CHARS>
       <accept>OK</accept>
       <status>unescaped character in CDATA</status>
       <ampersand><![CDATA[Abbott & Costello]]></ampersand>
       <squote><![CDATA[Logan’s Run]]></squote>
       <dquote><![CDATA[This is "realworld" stuff]]></dquote>
       <less><![CDATA[ e < pi ]]></less>
       <greater><![CDATA[ pen > sword ]]></greater>
    </CHARS>

   <CHARS>
      <accept>NO</accept>
      <status>single unescaped character</status>
      <ampersand>&</ampersand>
      <squote>’</squote>
      <dquote>"</dquote>
      <less></less>
      <greater></greater>
   </CHARS>
   <CHARS>
      <accept>NO</accept>
      <status>unescaped character in string</status>
      <ampersand>Dunn & Bradstreet</ampersand>
      <squote>Isn’t this silly?</squote>
      <dquote>Quoth the raven, "Nevermore!"</dquote>
      <less></less>
      <greater></greater>
   </CHARS>
</PERMIT>
```

First, using the default XML engine behavior, which expects XML markup to conform to W3C specifications, the following SAS program imports only the first two observations, which contain valid XML markup, and produces errors for the last two records, which contain non-escaped characters:

```
libname permit xml ’c:\My Documents\XML\permit.xml’;

proc print data=permit.chars;
run;
```

Output 3.4 SAS Log Output

```
ERROR: There is an illegal character in the entity name.
       encountered during XMLInput parsing
       occurred at or near line 24, column 22
NOTE: There were 2 observations read from the data set PERMIT.CHARS.
```

Specifying the LIBNAME statement option XMLPROCESS=PERMIT enables the XML engine to import the XML document:

```
libname permit xml ’c:\My Documents\XML\permit.xml’ xmlprocess=permit;

proc print data=permit.chars;
run;
```

Output 3.5 PROC PRINT Output

```
                                The SAS System                                 1

Obs    GREATER         LESS       DQUOTE                           SQUOTE

  1    >               <          "                                ’
  2    pen > sword     e < pi     This is "realworld" stuff        Logan’s Run
  3                               "                                ’
  4                               Quoth the raven, "Nevermore!"    Isn’t this silly?

Obs    AMPERSAND            STATUS                       ACCEPT

  1    &                    proper escape sequence         OK
  2    Abbott & Costello    unescaped character in CDATA   OK
  3    &                    single unescaped character     NO
  4    Dunn & Bradstreet    unescaped character in string  NO
```

Importing an XML Document Created by Microsoft Access

This example imports the following XML document, which was created from a Microsoft Access database. Because the XML document contains an embedded XML schema, you must specify the MSACCESS markup type rather than the default GENERIC type. MSACCESS obtains a variable’s attributes from the embedded schema.

```
<?xml version="1.0" encoding="windows-1252" ?>
<root      xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:od="urn:schemas-microsoft-com:officedata">
   <xs:schema>
```

```
<xs:element name="dataroot">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="SUPPLIERS"  minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="SUPPLIERS">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="HOMEPAGE" minOccurs="0"
                     od:jetType="text" od:sqlSType="nvarchar">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:maxLength value="94" />
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="FAX" minOccurs="0"
                     od:jetType="text" od:sqlSType="nvarchar">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:maxLength value="15" />
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="PHONE" minOccurs="0"
                     od:jetType="text" od:sqlSType="nvarchar">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:maxLength value="15" />
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="COUNTRY" minOccurs="0"
                     od:jetType="text" od:sqlSType="nvarchar">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:maxLength value="11" />
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="POSTALCODE" minOccurs="0"
                     od:jetType="text" od:sqlSType="nvarchar">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:maxLength value="8" />
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="REGION" minOccurs="0"
                     od:jetType="text" od:sqlSType="nvarchar">
            <xs:simpleType>
               <xs:restriction base="xs:string">
```

```
                     <xs:maxLength value="8" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="CITY" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="13" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="ADDRESS" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="45" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="CONTACTTITLE" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="28" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="CONTACTNAME" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="26" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="COMPANYNAME" minOccurs="0"
                        od:jetType="text" od:sqlSType="nvarchar">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="38" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
            <xs:element name="SUPPLIERID" minOccurs="0"
                        od:jetType="double" od:sqlSType="double" type="xs:double" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
<dataroot>
   <SUPPLIERS>
      <HOMEPAGE/>
      <FAX/>
```

```
         <PHONE>(272) 444-2222</PHONE>
         <COUNTRY>UK</COUNTRY>
         <POSTALCODE>EC1 4SD</POSTALCODE>
         <REGION/>
         <CITY>London</CITY>
         <ADDRESS>49 Franklin St.</ADDRESS>
         <CONTACTTITLE>Purchasing Manager</CONTACTTITLE>
         <CONTACTNAME>Charlotte Smith</CONTACTNAME>
         <COMPANYNAME>Exotic Flowers</COMPANYNAME>
         <SUPPLIERID>1</SUPPLIERID>
      </SUPPLIERS>
      <SUPPLIERS>
         <HOMEPAGE>#MYCAJUN.HTM#</HOMEPAGE>
         <FAX/>
         <PHONE>(512) 284-3677</PHONE>
         <COUNTRY>USA</COUNTRY>
         <POSTALCODE>70117</POSTALCODE>
         <REGION>LA</REGION>
         <CITY>New Orleans</CITY>
         <ADDRESS>P.O. Box 78934</ADDRESS>
         <CONTACTTITLE>Order Administrator</CONTACTTITLE>
         <CONTACTNAME>Shelley Martin</CONTACTNAME>
         <COMPANYNAME>New Orleans Cajun Foods</COMPANYNAME>
         <SUPPLIERID>2</SUPPLIERID>
      </SUPPLIERS>
   .
   .
   .
   </dataroot>
</root>
```

The following SAS program interprets the XML document as a SAS data set:

```
libname access xml '/u/myid/myfiles/suppliers.xml' xmltype=msaccess ❶
   xmlmeta=schemadata; ❶

proc print data=access.suppliers (obs=2); ❷
   var contactname companyname;
run;
```

1 The LIBNAME statement assigns the libref ACCESS to the physical location of the XML document (complete pathname, filename, and file extension) and specifies the XML engine. By default, the XML engine expects GENERIC markup, so you must include the XMLTYPE= option in order to read the XML document in MSACCESS markup and to obtain a variable's attributes from the embedded schema. The option XMLMETA=SCHEMADATA specifies to import both data and metadata-related information from the input XML document.

2 The PRINT procedure produces the output. The procedure uses the OBS= data set option to print only the first two observations and the VAR statement to print only specific variables (columns).

Output 3.6 PROC PRINT Output for ACCESS.SUPPLIERS

```
                              The SAS System                              1

   Obs    CONTACTNAME                   COMPANYNAME

     1    Charlotte Smith               Exotic Flowers
     2    Shelley Martin                New Orleans Cajun Foods
```

Using PROC CONTENTS, the output displays the file's attributes as well as the attributes of each interpreted column (variable), such as the variable's type and length, which are obtained from the embedded XML schema. Without the embedded XML schema, the results for the attributes would be default values.

```
proc contents data=access.suppliers;
run;
```

Output 3.7 PROC CONTENTS Output for ACCESS.SUPPLIERS

```
                               The SAS System                            2

                          The CONTENTS Procedure

     Data Set Name        ACCESS.SUPPLIERS    Observations          .
     Member Type          DATA                Variables             12
     Engine               XML                 Indexes               0
     Created              .                   Observation Length    0
     Last Modified        .                   Deleted Observations  0
     Protection                               Compressed            NO
     Data Set Type                            Sorted                NO
     Label
     Data Representation  Default
     Encoding             Default


                 Alphabetic List of Variables and Attributes

    #    Variable        Type    Len    Format    Informat    Label

    8    ADDRESS         Char     45    $45.      $45.        ADDRESS
    7    CITY            Char     13    $13.      $13.        CITY
   11    COMPANYNAME     Char     38    $38.      $38.        COMPANYNAME
   10    CONTACTNAME     Char     26    $26.      $26.        CONTACTNAME
    9    CONTACTTITLE    Char     28    $28.      $28.        CONTACTTITLE
    4    COUNTRY         Char     11    $11.      $11.        COUNTRY
    2    FAX             Char     15    $15.      $15.        FAX
    1    HOMEPAGE        Char     94    $94.      $94.        HOMEPAGE
    3    PHONE           Char     15    $15.      $15.        PHONE
    5    POSTALCODE      Char      8    $8.       $8.         POSTALCODE
    6    REGION          Char      8    $8.       $8.         REGION
   12    SUPPLIERID      Num       8    F8.       F8.         SUPPLIERID
```

Importing Concatenated XML Documents

For a file that is a concatenation of multiple XML documents, you can use the XML engine to import the file. To import concatenated XML documents, simply specify the LIBNAME statement option XMLCONCATENATE=YES.

Note: Use XMLCONCATENATE=YES cautiously. If an XML document consists of concatenated XML documents, the content is not standard XML construction. The option is provided for convenience, not to encourage invalid XML markup. △

This example imports the following file named ConcatStudents.XML, which consists of two XML documents:

```
<?xml version="1.0" ?>
<LIBRARY>
   <STUDENTS>
      <ID>1345</ID>
      <NAME>Linda Kay</NAME>
      <SCHOOL>Bellaire</SCHOOL>
      <CITY>Houston</CITY>
   </STUDENTS>
   <STUDENTS>
      <ID>2456</ID>
      <NAME>Chas Wofford</NAME>
      <SCHOOL>Sam Houston</SCHOOL>
      <CITY>Houston</CITY>
   </STUDENTS>
   <STUDENTS>
      <ID>3567</ID>
      <NAME>Jerry Kolar</NAME>
      <SCHOOL>Sharpstown</SCHOOL>
      <CITY>Houston</CITY>
   </STUDENTS>
</LIBRARY>

<?xml version="1.0" ?>
<LIBRARY>
   <STUDENTS>
      <ID>1234</ID>
      <NAME>Brad Martin</NAME>
      <SCHOOL>Reagan</SCHOOL>
      <CITY>Austin</CITY>
   </STUDENTS>
   <STUDENTS>
      <ID>2345</ID>
      <NAME>Zac Harvell</NAME>
      <SCHOOL>Westwood</SCHOOL>
      <CITY>Austin</CITY>
   </STUDENTS>
   <STUDENTS>
      <ID>3456</ID>
      <NAME>Walter Smith</NAME>
      <SCHOOL>Bowie</SCHOOL>
      <CITY>Austin</CITY>
   </STUDENTS>
</LIBRARY>
```

First, using the default XML engine behavior, which does not support concatenated XML documents (XMLCONCATENATE=NO), the following SAS program imports the first XML document, which consists of three observations, and produces an error for the second XML document:

```
libname concat xml '/u/My Documents/XML/ConcatStudents.xml';

proc datasets library=concat;
```

Output 3.8 SAS Log Output

```
NOTE: Libref CONCAT was successfully assigned as follows:
      Engine:        XML
      Physical Name: /u/My Documents/XML/ConcatStudents.xml
20   proc datasets library=concat;
ERROR: "xml" is illegal as a processing-instruction target name.
       encountered during XMLMap parsing
       occurred at or near line 23, column 7

                                            Directory

       Libref         CONCAT
       Engine         XML
       Physical Name  /u/My Documents/XML/ConcatStudents.xml
       XMLType        GENERIC
       XMLMap         NO XMLMAP IN EFFECT


                                                    Member
                                       #  Name      Type

                                       1  STUDENTS  DATA
```

Specifying the LIBNAME statement option XMLCONCATENATE=YES enables the XML engine to import the concatenated XML documents as one SAS data set:

```
libname concat xml '/u/My Documents/XML/ConcatStudents.xml' xmlconcatenate=yes;

proc print data=concat.students;
run;
```

Output 3.9 PROC PRINT Output

```
                               The SAS System                                     1

            Obs    CITY       SCHOOL         NAME                  ID

              1    Houston    Bellaire       Linda Kay           1345
              2    Houston    Sam Houston    Chas Wofford        2456
              3    Houston    Sharpstown     Jerry Kolar         3567
              4    Austin     Reagan         Brad Martin         1234
              5    Austin     Westwood       Zac Harvell         2345
              6    Austin     Bowie          Walter Smith        3456
```

Importing a CDISC ODM Document

This example imports the XML document that is shown in Appendix 1, “Sample XML Document,” on page 121. The document conforms to Version 1.2 of the CDISC Operational Data Model (ODM). To import a CDISC ODM document, you specify CDISCODM as the XML markup type, and you can specify values for the FORMATACTIVE=, FORMATLIBRARY=, and FORMATNOREPLACE= options.

The following SAS program imports the XML document as a SAS data set:

```
filename odm 'C:\Documents and Settings\myid\MyDocuments\CDISC\AE.XML'; ❶

libname odm xml xmltype=CDISCODM ❷ FormatActive=YES ❸
   FormatNoReplace=NO ❹ FormatLibrary="Work" ❺;

proc print data=odm.AE; ❻
run;
```

1 The FILENAME statement assigns the fileref ODM to the physical location of the XML document (complete pathname, filename, and file extension).

2 The LIBNAME statement uses the fileref to reference the XML document and specifies the XML engine. By default, the XML engine expects GENERIC markup, so you must include the XMLTYPE= option in order to read the XML document in CDISCODM markup.

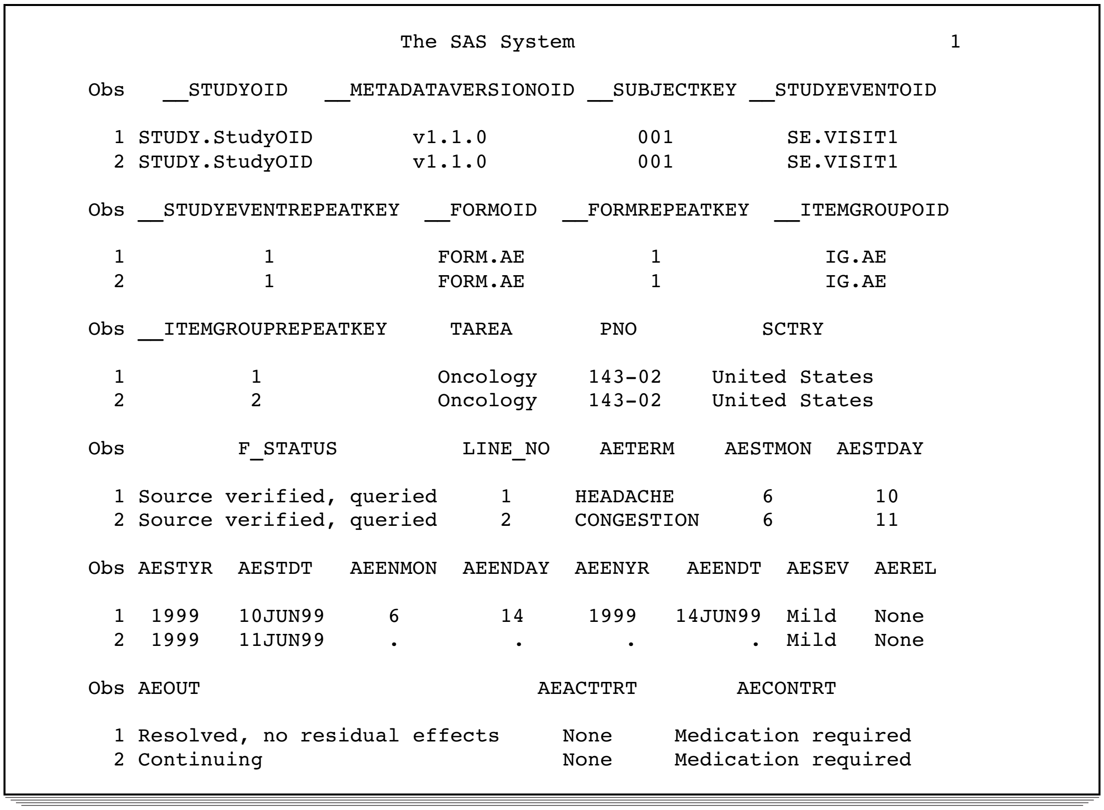
3 FORMATACTIVE=YES specifies to convert CDISC ODM CodeList elements in the document to SAS formats.

4 FORMATNOREPLACE=NO specifies to replace any existing SAS formats in the format catalog that have the same name as the converted formats.

5 FORMATACTIVE="Work" specifies to create the format catalog in the temporary Work library. The Work library is also the default if you omit the FORMATACTIVE= option.

6 The PRINT procedure produces the output.

Output 3.10 PROC PRINT Output for ODM.AE



```
                             The SAS System                             1

   Obs    __STUDYOID    __METADATAVERSIONOID __SUBJECTKEY __STUDYEVENTOID

     1 STUDY.StudyOID        v1.1.0            001         SE.VISIT1
     2 STUDY.StudyOID        v1.1.0            001         SE.VISIT1

   Obs __STUDYEVENTREPEATKEY  __FORMOID  __FORMREPEATKEY  __ITEMGROUPOID

     1           1             FORM.AE          1             IG.AE
     2           1             FORM.AE          1             IG.AE

   Obs __ITEMGROUPREPEATKEY     TAREA       PNO          SCTRY

     1          1              Oncology    143-02    United States
     2          2              Oncology    143-02    United States

   Obs         F_STATUS          LINE_NO    AETERM    AESTMON  AESTDAY

     1 Source verified, queried     1     HEADACHE       6        10
     2 Source verified, queried     2     CONGESTION     6        11

   Obs AESTYR  AESTDT   AEENMON  AEENDAY  AEENYR   AEENDT  AESEV  AEREL

     1  1999   10JUN99     6        14     1999   14JUN99  Mild   None
     2  1999   11JUN99     .         .        .         .  Mild   None

   Obs AEOUT                          AEACTTRT        AECONTRT

     1 Resolved, no residual effects     None     Medication required
     2 Continuing                        None     Medication required
```

The output from PROC CONTENTS displays the file's attributes as well as the attributes of each interpreted column (variable), such as the variable's type and length. The attributes are obtained from the embedded ODM metadata content. The VARNUM option causes the variables to be printed first in alphabetical order and then in the order of their creation.

```
proc contents data=odm.AE varnum;
run;
```

Output 3.11 PROC CONTENTS Output for ODM.AE

```
                               The SAS System                              2

                           The CONTENTS Procedure

          Data Set Name        ODM.AE      Observations          .
          Member Type          DATA        Variables             28
          Engine               XML         Indexes               0
          Created              .           Observation Length    0
          Last Modified        .           Deleted Observations  0
          Protection                       Compressed            NO
          Data Set Type                    Sorted                NO
          Label
          Data Representation  Default
          Encoding             Default


                        Variables in Creation Order

 # Variable           Type Len Format    Informat Label

 1 __STUDYOID         Char 100                    __STUDYOID
 2 __                 Char 100                    __METADATAVERSIONOID
   METADATAVERSIONOID
 3 __SUBJECTKEY       Char 100                    __SUBJECTKEY
 4 __STUDYEVENTOID    Char 100                    __STUDYEVENTOID
 5 __                 Char 100                    __STUDYEVENTREPEATKEY
   STUDYEVENTREPEATKE
   Y
 6 __FORMOID          Char 100                    __FORMOID
 7 __FORMREPEATKEY    Char 100                    __FORMREPEATKEY
 8 __ITEMGROUPOID     Char 100                    __ITEMGROUPOID
 9 __                 Char 100                    __ITEMGROUPREPEATKEY
   ITEMGROUPREPEATKEY
10 TAREA              Char   4 $TAREAF.           Therapeutic Area
11 PNO                Char  15                    Protocol Number
12 SCTRY              Char   4 $SCTRYF.           Country
13 F_STATUS           Char   1 $F_STATU.          Record status, 5 levels,
                                                  internal use
14 LINE_NO            Num    8           2.       Line Number
15 AETERM             Char 100                    Conmed Indication
16 AESTMON            Num    8           2.       Start Month - Enter
                                                  Two Digits 01-12
17 AESTDAY            Num    8           2.       Start Day - Enter
                                                  Two Digits 01-31
18 AESTYR             Num    8           4.       Start Year - Enter
                                                  Four Digit Year
19 AESTDT             Num    8 DATE.              Derived Start Date
20 AEENMON            Num    8           2.       Stop Month - Enter
                                                  Two Digits 01-12
21 AEENDAY            Num    8           2.       Stop Day - Enter
                                                  Two Digits 01-31
22 AEENYR             Num    8           4.       Stop Year - Enter
                                                  Four Digit Year
23 AEENDT             Num    8 DATE.              Derived Stop Date
24 AESEV              Char   1 $AESEV.            Severity
25 AEREL              Char   1 $AEREL.            Relationship to study drug
26 AEOUT              Char   1 $AEOUT.            Outcome
27 AEACTTRT           Char   1 $AEACTTR.          Actions taken re study drug
28 AECONTRT           Char   1 $AECONTR.          Actions taken, other
```

Importing Web Service Results Using the WSDL Markup Type (Preproduction)

What Is a Web Service?

A Web service is a program that runs over a network and enables computers to exchange information over the Internet. For example, a Web service can provide information such as weather data and stock quotes. When the Web service is created, a Web Services Description Language (WSDL) file is created as well. The WSDL file is an XML document that describes the available services, the input and output parameters for the Web service, and other information for interacting with the Web service. WSDL is developed jointly by Microsoft and IBM.

Understanding How to Import Web Service Results

Specifying the WSDL markup type enables the XML LIBNAME engine to invoke a Web service and import the Web service results. The WSDL markup type interacts with a WSDL file that supports the markup standards of WSDL 1.1.

The process of importing Web service results requires several steps and transmissions of data in the following order:

1. In the LIBNAME statement, you must specify the `XML92` engine nickname and the WSDL markup type to access a WSDL file. See "LIBNAME Statement Syntax" on page 87. In addition, in the LIBNAME statement, you can specify WSDL options for importing Web service results. See "Options" on page 88.
2. Before the XML LIBNAME engine can invoke a Web service and import the Web service results, you must determine the contents of the WSDL file. Use the DATASETS procedure, and then the CONTENTS procedure. PROC DATASETS translates the Web service descriptions in the WSDL file markup into SAS data set member names. Then, by submitting PROC CONTENTS for the input parameters member name, you determine the contents of the Web service input parameters such as the variable names and types.
3. After you have the information that you need to interact with the Web service, you must create two SAS data sets. The first data set must match the contents of the Web service input parameters. The second data set will contain the Web service results.

   When you create the SAS data set for the Web service results, this step submits the input parameters and imports the Web service results. To create the SAS data set for the Web service results, use the SET statement:

   □ Specify the libref from the LIBNAME statement (from step 1).

   □ Specify the member name of the Web service results (from step 2).

   □ Include the PARMS= data set option"PARMS= Data Set Option" on page 117 to specify the name of the SAS data set that you created that matches the contents of the Web service input parameters (from step 3).
4. The Web service results SAS data set now contains the results.

Importing Web Service Results

This example illustrates how to import the Web service results that are described by a WSDL file.

The following SAS program accesses the WSDL file to list the Web service descriptions, creates the necessary SAS data sets to contain the input and output parameters, and then imports the Web service results. The Web service that is invoked in this example adds the values for X and Y.

```
filename wsdl url "http://t2824/AddService.asmx?WSDL‘‘; ❶

libname wsdl xml92 xmltype=wsdl; ❷

proc datasets library=wsdl details; ❸
run;

proc contents data=wsdl.Add varnum; ❹
run;

proc contents data=wsdl.AddResponse varnum;  ❹
run;

data work.add; ❺
   x=1;
   y=1;
run;

data work.resp; ❻
   set wsdl.AddResponse (parms=work.add);
run;

proc contents data=work.resp varnum; ❼
run;

proc print data=work.resp; ❽
run;
```

1 The FILENAME statement assigns the fileref WSDL to the WSDL file.

2 The LIBNAME statement uses the WSDL fileref as the libref to reference the WSDL file, specifies the XML92 engine nickname, and specifies the WSDL markup type. To specify a fileref for the WSDL file that does not match the libref, you can use the XMLFILEREF= option.

3 The DATASETS procedure lists, as SAS data sets, the Web service descriptions that are in the WSDL file markup. The results in Output 3.12 show that the Web service has both input parameters and results. The engine translates these as two SAS data sets WSDL.Add and WSDL.AddResponse. The data set label tells you that Add is the input parameters, and AddResponse is the results.

The DETAILS option includes information in the log about the number of observations, variables, indexes, and data set labels. The input parameters data set contains two variables, and the results data set contains one variable.

4 The two CONTENTS procedures describe the contents of the Web service input parameters and results. The VARNUM option prints a list of the variables by their position in the data set. The input parameters data set is shown in Output 3.13, and the results data set is shown in Output 3.14.

The data set label tells you that Add is the input parameters data set, and AddResponse is the results data set. In addition, each PROC CONTENTS output contains the variable attributes such as name, type, and length. The variable label lists the column type as defined in the WSDL file.

5 The first DATA step creates a temporary SAS data set named WORK.ADD that contains the input parameters to be submitted to the Web service. The data set must be created as described in the PROC CONTENTS output for WSDL.Add, which means that WORK.ADD contains the numeric variables X and Y. The value for X and the value for Y, which is 1 for each variable, will be submitted to the Web service.

6 The second DATA step creates a temporary SAS data set named WORK.RESP to contain the results from the Web service.

- **a** The SET statement specifies the libref WSDL from the LIBNAME statement, the member name of the Web service results SAS data set (WSDL.AddResponse), and the PARMS= data set option. The PARMS= data set option specifies the name of the SAS data set to be submitted to the Web service (WORK.ADD).

  The SET statement causes the engine to read the input parameters that are contained in the SAS data set that is specified in the PARMS= data set option, which is WORK.ADD. In addition, the engine invokes the Web service.
- **b** The engine retrieves the response from the Web service, and imports the results into the WORK.RESP data set.

7 PROC CONTENTS describes the contents of WORK.RESP, which contains one variable named AddResult.

8 PROC PRINT prints the observations in WORK.RESP, which contains the Web service Add Result of 2.

Output 3.12 PROC DATASETS Output

```
                                   Directory

                    Libref         WSDL
                    Engine         XML92
                    Physical Name  http://t2824/AddService.asmx?WSDL
                    XMLType        Web Services Description Language
                    Version        .


                      Member  Obs, Entries
       #  Name         Type     or Indexes   Vars  Label

       1  Add          DATA         0         2    WebService Add - Input parameters
       2  AddResponse  DATA         0         1    WebService Add - Return results
```

Output 3.13 PROC CONTENTS Output for WSDL.Add

```
                                   The SAS System                                                         1

                               The CONTENTS Procedure

Data Set Name        WSDL.Add                                Observations          0
Member Type          DATA                                    Variables             2
Engine               XML92                                   Indexes               0
Created              .                                       Observation Length    0
Last Modified        .                                       Deleted Observations  0
Protection                                                   Compressed            NO
Data Set Type                                                Sorted                NO
Label                WebService Add - Input parameters
Data Representation  Default
Encoding             Default


                          Engine/Host Dependent Information

                                   Table Name  Add
                                   Table Path  .


                             Variables in Creation Order

                  #    Variable    Type    Len    Label

                  1    x           Num       8    datatype=integer
                  2    y           Num       8    datatype=integer
```

Output 3.14 PROC CONTENTS Output for WSDL.AddResponse

```
                                   The SAS System                                                         2

                               The CONTENTS Procedure

Data Set Name        WSDL.AddResponse                        Observations          0
Member Type          DATA                                    Variables             1
Engine               XML92                                   Indexes               0
Created              .                                       Observation Length    0
Last Modified        .                                       Deleted Observations  0
Protection                                                   Compressed            NO
Data Set Type                                                Sorted                NO
Label                WebService Add - Return results
Data Representation  Default
Encoding             Default


                          Engine/Host Dependent Information

                               Table Name  AddResponse
                               Table Path  .


                             Variables in Creation Order

                  #    Variable     Type    Len    Label

                  1    AddResult    Num       8    datatype=integer
```

Output 3.15 PROC CONTENTS Output for WORK.RESP

```
                                       The SAS System                                                        3

                                   The CONTENTS Procedure

      Data Set Name        WORK.RESP                               Observations          1
      Member Type          DATA                                    Variables             1
      Engine               V9                                      Indexes               0
      Created              Friday, September 11, 2009 03:05:39 PM Observation Length    8
      Last Modified        Friday, September 11, 2009 03:05:39 PM Deleted Observations  0
      Protection                                                   Compressed            NO
      Data Set Type                                                Sorted                NO
      Label
      Data Representation  WINDOWS_32
      Encoding             wlatin1  Western (Windows)


                              Engine/Host Dependent Information

Data Set Page Size          4096
Number of Data Set Pages    1
First Data Page             1
Max Obs per Page            501
Obs in First Data Page      1
Number of Data Set Repairs  0
Filename                    C:\DOCUME~1\xxxxxx\LOCALS~1\Temp\SAS
                            Temporary Files\_TD316\resp.sas7bdat
Release Created             9.0202M3
Host Created                XP_PRO


                                 Variables in Creation Order

                        #    Variable     Type    Len    Label

                        1    AddResult    Num       8    datatype=integer
```

Output 3.16 PROC PRINT Output for WORK.RESP

```
                                       The SAS System                                            4

                                                 Add
                                        Obs    Result

                                         1        2
```

CHAPTER 4

Exporting XML Documents Using an XMLMap

Why Use an XMLMap when Exporting?

To export an XML document that was imported using an XMLMap, you can use the XMLMap. The XMLMap syntax tells the XML engine how to map the SAS data back into the specific XML document structure.

To export an XML document using an XMLMap, specify the XML engine nickname XML92 in the LIBNAME statement and use the XMLMAP= option to specify the file.

Using an XMLMap to Export an XML Document with a Hierarchical Structure

This example explains how to use an existing XMLMap to tell the XML engine how to map a SAS data set back into the specific XML document structure. The XMLMap was used to import the SAS data set NHL.TEAMS in the section "Using an XMLMap to Import an XML Document as One SAS Data Set" on page 48.

First, here is the SAS data set named NHL.TEAMS to be exported:

Output 4.1 PROC PRINT of Data Set NHL.TEAMS

```
                      The SAS System                             1

 Obs    NAME          ABBREV    CONFERENCE    DIVISION

   1    Thrashers      ATL       Eastern      Southeast
   2    Hurricanes     CAR       Eastern      Southeast
   3    Panthers       FLA       Eastern      Southeast
   4    Lightning      TB        Eastern      Southeast
   5    Capitals       WSH       Eastern      Southeast
   6    Stars          DAL       Western      Pacific
   7    Kings          LA        Western      Pacific
   8    Ducks          ANA       Western      Pacific
   9    Coyotes        PHX       Western      Pacific
  10    Sharks         SJ        Western      Pacific
```

If the data were exported without an XMLMap, the structure of the resulting XML document would be rectangular and consist of a TEAMS element for each observation in the SAS data set. For example:

```
<?xml version="1.0" encoding="windows-1252" ?>
<LIBRARY type="GENERIC" version="9.2">
  <teams>
      <NAME>Thrashers</NAME>
      <ABBREV>ATL</ABBREV>
      <CONFERENCE>Eastern</CONFERENCE>
      <DIVISION>Southeast</DIVISION>
  </teams>
  <teams>
      <NAME>Hurricanes</NAME>
      <ABBREV>CAR</ABBREV>
      <CONFERENCE>Eastern</CONFERENCE>
      <DIVISION>Southeast</DIVISION>
  </teams>
.
.
.
```

To export the SAS data set as an XML document that structures data hierarchically by division within each conference, an XMLMap is required. The only change to the existing XMLMap is to include the OUTPUT element. Notations in the XMLMap syntax are explained:

```
<?xml version="1.0" ?>
<SXLEMAP version="1.9"> ❶
    <OUTPUT> ❷
       <HEADING> ❷
          <ATTRIBUTE  name="description" ❸
                      value="Teams of the National Hockey League" />
       </HEADING>
       <TABLEREF name="TEAMS" /> ❹
    </OUTPUT>

  <TABLE name="TEAMS">

        <TABLE-PATH>/NHL/CONFERENCE/DIVISION/TEAM</TABLE-PATH>

        <COLUMN name="NAME">
          <PATH>/NHL/CONFERENCE/DIVISION/TEAM/@name</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>30</LENGTH>
         </COLUMN>

        <COLUMN name="ABBREV">
          <PATH>/NHL/CONFERENCE/DIVISION/TEAM/@abbrev</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>3</LENGTH>
         </COLUMN>
       <COLUMN name="CONFERENCE" retain="YES">
          <PATH>/NHL/CONFERENCE</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

       <COLUMN name="DIVISION" retain="YES">
           <PATH>/NHL/CONFERENCE/DIVISION</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>
    </TABLE>
</SXLEMAP>
```

1 To use an XMLMap to export the SAS data set as an XML document, you must specify 1.9 as the XMLMap version number.

2 To use an XMLMap to export the SAS data set as an XML document, you must include the OUTPUT element in the XMLMap. The OUTPUT element contains one or more HEADING elements and one TABLEREF element.

3 The ATTRIBUTE element, which defines additional file attribute information, specifies a name and description for the exported XML document.

4 The TABLEREF element, which references the name of the table to be exported, specifies the table TEAMS.

The following SAS statements export the SAS data set named NHL.TEAMS to an XML document named NHLOUT.XML, using an XMLMap named NHLEXPORT.MAP:

```
libname nhl ’C:\My Documents\myfiles’;

filename out ’C:\My Documents\XML\NHLOUT.xml’;

libname out xml92 xmltype=xmlmap  xmlmap=’C:\My Documents\XML\NHLexport.map’;

data out.TEAMS;
   set nhl.teams;
run;
```

Here is the resulting XML document:

```
<?xml version="1.0" encoding="windows-1252" ?>
<!--    SAS XML Libname Engine (SAS92XML)
      SAS XMLMap Generated Output
      Version 9.02.02M3D10192009
      Created 2009-10-20T15:03:53
  -->

<NHL description="Teams of the National Hockey League">
   <CONFERENCE>Eastern
      <DIVISION>Southeast
         <TEAM name="Thrashers" abbrev="ATL" />
         <TEAM name="Hurricanes" abbrev="CAR" />
         <TEAM name="Panthers" abbrev="FLA" />
         <TEAM name="Lightning" abbrev="TB" />
         <TEAM name="Capitals" abbrev="WSH" />
      </DIVISION>
   </CONFERENCE>
   <CONFERENCE>Western
      <DIVISION>Pacific
         <TEAM name="Stars" abbrev="DAL" />
         <TEAM name="Kings" abbrev="LA" />
         <TEAM name="Ducks" abbrev="ANA" />
         <TEAM name="Coyotes" abbrev="PHX" />
         <TEAM name="Sharks" abbrev="SJ" />
      </DIVISION>
   </CONFERENCE>
</NHL>
```

CHAPTER

5

Importing XML Documents Using an XMLMap

Why Use an XMLMap When Importing?

The XML engine imports only XML documents that conform to the markup types supported in the XMLTYPE= option. Attempting to import free-form XML documents that do not conform to the specifications required by the supported markup types will generate errors. To successfully import files that do not conform to the XMLTYPE= markup types, you can create a separate XML document, called an *XMLMap*.

If your XML document does not import successfully, rather than transform the document, you can tell the XML engine how to interpret the XML markup in order to successfully import the XML document. You create an XMLMap that contains specific XMLMap syntax, which is XML markup. The XMLMap syntax tells the XML engine how to interpret the XML markup into a SAS data set or data sets, variables (columns), and observations (rows). See Chapter 8, "XMLMap Files for the XML Engine," on page 99.

After you have created the XMLMap, use the XMLMAP= option in the LIBNAME statement to specify the file.

Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type

What Is the Required Physical Structure?

For an XML document to be successfully imported, the requirements for well-formed XML must translate as follows:

- The root-enclosing element (top-level node) of an XML document is the document container. For SAS, it is like the SAS library.
- The nested elements (repeating element instances) that occur within the container begin with the second-level instance tag.
- The repeating element instances must represent a rectangular organization. For a SAS data set, they determine the observation boundary that becomes a collection of *rows* with a constant set of *columns*.

Here is an example of an XML document that illustrates the physical structure that is required:

```
<?xml version="1.0" encoding="windows-1252" ?>
<LIBRARY> ❶
   <STUDENTS> ❷
      <ID> 0755 </ID>
      <NAME> Brad Martin </NAME>
      <ADDRESS> 1611 Glengreen </ADDRESS>
      <CITY> Huntsville </CITY>
      <STATE> Texas </STATE>
   </STUDENTS>

   <STUDENTS> ❸
      <ID> 1522 </ID>
      <NAME> Zac Harvell </NAME>
      <ADDRESS> 11900 Glenda </ADDRESS>
      <CITY> Houston </CITY>
      <STATE> Texas </STATE>
   </STUDENTS>
.
.  more instances of <STUDENTS>
.
</LIBRARY>
```

When the previous XML document is imported, the following happens:

1. The XML engine recognizes <LIBRARY> as the root-enclosing element.
2. The engine goes to the second-level instance tag, which is <STUDENTS>, translates it as the data set name, and begins scanning the elements that are nested (contained) between the <STUDENTS> start tag and the </STUDENTS> end tag, looking for variables.
3. Because the instance tags <ID>, <NAME>, <ADDRESS>, <CITY>, and <STATE> are contained within the <STUDENTS> start tag and </STUDENTS> end tag, the XML engine interprets them as variables. The individual instance tag names become the data set variable names. The repeating element instances are translated into a collection of rows with a constant set of columns.

These statements result in the following SAS output:

```
libname test xml 'C:\My Documents\test\students.xml';

proc print data=test.students;
run;
```

Output 5.1 PROC PRINT of TEST.STUDENTS

```
                          The SAS System                                1

    Obs    STATE    CITY          ADDRESS           NAME                ID

      1    Texas    Huntsville    1611 Glengreen    Brad Martin        755
      2    Texas    Houston       11900 Glenda      Zac Harvell       1522
           .
           .
           .
```

Why Is a Specific Physical Structure Required?

Well-formed XML is determined by structure, not content. Therefore, while the XML engine can assume that the XML document is valid, well-formed XML, the engine cannot assume that the root element encloses only instances of a single node element, that is, only a single data set. Therefore, the XML engine has to account for the possibility of multiple nodes, that is, multiple SAS data sets.

For example, when the following correctly structured XML document is imported, it is recognized as containing two SAS data sets: HIGHTEMP and LOWTEMP.

```
<?xml version="1.0" encoding="windows-1252" ?>
<CLIMATE> ❶
   <HIGHTEMP> ❷
      <PLACE> Libya </PLACE>
      <DATE> 1922-09-13 </DATE>
      <DEGREE-F> 136 </DEGREE-F>
      <DEGREE-C> 58 </DEGREE-C>
   </HIGHTEMP>
.
.  more instances of <HIGHTEMP>
.
   <LOWTEMP> ❸
      <PLACE> Antarctica </PLACE>
      <DATE> 1983-07-21 </DATE>
      <DEGREE-F> -129 </DEGREE-F>
      <DEGREE-C> -89 </DEGREE-C>
   </LOWTEMP>
.
.  more instances of <LOWTEMP>
.
</CLIMATE>
```

When the previous XML document is imported, the following happens:

1. The XML engine recognizes the first instance tag <CLIMATE> as the root-enclosing element, which is the container for the document.
2. Starting with the second-level instance tag, which is <HIGHTEMP>, the XML engine uses the repeating element instances as a collection of rows with a constant set of columns.
3. When the second-level instance tag changes, the XML engine interprets that change as a different SAS data set.

The result is two SAS data sets: HIGHTEMP and LOWTEMP. Both happen to have the same variables, but of course, different data.

To ensure that an import result is what you expect, use the DATASETS procedure. For example, these SAS statements result in the following:

```
libname climate xml 'C:\My Documents\xml\climate.xml';

proc datasets library=climate;
quit;
```

Output 5.2 PROC DATASETS Output for CLIMATE Library

```
                                       Directory

    Libref         CLIMATE
    Engine         XML
    Physical Name  C:\My Documents\xml\climate.xml
    XMLType        GENERIC
    XMLMap         NO XMLMAP IN EFFECT


                                              Member
                                 #  Name      Type

                                 1  HIGHTEMP  DATA
                                 2  LOWTEMP   DATA
```

Handling XML Documents That Are Not in the Required Physical Structure

If your XML document is not in the required physical structure, you can tell the XML engine how to interpret the XML markup in order to successfully import the document. See Chapter 5, “Importing XML Documents Using an XMLMap,” on page 45.

Using an XMLMap to Import an XML Document as One SAS Data Set

This example explains how to create and use an XMLMap in order to tell the XML engine how to map XML markup to a SAS data set, variables, and observations.

First, here is the XML document NHL.XML to be imported. Although simply constructed and relatively easy for you to read, it does not import successfully because its XML markup is not in the required physical structure:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<NHL>
  <CONFERENCE> Eastern
    <DIVISION> Southeast
      <TEAM name="Thrashers"  abbrev="ATL" />
      <TEAM name="Hurricanes" abbrev="CAR" />
      <TEAM name="Panthers"   abbrev="FLA" />
      <TEAM name="Lightning"  abbrev="TB" />
      <TEAM name="Capitals"   abbrev="WSH" />
   </DIVISION>
 </CONFERENCE>

 <CONFERENCE> Western
   <DIVISION> Pacific
```

```
     <TEAM name="Stars"   abbrev="DAL" />
     <TEAM name="Kings"   abbrev="LA" />
     <TEAM name="Ducks"   abbrev="ANA" />
     <TEAM name="Coyotes" abbrev="PHX" />
     <TEAM name="Sharks"  abbrev="SJ" />
   </DIVISION>
  </CONFERENCE>
</NHL>
```

To successfully import the XML document, an XMLMap is needed. After familiarizing yourself with the data to be imported, you can code the XMLMap syntax so that the data is successfully imported. Here is the XMLMap used to import the XML document, with notations for the data investigation:

```
<?xml version="1.0" ?>
<SXLEMAP version="1.2">
  <TABLE name="TEAMS"> ❶
        <TABLE-PATH syntax="XPATH"> ❷
           /NHL/CONFERENCE/DIVISION/TEAM
         </TABLE-PATH>

        <COLUMN name="NAME"> ❸
          <PATH> ❺
             /NHL/CONFERENCE/DIVISION/TEAM@name
            </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>30</LENGTH>
         </COLUMN>

        <COLUMN name="ABBREV"> ❸
          <PATH> ❺
           /NHL/CONFERENCE/DIVISION/TEAM/@abbrev
            </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>3</LENGTH>
         </COLUMN>

       <COLUMN name="CONFERENCE" retain="YES"> ❹
          <PATH>/NHL/CONFERENCE</PATH> ❺
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

       <COLUMN name="DIVISION" retain="YES"> ❹
           <PATH> ❺
              /NHL/CONFERENCE/DIVISION
            </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>
  </TABLE>
</SXLEMAP>
```

The previous XMLMap syntax defines how to translate the XML markup as explained below, using the following data investigation steps:

1. *Locate and identify distinct tables of information.*

   You want a SAS data set (table) that contains some of the teams of the National Hockey League. Because that is the only information contained in the XML document, you can define a single data set named TEAMS in the XMLMap. (Note that other XML documents might contain more than one table of related information. Importing multiple tables is supported by the XMLMap syntax as shown in "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 51.)

2. *Identify the SAS data set observation boundary, which translates into a collection of rows with a constant set of columns.*

   In the XML document, information about individual teams occurs in a <TEAM> tag located with <CONFERENCE> and <DIVISION> enclosures. You want a new observation generated each time a TEAM element is read.

3. *Collect column definitions for each table.*

   For this XML document, the data content form is mixed. Some data occurs as XML PCDATA (for example, CONFERENCE), and other data is contained in attribute-value pairs (for example, NAME). Data types are all string values. The constructed observation will also include the team NAME and ABBREV. A length of 30 characters is sufficient for the NAME, and three characters is enough for the ABBREV field contents.

4. *Add foreign keys or required external context.*

   You want to include information about the league orientation for the teams. Also, you want to extract CONFERENCE and DIVISION data.

   Note: The retain= attribute in the column definition forces retention of processed data values after an observation is written to the output data set. Because the foreign key fields occur outside the observation boundary (that is, they are more sparsely populated in the hierarchical XML data than in the SAS observation), their values for additional rows need to be retained as they are encountered. △

5. *Define a location path for each variable definition.*

   The PATH element identifies a position in the XML document from which to extract data for each column. Element-parsed character data is treated differently than attribute values. There is no conditional selection criteria involved.

The following SAS statements import the XML document NHL.XML and specify the XMLMap named NHL.MAP. The PRINT procedure verifies that the import is successful:

```
filename NHL 'C:\My Documents\XML\NHL.xml';
filename MAP 'C:\My Documents\XML\NHL.map';
libname NHL xml xmlmap=MAP;

proc print data=NHL.TEAMS;
run;
```

Output 5.3 PROC PRINT of Data Set NHL.TEAMS

```
                   The SAS System                                       1

    Obs    NAME          ABBREV    CONFERENCE    DIVISION

      1    Thrashers      ATL       Eastern      Southeast
      2    Hurricanes     CAR       Eastern      Southeast
      3    Panthers       FLA       Eastern      Southeast
      4    Lightning      TB        Eastern      Southeast
      5    Capitals       WSH       Eastern      Southeast
      6    Stars          DAL       Western      Pacific
      7    Kings          LA        Western      Pacific
      8    Ducks          ANA       Western      Pacific
      9    Coyotes        PHX       Western      Pacific
     10    Sharks         SJ        Western      Pacific
```

Using an XMLMap to Import an XML Document as Multiple SAS Data Sets

This example explains how to create and use an XMLMap in order to define how to map XML markup into two SAS data sets. The example uses the XML document RSS.XML, which does not import successfully because its XML markup is incorrectly structured for the XML engine to translate successfully.

Note: The XML document RSS.XML uses the XML format RSS (Rich Site Summary), which was designed by Netscape originally for exchange of content within the My Netscape Network (MNN) community. The RSS format has been widely adopted for sharing headlines and other Web content and is a good example of XML as a transmission format. △

First, here is the XML document RSS.XML to be imported:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
   <channel>
      <title>WriteTheWeb</title>
   <link>http://writetheweb.com</link>
      <description>News for web users that write back</description>
      <language>en-us</language>
      <copyright>Copyright 2000, WriteTheWeb team.</copyright>
      <managingEditor>editor@writetheweb.com</managingEditor>
      <webMaster>webmaster@writetheweb.com</webMaster>
      <image>
         <title>WriteTheWeb</title>
   <url>http://writetheweb.com/images/mynetscape88.gif</url>
   <link>http://writetheweb.com</link>
   <width>88</width>
   <height>31</height>
   <description>News for web users that write back</description>
   </image>
      <item>
         <title>Giving the world a pluggable Gnutella</title>
   <link>http://writetheweb.com/read.php?item=24</link>
   <description>WorldOS is a framework on which to build programs that work
like Freenet or Gnutella -allowing distributed applications using
```

```
peer-to-peer routing.</description>
      </item>
      <item>
         <title>Syndication discussions hot up</title>
   <link>http://writetheweb.com/read.php?item=23</link>
   <description>After a period of dormancy, the Syndication mailing list
has become active again, with contributions from leaders in traditional media
and Web syndication.</description>
      </item>
   <item>
         <title>Personal web server integrates file sharing and messaging
   </title>
   <link>http://writetheweb.com/read.php?item=22</link>
   <description>The Magi Project is an innovative project to create a
combined personal web server and messaging system that enables the sharing
and synchronization of information across desktop, laptop and palmtop devices.
    </description>
      </item>
      <item>
         <title>Syndication and Metadata</title>
   <link>http://writetheweb.com/read.php?item=21</link>
   <description>RSS is probably the best known metadata format around.
RDF is probably one of the least understood. In this essay, published on my
O&apos;Reilly Network weblog, I argue that the next generation of RSS
should be based on RDF.</description>
      </item>
      <item>
         <title>UK bloggers get organized</title>
   <link>http://writetheweb.com/read.php?item=20</link>
   <description>Looks like the weblogs scene is gathering pace beyond
the shores of the US. There&apos;s now a UK-specific page on weblogs.com,
and a mailing list at egroups.</description>
      </item>
      <item>
         <title>Yournamehere.com more important than anything</title>
   <link>http://writetheweb.com/read.php?item=19</link>
   <description>Whatever you&apos;re publishing on the web, your site
name is the most valuable asset you have, according to Carl Steadman.
     </description>
      </item>
   </channel>
</rss>
```

The XML document can be successfully imported by creating an XMLMap that defines how to map the XML markup. The following is the XMLMap named RSS.MAP, which contains the syntax that is needed to successfully import RSS.XML. The syntax tells the XML engine how to interpret the XML markup as explained in the subsequent descriptions. The contents of RSS.XML results in two SAS data sets: CHANNEL to contain content information and ITEMS to contain the individual news stories.

```
<?xml version="1.0" ?>

<SXLEMAP version="1.2"> ❶

   <!-- TABLE (CHANNEL) -->
```

```
<!-- top level channel content description (TOC) -->
<TABLE name="CHANNEL"> ❷
   <TABLE-PATH syntax="XPATH"> /rss/channel </TABLE-PATH> ❸
   <TABLE-END-PATH syntax="XPATH" beginend="BEGIN">
      /rss/channel/item </TABLE-END-PATH> ❹

   <!-- title -->
   <COLUMN name="title"> ❺
      <PATH> /rss/channel/title </PATH>
      <TYPE> character </TYPE>
      <DATATYPE> string </DATATYPE>
      <LENGTH> 200 </LENGTH>
   </COLUMN>

   <!-- link -->
   <COLUMN name="link"> ❻
      <PATH> /rss/channel/link </PATH>
      <TYPE> character </TYPE>
      <DATATYPE> string </DATATYPE>
      <LENGTH> 200 </LENGTH>
      <DESCRIPTION> Story link </DESCRIPTION>
   </COLUMN>

   <!-- description -->
   <COLUMN name="description">
      <PATH> /rss/channel/description </PATH>
      <TYPE> character </TYPE>
      <DATATYPE> string </DATATYPE>
      <LENGTH> 1024 </LENGTH>
   </COLUMN>

   <!-- language -->
   <COLUMN name="language">
      <PATH> /rss/channel/language </PATH>
      <TYPE> character </TYPE>
      <DATATYPE> string </DATATYPE>
      <LENGTH> 8 </LENGTH>
   </COLUMN>

   <!-- version -->
   <COLUMN name="version"> ❼
      <PATH> /rss@version </PATH>
      <TYPE> character </TYPE>
      <DATATYPE> string </DATATYPE>
      <LENGTH> 8 </LENGTH>
   </COLUMN>
</TABLE>


<!-- TABLE (ITEMS) -->
<!-- individual news stories -->
<TABLE name="ITEMS"> ❽
   <TABLE-PATH syntax="XPATH"> /rss/channel/item </TABLE-PATH>
   <TABLE-DESCRIPTION> Individual news stories </TABLE-DESCRIPTION>
```

```
      <!-- title -->
      <COLUMN name="title"> ❾
         <PATH> /rss/channel/item/title </PATH>
         <TYPE> character </TYPE>
         <DATATYPE> string </DATATYPE>
         <LENGTH> 200 </LENGTH>
      </COLUMN>

      <!-- link -->
      <!-- link is renamed to url, assigned a label and max length -->
      <COLUMN name="URL"> ❿
         <PATH> /rss/channel/item/link </PATH>
         <TYPE> character </TYPE>
         <DATATYPE> string </DATATYPE>
         <LENGTH> 200 </LENGTH>
         <DESCRIPTION> Story link </DESCRIPTION>
      </COLUMN>

      <!-- description -->
      <COLUMN name="description">
         <PATH> /rss/channel/item/description </PATH>
         <TYPE> character </TYPE>
         <DATATYPE> string </DATATYPE>
         <LENGTH> 1024 </LENGTH>
      </COLUMN>
   </TABLE>

</SXLEMAP>
```

The previous XMLMap defines how to translate the XML markup as explained below:

1. Root-enclosing element for SAS data set definitions.
2. Element for the CHANNEL data set definition.
3. Element specifying the location path that defines where in the XML document to collect variables for the CHANNEL data set.
4. Element specifying the location path that specifies when to stop processing data for the CHANNEL data set.
5. Element containing the attributes for the TITLE variable in the CHANNEL data set. The XPath construction specifies where to find the current tag and to access data from the named element.
6. Subsequent COLUMN elements define the variables LINK, DESCRIPTION, and LANGUAGE for the CHANNEL data set.
7. Element containing the attributes for the last variable in the CHANNEL data set, which is VERSION. This XPath construction specifies where to find the current tag and uses the attribute form to access data from the named attribute.
8. Element for the ITEMS data set definition.
9. Element containing the attributes for the TITLE variable in the ITEMS data set.
10. Subsequent COLUMN elements define other variables for the ITEMS data set, which are URL and DESCRIPTION.

The following SAS statements import the XML document RSS.XML and specify the XMLMap named RSS.MAP. The DATASETS procedure then verifies the import results:

```
filename rss 'C:\My Documents\xml\rss.xml';
filename map 'C:\My Documents\xml\rss.map';

libname rss xml xmlmap=map access=readonly;

proc datasets library=rss;
run;
quit;
```

Output 5.4 PROC DATASETS Output for RSS Library Showing Two Data Sets

```
       Directory

Libref         RSS
Engine         XML
Access         READONLY
Physical Name  RSS
XMLType        GENERIC
XMLMap         MAP


              Member
  #  Name     Type

  1  CHANNEL  DATA
  2  ITEMS    DATA
```

Importing Hierarchical Data as Related Data Sets

XML documents often contain hierarchical data in that the data is structured into different levels like a company organization chart. Hierarchical structures are one-to-many relationships, with top items having one or more items below it (for example, customer to orders).

This example explains how to define an XMLMap in order to import an XML document as two data sets that have related information.

First, here is the XML document Pharmacy.XML. The file contains hierarchical data with related entities in the form of individual customers and their prescriptions. Each customer can have one or multiple prescriptions. Notice that PRESCRIPTION elements are nested within each <PERSON> start tag and </PERSON> end tag:

```
<?xml version="1.0" ?>
<PHARMACY>
  <PERSON>
    <NAME>Brad Martin</NAME>
    <STREET>11900 Glenda Court</STREET>
    <CITY>Austin</CITY>
   <PRESCRIPTION>
    <NUMBER>1234</NUMBER>
    <DRUG>Tetracycline</DRUG>
  </PRESCRIPTION>
  <PRESCRIPTION>
    <NUMBER>1245</NUMBER>
    <DRUG>Lomotil</DRUG>
  </PRESCRIPTION>
  </PERSON>
 <PERSON>
    <NAME>Jim Spano</NAME>
    <STREET>1611 Glengreen</STREET>
    <CITY>Austin</CITY>
  <PRESCRIPTION>
    <NUMBER>1268</NUMBER>
    <DRUG>Nexium</DRUG>
  </PRESCRIPTION>
 </PERSON>
</PHARMACY>
```

To import separate data sets, one describing the customers and the other containing prescription information, a relation between each customer and associated prescriptions must be designated in order to know which prescriptions belong to each customer.

An XMLMap defines how to translate the XML markup into two SAS data sets. The customer table imports the name and address of each customer, and the prescription table imports the customer's name, prescription number, and drug. Notations in the XMLMap syntax are explained below.

Note: The XMLMap was generated by using SAS XML Mapper. △

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ############################################################ -->
<!-- 2003-04-08T15:03:16 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 9.1.10.20030407.1378 -->
<!-- ############################################################ -->

<SXLEMAP version="1.2" name="SXLEMAP"> ❶

  <!-- ############################################################ -->
  <TABLE name="PERSON"> ❷
    <TABLE-PATH syntax="XPATH">/PHARMACY/PERSON</TABLE-PATH>

    <COLUMN name="NAME"> ❸
      <PATH  syntax="XPATH">/PHARMACY/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>11</LENGTH>
    </COLUMN>

    <COLUMN name="STREET"> ❸
      <PATH  syntax="XPATH">/PHARMACY/PERSON/STREET</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>18</LENGTH>
    </COLUMN>

    <COLUMN name="CITY"> ❸
      <PATH  syntax="XPATH">/PHARMACY/PERSON/CITY</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>6</LENGTH>
    </COLUMN>

  </TABLE>

  <!-- ############################################################ -->
  <TABLE name="PRESCRIPTION"> ❹
    <TABLE-PATH syntax="XPATH">/PHARMACY/PERSON/PRESCRIPTION</TABLE-PATH>

    <COLUMN name="NAME" retain="YES"> ❺
      <PATH  syntax="XPATH">/PHARMACY/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>11</LENGTH>
    </COLUMN>

    <COLUMN name="NUMBER"> ❻
      <PATH  syntax="XPATH">/PHARMACY/PERSON/PRESCRIPTION/NUMBER</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="DRUG"> ❻
      <PATH  syntax="XPATH">/PHARMACY/PERSON/PRESCRIPTION/DRUG</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>12</LENGTH>
    </COLUMN>

  </TABLE>

</SXLEMAP>
```

1 SXLEMAP is the root-enclosing element for the two SAS data set definitions.

2 First TABLE element defines the Person data set.

3 COLUMN elements contain the attributes for the Name, Street, and City variables in the Person data set.

4 Second TABLE element defines the Prescription data set.

5 COLUMN element contains the attributes for the Name variable in the Prescription data set. Specifying the **`retain="YES"`** attribute causes the name to be held for each observation until it is replaced by a different value. (The retain= attribute is like the SAS DATA step RETAIN statement, which causes a variable to retain its value from one iteration of the DATA step to the next.)

6 COLUMN elements contain the attributes for the Number and Drug variables in the Prescription data set.

The following SAS statements import the XML document and specify the XMLMap:

```
filename pharm 'c:\My Documents\XML\Pharmacy.xml';
filename map 'c:\My Documents\XML\Pharmacy.map';
libname pharm xml xmlmap=map;
```

The DATASETS procedure verifies that SAS interprets the XML document Pharmacy.XML as two SAS data sets: PHARM.PERSON and PHARM.PRESCRIPTION.

```
proc datasets library=pharm;
```

Output 5.5 PROC DATASETS Output for the PHARM Data Library

```
5    proc datasets library=pharm;
                                            Directory

                                      Libref         PHARM
                                      Engine         XML
                                      Physical Name  PHARM
                                      XMLType        GENERIC
                                      XMLMap         MAP


                                                       Member
                                      #  Name          Type

                                      1  PERSON        DATA
                                      2  PRESCRIPTION  DATA
```

Here is PROC PRINT output for both of the imported SAS data sets.

Output 5.6 PROC PRINT Output for PHARM.PERSON

```
                          The SAS System                         1

    Obs    NAME              STREET                 CITY

      1    Brad Martin       11900 Glenda Court     Austin
      2    Jim Spano         1611 Glengreen         Austin
```

Output 5.7 PROC PRINT Output for PHARM.PRESCRIPTION

```
                          The SAS System                         2

            Obs    NAME               NUMBER    DRUG

              1    Brad Martin         1234     Tetracycline
              2    Brad Martin         1245     Lomotil
              3    Jim Spano           1268     Nexium
```

Including a Key Field with Generated Numeric Keys

This example imports the XML document Pharmacy.XML, which contains hierarchical data and is used in the example “Importing Hierarchical Data as Related Data Sets” on page 55. This example continues with the XMLMap by adding a key field with generated numeric key values in order to provide a relationship between the two data sets. (A key field holds unique data in order to identify that record from the other records. For example, account number, product code, and customer name are typical key fields.)

To generate key field values, use the class="ORDINAL" attribute in the COLUMN element in order to create a counter variable. A counter variable keeps track of the number of times the location path, which is specified by the INCREMENT-PATH element, is encountered. The counter variable increments its count by 1 each time the location path is matched. (The counter variable is similar to the _N_ automatic variable in DATA step processing in that it counts the number of observations being read into a SAS data set.)

Note: When using a counter variable to create a key field for related data sets, you must specify the same location paths for both TABLE elements. Otherwise, the results will not match. Each table must have the same generated key for like-named data elements. △

The following XMLMap imports Pharmacy.XML document as two SAS data sets that have related information and also creates a key field that holds generated numeric key values:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!-- ############################################################ -->
<!-- 2003-04-15T10:55:43 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 9.1.10.20030413.1400 -->
<!-- ############################################################ -->

<SXLEMAP version="1.2" name="SXLEMAP">

  <!-- ############################################################ -->
  <TABLE name="PERSON">
    <TABLE-PATH syntax="XPATH">/PHARMACY/PERSON</TABLE-PATH> ❶

    <COLUMN name="KEY" retain="YES" class="ORDINAL"> ❷
      <INCREMENT-PATH syntax="XPATH">/PHARMACY/PERSON</INCREMENT-PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>integer</DATATYPE>
      <FORMAT width="3">Z</FORMAT>
    </COLUMN>

    <COLUMN name="NAME">
      <PATH  syntax="XPATH">/PHARMACY/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>11</LENGTH>
    </COLUMN>

    <COLUMN name="STREET">
      <PATH  syntax="XPATH">/PHARMACY/PERSON/STREET</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>18</LENGTH>
    </COLUMN>

    <COLUMN name="CITY">
      <PATH  syntax="XPATH">/PHARMACY/PERSON/CITY</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>6</LENGTH>
    </COLUMN>
  </TABLE>

  <!-- ############################################################ -->
  <TABLE name="PRESCRIPTION">
    <TABLE-PATH syntax="XPATH">/PHARMACY/PERSON/PRESCRIPTION</TABLE-PATH> ❸

    <COLUMN name="KEY" retain="YES" class="ORDINAL"> ❹
      <INCREMENT-PATH syntax="XPATH">/PHARMACY/PERSON</INCREMENT-PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>integer</DATATYPE>
      <FORMAT width="3">Z</FORMAT>
    </COLUMN>

    <COLUMN name="NUMBER">
      <PATH  syntax="XPATH">/PHARMACY/PERSON/PRESCRIPTION/NUMBER</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="DRUG">
      <PATH  syntax="XPATH">/PHARMACY/PERSON/PRESCRIPTION/DRUG</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>12</LENGTH>
    </COLUMN>

  </TABLE>

</SXLEMAP>
```

The following explains the XMLMap syntax that generates the key fields:

1 In the TABLE element that defines the Person data set, the TABLE-PATH element identifies the observation boundary for the data set. The location path generates a new observation each time a PERSON element is read.

2 For the Person data set, the COLUMN element for the Key variable contains the class="ORDINAL" attribute as well as the INCREMENT-PATH element. The XML engine follows this process in order to generate the key field values for the Person data set:

   a When the XML engine encounters the <PERSON> start tag, it reads the value into the input buffer, and then increments the value for the Key variable by 1.
   b The XML engine continues reading values into the input buffer until it encounters the </PERSON> end tag, at which time it writes the completed input buffer to the SAS data set as one observation.
   c The process is repeated for each <PERSON> start tag (from INCREMENT-PATH) and </PERSON> end tag (from TABLE-PATH) sequence.
   d The result is four variables and two observations.

3 In the TABLE element that defines the Prescription data set, the TABLE-PATH element identifies the observation boundary for the data set. The location path generates a new observation each time a PRESCRIPTION element is read.

4 For the Prescription data set, the COLUMN element for the Key variable contains the class="ORDINAL" attribute as well as the INCREMENT-PATH element.

The XML engine follows this process in order to generate the key field values for the Prescription data set:

a When the XML engine encounters the <PERSON> start tag, it reads the value into the input buffer, and then increments the value for the Key variable by 1.

b The XML engine continues reading values into the input buffer until it encounters the </PRESCRIPTION> end tag, at which time it writes the completed input buffer to the SAS data set as one observation.

Note: Because the location paths for the counter variables must be the same for both TABLE elements, the behavior of the XML engine for the Prescription table Key variable is the same as the Person table Key variable. While the XML engine tracks the occurrence of a PERSON tag as a key for both counter variables, the observations are derived from different TABLE-PATH locations. △

c The process is repeated for each <PERSON> start tag (from INCREMENT-PATH) and </PRESCRIPTION> end tag (from TABLE-PATH) sequence.

d The result is three variables and three observations.

The following SAS statements import the XML document:

```
filename pharm 'c:\My Documents\XML\Pharmacy.xml';
filename map 'c:\My Documents\XML\PharmacyOrdinal.map';
libname pharm xml92 xmlmap=map;
```

Here is PROC PRINT output for both of the imported SAS data sets with a numeric key:

Output 5.8 PROC PRINT Output for PHARM.PERSON

```
                     The SAS System                                    1

      Obs         KEY    NAME           STREET                CITY

        1         001    Brad Martin    11900 Glenda Court    Austin
        2         002    Jim Spano      1611 Glengreen        Austin
```

Output 5.9 PROC PRINT Output for PHARM.PRESCRIPTION

```
                    The SAS System                                     2

               Obs         KEY      NUMBER    DRUG

                 1         001        1234    Tetracycline
                 2         001        1245    Lomotil
                 3         002        1268    Nexium
```

Determining the Observation Boundary to Avoid Concatenated Data

This example imports an XML document that illustrates how to determine the observation boundary so that the result is separate observations and not concatenated data.

The observation boundary translates into a collection of rows with a constant set of columns. Using an XMLMap, you determine the observation boundary with the TABLE-PATH element by specifying a location path. The end tag for the location path determines when data is written to the SAS data set as an observation.

Identifying the observation boundary can be tricky due to sequences of start tag and end-tag pairing. If you do not identify the appropriate observation boundary, the result could be a concatenated data string instead of separate observations. This example illustrates pairing situations that can cause unwanted results.

For the following XML document, an XMLMap is necessary in order to import the file successfully. Without an XMLMap, the XML engine would import a data set named FORD with columns ROW0, MODEL0, YEAR0, ROW1, MODEL1, YEAR1, and so on.

```
<?xml version="1.0" ?>
<VEHICLES>
  <FORD>
    <ROW>
      <Model>Mustang</Model>
      <Year>1965</Year>
    </ROW>
    <ROW>
      <Model>Explorer</Model>
      <Year>1982</Year>
    </ROW>
    <ROW>
      <Model>Taurus</Model>
      <Year>1998</Year>
    </ROW>
    <ROW>
      <Model>F150</Model>
      <Year>2000</Year>
    </ROW>
  </FORD>
</VEHICLES>
```

Looking at the above XML document, there are three sequences of element start tags and end tags: VEHICLES, FORD, and ROW. If you specify the following table location path and column locations paths, the XML engine processes the XML document as follows:

```
<TABLE-PATH syntax="XPATH"> /VEHICLES/FORD </TABLE-PATH>
  <PATH syntax="XPATH"> /VEHICLES/FORD/ROW/Model </PATH>
  <PATH syntax="XPATH"> /VEHICLES/FORD/ROW/Year </PATH>
```

1 The XML engine reads the XML markup until it encounters the <FORD> start tag, because FORD is the last element specified in the table location path.

2 The XML engine clears the input buffer and scans subsequent elements for variables based on the column location paths. As a value for each variable is

encountered, it is read into the input buffer. For example, after reading the first ROW element, the input buffer contains the values **`Mustang`** and **`1965`**.

3 The XML engine continues reading values into the input buffer until it encounters the </FORD> end tag, at which time it writes the completed input buffer to the SAS data set as an observation.

4 The end result is one observation, which is not what you want.

Here is PROC PRINT output showing the concatenated observation. (The data in the observation is truncated due to the LENGTH element.)

Output 5.10 PROC PRINT Output Showing Unacceptable FORD Data Set

```
                                  The SAS System                              1

                           Model                   Year

                           Mustang Explorer Tau    1965
```

To get separate observations, you must change the table location path so that the XML engine writes separate observations to the SAS data set. Here are the correct location paths and the process that the engine would follow:

```
<TABLE-PATH syntax="XPATH"> /VEHICLES/FORD/ROW </TABLE-PATH>
  <PATH syntax="XPATH"> /VEHICLES/FORD/ROW/Model </PATH>
  <PATH syntax="XPATH"> /VEHICLES/FORD/ROW/Year </PATH>
```

1 The XML engine reads the XML markup until it encounters the <ROW> start tag, because ROW is the last element specified in the table location path.

2 The XML engine clears the input buffer and scans subsequent elements for variables based on the column location paths. As a value for each variable is encountered, it is read into the input buffer.

3 The XML engine continues reading values into the input buffer until it encounters the </ROW> end tag, at which time it writes the completed input buffer to the SAS data set as an observation. That is, one observation is written to the SAS data set that contains the values **`Mustang`** and **`1965`**.

4 The process is repeated for each <ROW> start-tag and </ROW> end-tag sequence.

5 The result is four observations.

Here is the complete XMLMap syntax:

```
<?xml version="1.0" ?>
<SXLEMAP version="1.2" name="path" description="XMLMap for path">
  <TABLE name="FORD">
    <TABLE-PATH syntax="XPATH"> /VEHICLES/FORD/ROW </TABLE-PATH>
    <COLUMN name="Model">
      <DATATYPE> string </DATATYPE>
      <LENGTH> 20 </LENGTH>
      <TYPE> character </TYPE>
      <PATH syntax="XPATH"> /VEHICLES/FORD/ROW/Model </PATH>
    </COLUMN>
    <COLUMN name="Year">
      <DATATYPE> string </DATATYPE>
      <LENGTH> 4 </LENGTH>
      <TYPE> character </TYPE>
      <PATH syntax="XPATH"> /VEHICLES/FORD/ROW/Year </PATH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

The following SAS statements import the XML document and specify the XMLMap. The PRINT procedure verifies the results.

```
filename PATH ’c:\My Documents\XML\path.xml’;
filename MAP ’c:\My Documents\XML\path.map’;
libname PATH xml xmlmap=MAP;

proc print data=PATH.FORD noobs;
run;
```

Output 5.11 PROC PRINT Output Showing Desired FORD Data Set

```
                              The SAS System                              1

                       Model                 Year

                       Mustang               1965
                       Explorer              1982
                       Taurus                1998
                       F150                  2000
```

Determining the Observation Boundary to Select the Best Columns

This example imports an XML document that illustrates how to determine the observation boundary so that the result is the best collection of columns.

The observation boundary translates into a collection of rows with a constant set of columns. Using an XMLMap, you determine the observation boundary with the TABLE-PATH element by specifying a location path.

In the following XML document, PUBLICATION appears to be a possible element to use as the observation boundary, which would result in these columns: TITLE, ACQUIRED, TOPIC. However, the TOPIC element occurs arbitrarily within a single PUBLICATION container, so the result would be a set of columns with TOPIC occurring more than once. Therefore, the TOPIC element is the better choice to use as the observation boundary in order to result in these columns: TITLE, ACQUIRED, TOPIC, MAJOR.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<Library>
  <Publication>
    <Title>Developer’s Almanac</Title>
    <Acquired>12-11-2000</Acquired>
    <Topic Major="Y">JAVA</Topic>
  </Publication>
  <Publication>
    <Title>Inside Visual C++</Title>
    <Acquired>06-19-1998</Acquired>
    <Topic>Major="Y">C</Topic>
    <Topic>Reference</Topic>
  </Publication>
  <Publication>
    <Title>Core Servlets</Title>
    <Acquired>05-30-2001</Acquired>
    <Topic Major="Y">JAVA</Topic>
    <Topic>Servlets</Topic>
    <Topic>Reference</Topic>
  </Publication>
</Library>
```

Here is the XMLMap syntax to use in order to import the previous XML document:

```
<?xml version="1.0" ?>
<SXLEMAP version="1.2">
  <TABLE name="Publication">
    <TABLE-PATH syntax="XPATH">
       /Library/Publication/Topic ❶
    </TABLE-PATH>

    <COLUMN name="Title" retain="YES">
      <PATH>
         /Library/Publication/Title
      </PATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>19</LENGTH>
    </COLUMN>

    <COLUMN name="Acquired" retain="YES">
      <PATH>
        /Library/Publication/Acquired
      </PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>FLOAT</DATATYPE>
      <LENGTH>10</LENGTH>
      <FORMAT width="10" >mmddyy</FORMAT> ❷
      <INFORMAT width="10" >mmddyy</INFORMAT>
    </COLUMN>

    <COLUMN name="Topic">
      <PATH>
         /Library/Publication/Topic</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>9</LENGTH>
    </COLUMN>

    <COLUMN name="Major">
      <PATH>
         /Library/Publication/Topic/@Major
      </PATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>1</LENGTH>
      <ENUM> ❸
        <VALUE>Y</VALUE>
        <VALUE>N</VALUE>
      </ENUM>
     <DEFAULT>N</DEFAULT> ❹
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

The previous XMLMap tells the XML engine how to interpret the XML markup as explained below:

1 The TOPIC element determines the location path that defines where in the XML document to collect variables for the SAS data set. An observation is written each time a </TOPIC> end tag is encountered in the XML document.

2 For the ACQUIRED column, the date is constructed using the XMLMap syntax FORMAT element. Elements like FORMAT and INFORMAT are useful for situations where data must be converted for use by SAS. The XML engine also supports user-written formats and informats, which can be used independently of each other.

3 Enumerations are also supported by XMLMap syntax. The ENUM element specifies that the values for the column MAJOR must be either Y or N. Incoming values not contained within the ENUM list are set to MISSING.

4 By default, a missing value is set to MISSING. The DEFAULT element specifies a default value for a missing value, which for this example is specified as N. Note that when the ENUM element is used, a value specified by DEFAULT must be one of the ENUM values in order to be valid.

The following SAS statements import the XML document and specify the XMLMap. The PRINT procedure verifies the results.

```
filename REP 'C:\My Documents\XML\Rep.xml';
filename MAP 'C:\My Documents\XML\Rep.map';
libname REP xml xmlmap=MAP;

proc print data=REP.Publication noobs;
run;
```

Output 5.12 PROC PRINT Output for PUBLICATION Data Set

```
                             The SAS System                              1

          Title                   Acquired    Topic        Major

          Developer's Almanac    12/11/2000    JAVA           Y
          Inside Visual C++      06/19/1998    C              Y
          Inside Visual C++      06/19/1998    Reference      N
          Core Servlets          05/30/2001    JAVA           Y
          Core Servlets          05/30/2001    Servlets       N
          Core Servlets          05/30/2001    Reference      N
```

Using ISO 8601 SAS Informats and Formats to Import Dates

This simple example illustrates importing an XML document that contains date values in both the basic format and the extended format. The XMLMap uses the FORMAT and INFORMAT elements to specify the appropriate SAS format and SAS informat in order to represent the dates according to ISO 8601 standards.

First, here is the XML document:

```
<?xml version="1.0" ?>
<Root>
  <ISODATE>
    <BASIC>20010911</BASIC>
    <EXTENDED>2001-09-11</EXTENDED>
  </ISODATE>
</Root>
```

The following XMLMap imports the XML document using the SAS informats and formats to read and write the date values:

```
<?xml version="1.0" encoding=UTF-8’’?>

<!-- ############################################################ -->
<!-- 2003-08-22T13:54:16 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 9.1.10.20030602.99000 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- Map validation completed successfully. -->
<!-- ############################################################ -->

<SXLEMAP version="1.2" name="ISOdate"
   description="Reading a Basic and Extended format ISO date field">

  <!-- ############################################################ -->
  <TABLE name="ISODATE">
    <TABLE-PATH syntax="XPath">/Root/ISODATE</TABLE-PATH>

    <COLUMN name="BASIC">
      <PATH  syntax="XPath">/Root/ISODATE/BASIC</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>date</DATATYPE>
      <FORMAT width="10">E8601DA</FORMAT> ❶
      <INFORMAT width="8">B8601DA</INFORMAT> ❷
    </COLUMN>

    <COLUMN name="EXTENDED">
      <PATH  syntax="XPath">/Root/ISODATE/EXTENDED</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>date</DATATYPE>
      <FORMAT>E8601DA</FORMAT> ❸
      <INFORMAT width="10">E8601DA</INFORMAT> ❹
    </COLUMN>

  </TABLE>
```

```
</SXLEMAP>
```

The following explains the XMLMap syntax that imports the date values:

1 For the Basic variable, the FORMAT element specifies the E8601DA SAS format, which writes data values in the extended format *YYYY-MM-DD*.

2 For the Basic variable, the INFORMAT element specifies the B8601DA SAS informat, which reads date values into a variable in the basic format *YYYYMMDD*.

   Note: As recommended, when you read values into a variable with a basic format SAS informat, this example writes the values with the corresponding extended format SAS format. △

3 For the Extended variable, the FORMAT element specifies the E8601DA SAS format, which writes data values in the extended format *YYYY-MM-DD*.

4 For the Extended variable, the INFORMAT element specifies the E8601DA SAS informat, which reads date values into a variable in the basic format *YYYY-MM-DD*.

The following SAS statements import the XML document and display PRINT procedure output:

```
filename dates 'c:\My Documents\XML\ISOdate.xml';
filename map 'c:\My Documents\XML\ISOdate.map';
libname dates xml xmlmap=map;

proc print data=dates.isodate;
run;
```

Output 5.13 PRINT Procedure Output for Imported Data Set DATES.ISODATE

```
                    The SAS System                                   1

               Obs    BASIC          EXTENDED

                 1    2001-09-11    2001-09-11
```

Using ISO 8601 SAS Informats and Formats to Import Time Values with a Time Zone

This example illustrates importing an XML document that contains time values in various forms. The XMLMap uses the FORMAT and INFORMAT elements to specify the appropriate SAS formats and SAS informats in order to represent the times appropriately.

First, here is an XML document that contains a variety of time values:

```
<?xml version="1.0" ?>
<Root>
  <TIME>
    <LOCAL>09:00:00</LOCAL>
    <UTC>09:00:00Z</UTC>
    <OFFSET>14:00:00+05:00</OFFSET>
  </TIME>
```

```
</Root>
```

The following XMLMap imports the XML document using the SAS informats and formats to read and write the time values:

```
<?xml version="1.0" encoding="UTF-8"?>
<SXLEMAP version="1.2" name="ISOtime">
      description="Reading time values with and without offsets">
  <!-- ########################################################### -->
  <TABLE name="TIME">
    <TABLE-PATH syntax="XPath">/Root/TIME</TABLE-PATH>

    <COLUMN name="LOCAL">
      <PATH  syntax="XPath">/Root/TIME/LOCAL</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>time</DATATYPE>
      <INFORMAT width="8">E8601TM</INFORMAT> ❶
      <FORMAT width="8">E8601TM</FORMAT>
         </COLUMN>

    <COLUMN name="LOCALZONE">
      <PATH  syntax="XPath">/Root/TIME/LOCAL</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>time</DATATYPE>
      <INFORMAT width="8">E8601TM</INFORMAT> ❷
      <FORMAT width="14">E8601LZ</FORMAT>
    </COLUMN>

    <COLUMN name="UTC">
      <PATH  syntax="XPath">/Root/TIME/UTC</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>time</DATATYPE>
      <INFORMAT width="9">E8601TZ</INFORMAT> ❸
      <FORMAT width="9">E8601TZ</FORMAT>
          </COLUMN>

    <COLUMN name="OFFSET">
      <PATH  syntax="XPath">/Root/TIME/OFFSET</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>time</DATATYPE>
      <INFORMAT width="14">E8601TZ</INFORMAT> ❹
      <FORMAT width="14">E8601TZ</FORMAT>
          </COLUMN>
  </TABLE>

</SXLEMAP>
```

The following explains the XMLMap syntax that imports the time values:

1 For the Local variable, the INFORMAT and FORMAT elements specify the *E8601TM* SAS informat and format, which reads and writes time values in the extended format *hh:mm:ss*[*.fffff*]. Because there is no time zone indicator, the context of the value is local time.

2 For the Localzone variable, which reads the same value as the Local variable, the INFORMAT element specifies the *E8601TM* SAS informat, which reads time values in the extended format *hh:mm:ss*[*.fffff*. Because there is no time zone indicator, the context of the value is local time.

The FORMAT element, however, specifies the *E8601LZ* SAS format, which writes time values in the extended format *hh:mm:ss*[*.fffff*][Z][+|-]*hh:mm*]. The E8601LZ format appends the UTC offset to the value as determined by the local, current SAS session. Using the E8601LZ format enables you to provide a time notation in order to eliminate the ambiguity of local time.

Note: Even with the time notation, it is recommended that you do not mix time-based values. △

3. For the UTC variable, the INFORMAT and FORMAT elements specify the *E8601TZ* SAS informat and format, which reads and writes time values in the extended format *hh:mm:ss*[*.fffff*][Z][+|-]*hh:mm*]. Because there is a time zone indicator, the value is assumed to be expressed in UTC. No adjustment or conversion is made to the value.
4. For the Offset variable, the INFORMAT and FORMAT elements specify the *E8601TZ* SAS informat and format, which reads and writes time values in the extended format *hh:mm:ss*[*.fffff*][Z][+|-]*hh:mm*]. Because there is a time zone offset present, when the time value is read into the variable using the time zone sensitive SAS informat, the value is adjusted to UTC as requested via the time zone indicator, but the time zone context is not stored with the value. When the time value is written using the time zone sensitive SAS format, the value is expressed as UTC with a zero offset value and is not adjusted to or from local time.

The following SAS statements import the XML document and display the PRINT procedure output:

```
filename timzn 'c:\My Documents\XML\Time.xml';
filename map 'c:\My Documents\XML\Time.map';
libname timzn xml xmlmap=map;

proc print data=timzn.time;
run;
```

Output 5.14 PRINT Procedure Output for Imported Data Set TIMZN.TIME

```
                              The SAS System                              1

        Obs    LOCAL       LOCALZONE          UTC          OFFSET

          1    09:00:00    09:00:00-05:00     09:00:00Z    09:00:00+00:00
```

Referencing a Fileref Using the URL Access Method

Using several methods, the XML engine can access an XML document that is referenced by a fileref. When using the URL access method to reference a fileref, you should also specify an XMLMap. Specifying an XMLMap causes the XML engine to process the XML document with a single pass of the file, rather than a double pass, which is what happens when you do not specify an XMLMap.

This example illustrates how to access an XML document by referencing a fileref and using the URL access method:

```
filename NHL url 'http://www.a.com/NHL.xml'; ❶
filename MAP 'C:\My Documents\XML\NHL.map'; ❷

libname NHL xml xmlmap=MAP; ❸
```

```
proc copy indd=NHL outdd=work; ❹
   select NHL;
run;

proc print data=NHL; ❺
run;
```

1 The first FILENAME statement assigns the fileref NHL to the XML document by using the URL access method.

2 The second FILENAME statement assigns the fileref MAP to the physical location of the XMLMap NHL.map.

3 The LIBNAME statement uses the fileref NHL to reference the XML document, specifies the XML engine, and uses the fileref MAP to reference the XMLMap.

4 The COPY procedure reads the XML document, and writes its content as a temporary SAS data set. When using the URL access method, you should include the step to create the SAS data set with either a PROC COPY or a DATA step.

5 The PRINT procedure prints the SAS data set WORK.NHL.

Specifying a Location Path on the PATH Element

The XMLMap PATH element supports several XPath forms to specify a location path. The location path tells the XML engine where in the XML document to locate and access a specific tag for the current variable. In addition, the location path tells the XML engine to perform a function, which is determined by the XPath form, to retrieve the value for the variable.

This example imports an XML document and illustrates each of the supported XPath forms, which include three element forms and two attribute forms.

First, here is the XML document NHL.XML to be imported:

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<NHL>
  <CONFERENCE> Eastern
    <DIVISION> Southeast
      <TEAM founded="1999" abbrev="ATL"> Thrashers </TEAM>
      <TEAM founded="1997" abbrev="CAR"> Hurricanes </TEAM>
      <TEAM founded="1993" abbrev="FLA"> Panthers </TEAM>
      <TEAM founded="1992" abbrev="TB" > Lightning </TEAM>
      <TEAM founded="1974" abbrev="WSH"> Capitals </TEAM>
   </DIVISION>
 </CONFERENCE>
</NHL>
```

Here is the XMLMap used to import the XML document, with notations for each XPath form on the PATH element:

```
<?xml version="1.0" ?>
<SXLEMAP version="1.9">
  <TABLE name="TEAMS">
        <TABLE-PATH>
           /NHL/CONFERENCE/DIVISION/TEAM
         </TABLE-PATH>
        <COLUMN name="ABBREV">
          <PATH>
           /NHL/CONFERENCE/DIVISION/TEAM/@abbrev ❶
            </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>3</LENGTH>
         </COLUMN>

         <COLUMN name="FOUNDED">
           <PATH>
              /NHL/CONFERENCE/DIVISION/TEAM@founded[@abbrev="ATL"] ❷
            </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

       <COLUMN name="CONFERENCE" retain="YES">
          <PATH>
             /NHL/CONFERENCE ❸
           </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

        <COLUMN name="TEAM">
           <PATH>
              /NHL/CONFERENCE/DIVISION/TEAM[@founded="1993"] ❹
            </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

        <COLUMN name="TEAM5">
           <PATH>
              /NHL/CONFERENCE/DIVISION/TEAM[position()=5] ❺
             </PATH>
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>


    </TABLE>
</SXLEMAP>
```

1 The Abbrev variable uses the attribute form that selects values from a specific attribute. The XML engine scans the XML markup until it finds the TEAM element. The XML engine retrieves the value from the abbrev= attribute, which results in each team abbreviation.

2 The Founded variable uses the attribute form that conditionally selects from a specific attribute based on the value of another attribute. The XML engine scans the XML markup until it finds the TEAM element. The XML engine retrieves the value from the founded= attribute where the value of the abbrev= attribute is ATL, which results in the value 1999. The two attributes must be for the same element.

3 The Conference variable uses the element form that selects PCDATA from a named element. The XML engine scans the XML markup until it finds the CONFERENCE element. The XML engine retrieves the value between the <CONFERENCE> start tag and the </CONFERENCE> end tag, which results in the value Eastern.

4 The Team variable uses the element form that conditionally selects PCDATA from a named element. The XML engine scans the XML markup until it finds the TEAM element where the value of the founded= attribute is 1993. The XML engine retrieves the value between the <TEAM> start tag and the </TEAM> end tag, which results in the value Panthers.

5 The Team5 variable uses the element form that conditionally selects PCDATA from a named element based on a specific occurrence of the element. The position function tells the XML engine to scan the XML markup until it finds the fifth occurrence of the TEAM element. The XML engine retrieves the value between the <TEAM> start tag and the </TEAM> end tag, which results in the value Capitals.

The following SAS statements import the XML document NHL.XML and specify the XMLMap named NHL1.MAP. The PRINT procedure shows the resulting variables with selected values:

```
filename NHL 'C:\My Documents\XML\NHL.xml';
filename MAP 'C:\My Documents\XML\NHL1.map';
libname NHL xml xmlmap=MAP;

proc print data=NHL.TEAMS noobs;
run;
```

Output 5.15 PROC PRINT of Data Set NHL.TEAMS

```
                         The SAS System                                    1

 ABBREV    FOUNDED       CONFERENCE    TEAM          TEAM5

  ATL      1999          Eastern
  CAR                    Eastern
  FLA                    Eastern       Panthers
  TB                     Eastern
  WSH                    Eastern                     Capitals
```

CHAPTER

6

Understanding and Using Tagsets for the XML Engine

What Is a Tagset?

A tagset specifies instructions for generating a markup language from your SAS data set. The resulting output contains embedded instructions defining layout and some content. SAS provides tagsets for a variety of markup languages, including the XML markup language.

Creating Customized Tagsets

In addition to using the tagsets provided by SAS, you can modify the SAS tagsets, and you can create your own tagsets. To create a tagset, use the TEMPLATE procedure to define the tagset definition. For information about defining customized tagsets, see the TEMPLATE procedure in *SAS Output Delivery System: User's Guide*.

CAUTION:

Use customized tagsets with caution. If you are unfamiliar with XML output, do not specify different tagsets. If you alter the tagset when exporting an XML document, and then attempt to import the XML document generated by that altered tagset, the XML engine might not be able to translate the XML markup back to SAS proprietary format. △

Exporting an XML Document Using a Customized Tagset

Example Overview

This example defines a customized tagset, and then uses the tagset with the XML engine to export an XML document with customized tags.

Define Customized Tagset using TEMPLATE Procedure

The following TEMPLATE procedure defines a customized tagset named Tagsets.Custom.

You can use the following code as a template to define your own customized tagsets. For example, to create your own customized tagset, only the EmitMeta, EmitRow, and EmitCol events would require minor modifications.

1 The EmitMeta event generates an XML comment that contains a list of the variables from the SAS data set. The event contains an example of iteration for a list variable, which processes all of the variables in the SAS data set. For more information about iteration, see the ITERATE statement in the TEMPLATE procedure DEFINE EVENT statement in *SAS Output Delivery System: User's Guide*.

2 The EmitRow event creates XML output from the three SAS data set observations. The EmitRow event names specific variables to process, which are Name, Height, and Weight.

3 The EmitCol event creates generic-looking XML for each processed variable.

```
proc template;

   /* +------------------------------------------------+
      |                                                |
      +------------------------------------------------+ */

   define tagset tagsets.custom ;
      notes "SAS XML Engine output event model(interface)";

      indent = 3;
      map = ’<>&"’’’;
      mapsub = ’/&lt;/&gt;/&amp;/&quot;/&apos;/’;


   /* +------------------------------------------------+
      |                                                |
      +------------------------------------------------+ */

      define event XMLversion;
         put  ’<?xml version="1.0"’;
         putq ’ encoding=’ ENCODING;
         put  ’ ?>’ CR;
         break;
      end;


      define event XMLcomment;
         put  ’<!-- ’ CR;
         put  '     ' TEXT CR;
         put  '  -->' CR;
         break;
      end;


      define event initialize;
      set   $LIBRARYNAME          'LIBRARY' ;
      set   $TABLENAME            'DATASET' ;
      set   $COLTAG               'column' ;
      set   $META                 'FULL' ;

      eval  $is_engine            1;
      eval  $is_procprint         0;
      eval  $is_OUTBOARD          1;
      end;


   /* +-------------------------------------------------+
      |                                                 |
      +-------------------------------------------------+ */

      define event doc;
      start:
         trigger initialize;
         trigger XMLversion;
         break;
      finish:
         break;
      end;


      define event doc_head;
      start:
         break;
      finish:
         break;
      end;


      define event doc_body;
      start:
         break;
      finish:
         break;
      end;


      define event proc;
      start:
         break / if frame_name ;              /* set by ODS statement use  */
         eval $is_OUTBOARD 0 ;                /* default for non-engine    */
         do / if cmp(XMLCONTROL, "OUTBOARD"); /* only the engine sets this */
            eval $is_OUTBOARD 1 ;
         else ;
            eval $is_OUTBOARD 0 ;
         done ;
         break;
      finish:
         break;
      end;


      define event leaf;
      start:
         /*
          *  PROC PRINT
          *  data set reference is in the value and label fields
          *  and NOT in the output_label field
          */
         eval $is_engine     0; /* NOT ENGINE */
         break / if ^cmp("Print", name);
         eval $is_procprint  1; /* PROC PRINT */
         eval $regex prxparse("/\.(.+)/");
         eval $match prxmatch($regex, value);
         set  $TABLENAME prxposn($regex, 1, value);
         break;
      finish:
         break;
      end;


      define event output;
      start:
         break / if $is_procprint ;
         eval  $is_engine     0;            /* NOT ENGINE   */
         set   $TABLENAME name / if name;   /* TABLE VIEWER */
         break;
      finish:
         break;
      end;


      define event table;
      start:
         unset $col_names;
         unset $col_types;
         unset $col_width;
         eval  $index      1;
         eval  $index_max  0;
         set   $TABLENAME name / if name;           /* LIBNAME ENGINE */
         set   $META XMLMETADATA / if XMLMETADATA ; /* LIBNAME ENGINE */
         set   $SCHEMA XMLSCHEMA / if XMLSCHEMA ;   /* LIBNAME ENGINE */
         break;
      finish:
         break;
      end;
```

```
   define event colspecs;
   start:
      break / if cmp(XMLMETADATA, "NONE");
   finish:
      break / if cmp(XMLMETADATA, "NONE");
   end;


   define event colgroup;
   start:
      break / if cmp(XMLMETADATA, "NONE");
   finish:
      break / if cmp(XMLMETADATA, "NONE");
   end;


/* +-------------------------------------------------+
   |                                                 |
   +-------------------------------------------------+ */

   define event colspec_entry;
   start:
      break / if ^$is_engine and $index eq 1 and cmp(name, "Obs");
      eval  $index_max $index_max+1;
      set $col_names[] name;
      set $col_types[] type;
      set $col_width[] width;
      break;
   finish:
      break;
   end;


   define event table_head;
   start:
      break;
   finish:
      break;
   end;


   define event table_body;
   start:
      trigger EmitMeta ;
      break;
   finish:
      trigger EmitMeta ;
      break;
   end;

/* +-------------------------------------------------+
   |                                                 |
   +-------------------------------------------------+ */
   define event row;
   start:
      break / if !cmp(SECTION, "body");
      break / if cmp(XMLMETADATA, "ONLY");
      eval    $index 1;
      unset   $col_values;
      break;
   finish:
      break / if !cmp(SECTION, "body");
      break / if cmp(XMLMETADATA, "ONLY");
      trigger EmitRow ;
      break;
   end;


   define event data;
   start:
      break / if !cmp(SECTION, "body");
      do / if $is_engine ;
         break / if !cmp(XMLCONTROL, "Data");
      else ;
         break / if !cmp(HTMLCLASS, "Data");
      done ;
      break / if cmp(XMLMETADATA, "ONLY");
      set $name $col_names[$index];
      do / if exists(MISSING);
         eval  $is_MISSING  1;
         eval  $value_MISSING MISSING;
         set   $col_values[$name] " ";
      else ;
         eval  $is_MISSING  0;
         set   $col_values[$name] VALUE;
      done;
      break;
   finish:
      break / if !cmp(SECTION, "body");
      do / if $is_engine ;
         break / if !cmp(XMLCONTROL, "Data");
      else ;
         break / if !cmp(HTMLCLASS, "Data");
      done ;
      break / if cmp(XMLMETADATA, "ONLY");
      set  $name  $col_names[$index];
      eval $index $index+1;
      break;
   end;


/* +-------------------------------------------------+
   |                                                 |
   | at this point, we just take over XML output.    |
   | EmitRow() is triggered each time the data is    |
   |           loaded into the $col_values array.    |
      |                                                |
      | we can output anything we desire from here...  |
      |                                                |
      +------------------------------------------------+ */

      define event EmitMeta; ❶
      start:
         put ’<’ $LIBRARYNAME ’>’ CR ;
         put ’   <!-- ’ CR ;
         put ’        List of available columns’ CR  ;

         eval $index  1;
         iterate $col_names ;
         do /while _value_;
            put ’           ’ $index ’ ’ _value_ CR  ;
            next $col_names;
            eval $index $index+1;
            done;
         put ’   -->’ CR ;
         break;
      finish:
         put ’</’ $LIBRARYNAME ’>’ ;
         break;
      end;


      define event EmitRow; ❷
         ndent;
         put "<STUDENT>" CR ;
         ndent;

         set $name  "Name";   trigger EmitCol ;
         set $name  "Height"; trigger EmitCol ;
         set $name  "Weight"; trigger EmitCol ;

         xdent;
         put "</STUDENT>" CR ;
         xdent;
         break;
      end;


      define event EmitCol; ❸
         unset $value;
         set $value $col_values[$name];
         put ’<’  $name ’>’ ;
         put      $value ;
         put ’</’ $name ’>’ CR ;
         break;
      end;

   end; /* custom */
run;
```

Export XML Document using Customized Tagset

The following SAS program exports a SAS data set as an XML document using the customized tagset:

```
data work.class; ❶
   set sashelp.class (obs=3);
run;

filename XMLout "C:\My Documents\XML\engine92.xml"; ❷

libname  XMLout xml xmltype=GENERIC tagset=tagsets.custom ; ❸

data XMLout.class; ❹
   set work.class;
run;
```

1 The DATA step creates a data set named WORK.CLASS that consists of only three observations.

2 The FILENAME statement assigns the fileref XMLOUT to the physical location of the file that will store the exported XML document (complete pathname, filename, and file extension).

3 The LIBNAME statement uses the fileref to reference the XML document and specifies the XML engine. The TAGSET= option specifies the customized tagset named Tagsets.Custom.

4 The DATA step reads the data set WORK.CLASS and writes its content to the specified XML document in the format that is defined by the customized tagset.

Here is the resulting XML document:

Output 6.1 Exported XML Document Using Customized Tagset

```
<?xml version="1.0" encoding="windows-1252" ?>
<LIBRARY>
   <!--
        List of available columns
           1 Name
           2 Sex
           3 Age
           4 Height
           5 Weight
   -->
   <STUDENT>
      <Name>Alfred</Name>
      <Height>69</Height>
      <Weight>112.5</Weight>
   </STUDENT>
   <STUDENT>
      <Name>Alice</Name>
      <Height>56.5</Height>
      <Weight>84</Weight>
   </STUDENT>
   <STUDENT>
      <Name>Barbara</Name>
      <Height>65.3</Height>
      <Weight>98</Weight>
   </STUDENT>
</LIBRARY>
```

PART 2

Reference

CHAPTER 7

LIBNAME Statement for the XML Engine

Using the LIBNAME Statement

For the XML engine, the LIBNAME statement associates a SAS libref with either a SAS library that stores XML documents or a specific XML document in order to import or export an XML document.

For basic examples, see Chapter 3, “Importing XML Documents,” on page 21 and Chapter 2, “Exporting XML Documents,” on page 7.

Understanding the XML LIBNAME Engine Versions: XML and XML92

About the XML Engine Versions

SAS 9.2 provides two versions for XML LIBNAME engine functionality by implementing engine nicknames in the LIBNAME statement.

□ By specifying the engine nickname **`XML`**, you access the SAS 9.1.3 XML engine functionality.

□ By specifying the engine nickname **`XML92`**, you access the new SAS 9.2 XML engine functionality with enhancements and changes. For example, the XML92 version provides enhanced LIBNAME statement functionality, new XMLMap functionality, and diagnostics of obsolete syntax.

The major differences between the versions include the following:

□ The XML92 version is XML compliant.

- □ LIBNAME statement functionality enhancements for XML92 include the WSDL and XMLMAP markup types, additional options, and the ability to assign a libref to a SAS library.
- □ New XMLMap functionality for XML92 includes the ability to use an XMLMap for exporting.

Comparing the XML Engine Versions: XML92 and XML

XML Compliance

The XML92 version is XML compliant, which means that XML92 requires XML markup to be well-formed and in valid construction that is in compliance with the W3C specifications. Because the XML92 version is XML compliant, using XML92 could affect the following situations:

- □ XML documents that are imported with the XML version might not pass the more strict parsing rules in the XML92 version. For example, like XML markup, the XML92 version is case sensitive. Opening and closing tags must be written in the same case, such as **`<BODY> ...</BODY>`** and **`<Message>...</Message>`**. For the XML92 version, the tag **`<Letter>`** is different from the tag **`<letter>`**. Attribute names are also case sensitive, and the attribute value must be enclosed in quotation marks, such as **`<Note date="09/24/1975">`**.
- □ XMLMap files that are accepted by the XML version might not work with the XML92 version. The XML92 version requires that XMLMap files be XML compliant, which means that the markup is case sensitive. In addition, the XMLMap markup must follow the specific XMLMap rules. Tag names must be uppercase. Element attributes must be lowercase. An example is **`<SXLEMAP version="1.9">`**. In addition, the supported XPath syntax is case sensitive.

XMLMap Files

The XML version supports all XMLMap files starting with XMLMap version 1.0. The XML92 version supports XMLMap files starting with XMLMap version 1.2. The documented XMLMap syntax version is 1.9. See “XMLMap Syntax Version 1.9” on page 100.

LIBNAME Statement Functionality Enhancements for XML92

The XML92 version provides the following LIBNAME statement functionality:

- □ The ability to assign a libref to a SAS library, rather than assigning the libref to a specific XML document. See SAS-library.
- □ Additional XML markup types such as WSDL and XMLMAP. See XMLTYPE=.
- □ Additional options. For a list of the LIBNAME statement options that are available for the XML and XML92 nicknames, see Table 7.1 on page 88.
- □ Using the XML92 version and the GENERIC markup type, you can export an XML document from multiple SAS data sets. For example, if you have two SAS data sets named Grades.Fred and Grades.Wilma, the following code exports an XML document named Grades.xml that includes the grades from both SAS data sets:

```
libname stones xml92 'c:\Grades.xml';
```

```
data stones.fred;
   set grades.fred;
run;

data stones.wilma;
   set grades.wilma;
run;
```

LIBNAME Statement Syntax

Syntax

LIBNAME *libref engine* <'*SAS-library* | *XML-document-path*'> <*options*>;

Required Arguments

libref

is a valid SAS name that serves as a shortcut name to associate with the physical location of the XML document. The name must conform to the rules for SAS names. A libref cannot exceed eight characters.

engine

is the engine nickname for the SAS XML LIBNAME engine that imports and exports an XML document.

XML

specifies the XML engine nickname that accesses the SAS 9.1.3 XML engine functionality. The syntax for functionality that is available only for the XML engine nickname is labeled with the symbol [XML Only].

XML92

specifies the XML engine nickname that accesses the new SAS 9.2 XML engine functionality. The syntax for functionality that is available only for the XML92 engine nickname is labeled with the symbol [XML92 Only].

Note: In the third maintenance release for SAS 9.2, the XML92 functionality is production except for the WSDL markup type, which is preproduction. △

Interaction: If syntax is not labeled with either the [XML Only] or [XML92 Only] symbols, the functionality is available for both engine nicknames.

Tip: At your site, the engine nicknames could be different if your system administrator assigned an alias to the XML LIBNAME engine. See your system administrator to determine whether an alias is assigned.

'*SAS-library* | *XML-document-path*'

is the physical location of the XML document for export or import. Enclose the physical location in single or double quotation marks.

SAS-library [XML92 Only]

is the pathname for a collection of one or more files that are recognized by SAS, and that are referenced and stored as a unit. For example:

```
'C:\My Documents\XML'
```

XML-document-path
includes the pathname, filename, and file extension. For example:
'C:\My Documents\XML\myfile.xml'

Operating Environment Information: For details about specifying the physical location of files, see the SAS documentation for your operating environment. △

Interaction: You can use the FILENAME statement in order to assign a fileref to be associated with the physical location of the XML document to be exported or imported.

If the fileref matches the libref, you do not need to specify the physical location of the XML document in the LIBNAME statement. For example, the following code writes to the XML document Fred.XML:

```
filename bedrock 'C:\XMLdata\fred.xml';

libname bedrock xml;

proc print data=bedrock.fred;
run;
```

To specify a fileref for the XML document that does not match the libref, you can use the XMLFILEREF= option on page 94. For example, the following code writes to the XML document Wilma.XML:

```
filename cartoon 'C:\XMLdata\wilma.xml';

libname bedrock xml xmlfileref=cartoon;

proc print data=bedrock.wilma;
run;
```

Options

The following table lists the available LIBNAME statement options. The ✓ symbol indicates whether the option is available for an engine nickname.

Table 7.1 LIBNAME Statement Options

Task	Option	XML	XML92
When importing Web service results with the WSDL markup type			
Specify an HTTP proxy server host name	PROXYHOST=		✓
Specify an HTTP proxy server password	PROXYPASSWORD=		✓
Specify an HTTP proxy server port	PROXYPORT=		✓
Specify an HTTP proxy server user name	PROXYUSERNAME=		✓
Specify a Web server password	WEBPASSWORD=		✓
Specify a Web server user name	WEBUSERNAME=		✓
Determine whether SAS formats are used	FORMATACTIVE=	✓	✓
When importing or exporting a CDISC ODM XML document with the CDISCODM markup type			

Task	Option	XML	XML92
Determine whether SAS formats are used	FORMATACTIVE=	✓	
Specify the libref to create a format catalog	FORMATLIBRARY=	✓	
Replace existing format entries in the format catalog	FORMATNOREPLACE=	✓	
Indent nested elements in exported XML document	INDENT=	✓	✓
Specify the character set to use for the output file	ODSCHARSET=	✓	✓
Control the generation of a record separator	ODSRECSEP=	✓	✓
Specify the translation table to use for the output file	ODSTRANTAB=	✓	✓
Override the default tagset	TAGSET=	✓	✓
Import concatenated XML documents	XMLCONCATENATE=	✓	✓
Specify the tag format to contain SAS variable information	XMLDATAFORM=	✓	✓
Control the results of numeric values	XMLDOUBLE=	✓	✓
Override the SAS data set's encoding for the output file	XMLENCODING=	✓	✓
Specify a fileref for the XML document	XMLFILEREF=	✓	✓
Specify an XMLMap	XMLMAP=	✓	✓
Determine whether metadata-related information is included	XMLMETA=	✓	✓
Determine whether to process nonconforming character data	XMLPROCESS=	✓	✓
Specify an external file to contain exported metadata-related information	XMLSCHEMA=	✓	✓
Specify the XML markup type	XMLTYPE=	✓	✓

FORMATACTIVE=NO | YES

determines whether SAS formats are used.

For the CDISCODM markup type, FORMATACTIVE= specifies whether CDISC ODM CodeList elements, which contain instructions for transcoding display data in a CDISC ODM document, are to be converted to SAS formats, and vice versa.

NO XML Only

causes formatting controls to be suppressed for both importing and exporting.

YES XML Only

when importing, converts the CDISC ODM CodeList elements to the corresponding SAS formats, registers the SAS formats on the referenced variables, and stores the created SAS formats in the format catalog.

When exporting, converts the SAS formats to the corresponding CDISC ODM CodeList elements.

Tip: By default, the format catalog is created in the Work library. If you want to store the catalog in a permanent library, use the FORMATLIBRARY= option.

Tip: When the format catalog is updated, the default behavior is that any new SAS formats that are created by converting CDISC ODM CodeList elements will overwrite any existing SAS formats that have the same name. To prevent existing SAS formats from being overwritten, specify FORMATNOREPLACE=YES.

Featured in: "Exporting an XML Document in CDISC ODM Markup" on page 18

For the GENERIC markup type, specifies whether output values are affected by SAS formats.

NO
: writes the actual data value to the XML markup.

YES
: causes the XML markup to contain the formatted data value.

Restriction: For the GENERIC markup type, if you export a SAS data set with formatted data values, and then you try to import the XML document back into the existing SAS data set, the import might fail. Exporting a SAS data set with formatted data values can result in different variables or different variable attributes.

Default: NO

Restriction: Use this option for the CDISCODM and GENERIC markup types only.

FORMATLIBRARY=*libref* XML Only

specifies the libref of an existing SAS library in which to create the format catalog.

Restriction: Use this option when importing an XML document only.

Restriction: Use this option only for the CDISCODM markup type with FORMATACTIVE=YES.

FORMATNOREPLACE=NO | YES XML Only

specifies whether to replace existing format entries in the format catalog search path in cases where an existing format entry has the same name as a format that is being created by the XML engine when it converts a CDISC ODM CodeList element.

NO
: does not replace formats that have the same name.

YES
: replaces formats that have the same name.

Default: NO

Restriction: Use this option when importing an XML document only.

Restriction: Use this option for the CDISCODM markup type only.

INDENT=*integer*

specifies the number of columns to indent each nested element in the exported XML document. The value can be from 0 (which specifies no indention) through 32. This specification is cosmetic and is ignored by an XML-enabled browser.

Default: 3

Restriction: Use this option when exporting an XML document only.

ODSCHARSET=*character-set*

specifies the character set to use for the output file. A character set includes letters, logograms, digits, punctuation, symbols, and control characters that are used for display and printing. An example of a character set is ISO-8859-1.

Restriction: Use this option when exporting an XML document only.

Requirement: Use this option with caution. If you are unfamiliar with character sets, encoding methods, or translation tables, do not use this option without proper technical advice.

Tip: The combination of the character set and translation table (encoding method) results in the file's encoding.

See: "ODSCHARSET= Option" in *SAS National Language Support (NLS): Reference Guide*.

ODSRECSEP= DEFAULT | NONE | YES

controls the generation of a record separator that marks the end of a line in the output XML document.

DEFAULT

enables the XML engine to determine whether to generate a record separator based on the operating environment where you run the SAS job.

The use of a record separator varies by operating environment.

Tip: If you do not transfer XML documents across environments, use the default behavior.

NONE

specifies to not generate a record separator.

The XML engine uses the logical record length of the file that you are writing to and writes one line of XML markup at a time to the output file.

Requirement: The logical record length of the file that you are writing to must be at least as long as the longest line that is produced. If the logical record length of the file is not long enough, then the markup might wrap to another line at an inappropriate place.

Interaction: Transferring an XML document that does not contain a record separator can be a problem. For example, FTP needs a record separator in order to transfer data properly in ASCII (text) mode.

YES

specifies to generate a record separator.

Default: The XML engine determines whether to generate a record separator based on the operating environment where you run the SAS job.

Restriction: Use this option when exporting an XML document only.

Interaction: Most transfer utilities interpret the record separator as a carriage return sequence. For example, using FTP in ASCII (text) mode to transfer an XML document that contains a record separator results in properly constructed line breaks for the target environment.

ODSTRANTAB=*table-name*

specifies the translation table to use for the output file. The translation table (encoding method) is a set of rules that are used to map characters in a character set to numeric values. An example of a translation table is one that converts characters from EBCDIC to ASCII-ISO. The *table-name* can be any translation table that SAS provides or any user-defined translation table. The value must be the name of a SAS catalog entry in either the SASUSER.PROFILE catalog or the SASHELP.HOST catalog.

Restriction: Use this option when exporting an XML document only.

Requirement: Use this option with caution. If you are unfamiliar with character sets, encoding methods, or translation tables, do not use this option without proper technical advice.

Tip: The combination of the character set and translation table results in the file's encoding.

See: "ODSTRANTAB= Option" in *SAS National Language Support (NLS): Reference Guide*.

PROXYHOST=’*hostname*’ XML92 Only

specifies an HTTP proxy server host name. An example is **proxyhost='proxygw.abc.sas.com'**.

Restriction: Use this option for the WSDL markup type only.

Restriction: The functionality is for the third maintenance release for SAS 9.2.

PROXYPASSWORD=’*password*’ XML92 Only

specifies an HTTP proxy server password. This option is required only if your proxy server requires credentials. The password is case sensitive and must be enclosed in single or double quotation marks. An example is **proxypassword='Abc_123'**.

Restriction: Use this option for the WSDL markup type only.

Restriction: The functionality is for the third maintenance release for SAS 9.2.

PROXYPORT=*number* XML92 Only

specifies an HTTP proxy server port. An example is **proxyport=80**.

Restriction: Use this option for the WSDL markup type only.

Restriction: The functionality is for the third maintenance release for SAS 9.2.

PROXYUSERNAME=’*user-ID*’ XML92 Only

specifies an HTTP proxy server user name. This option is required only if your proxy server requires credentials. The user ID is case sensitive and must be enclosed in single or double quotation marks. An example is **proxyusername='test_acc'**.

Restriction: Use this option for the WSDL markup type only.

Restriction: The functionality is for the third maintenance release for SAS 9.2.

TAGSET=*tagset-name*

specifies the name of a tagset in order to override the default tagset that is used by the markup type that is specified with XMLTYPE=.

To change the tags that are produced, you can create a customized tagset and specify it with the TAGSET= option. For information about creating customized tagsets, see PROC TEMPLATE in the *SAS Output Delivery System: User’s Guide*.

Restriction: Use this option when exporting an XML document only.

Requirement: Use this option with caution. If you are unfamiliar with XML markup, do not use this option.

See also: Chapter 6, “Understanding and Using Tagsets for the XML Engine,” on page 75

Featured in: “Exporting an XML Document Using a Customized Tagset” on page 75

CAUTION:

If you alter the tagset when exporting an XML document and then attempt to import the XML document generated by that altered tagset, the XML engine might not be able to translate the XML markup back to SAS proprietary format. △

WEBPASSWORD=’*password*’ XML92 Only

specifies a password for either Basic or NTLM Web server authentication. The password is case sensitive and must be enclosed in single or double quotation marks. An example is **webpassword='MyPW'**.

Restriction: Use this option for the WSDL markup type only.

Restriction: The functionality is for the third maintenance release for SAS 9.2.

WEBUSERNAME=*'user-ID'* XML92 Only

specifies a user name for either Basic or NTLM Web server authentication. The user ID is case sensitive and must be enclosed in single or double quotation marks. An example is `webusername='mydomain\myuserid'`.

Restriction: Use this option for the WSDL markup type only.

Restriction: The functionality is for the third maintenance release for SAS 9.2.

XMLCONCATENATE | XMLCONCAT=NO | YES

specifies whether the file to be imported contains multiple, concatenated XML documents. Importing multiple, concatenated XML documents can be useful (for example, if an application is producing a complete document per query/response, as in a Web form).

Default: NO

Restriction: Use this option when importing an XML document only.

Requirement: Use XMLCONCATENATE=YES cautiously. If an XML document consists of concatenated XML documents, the content is not standard XML construction. The option is provided for convenience, not to encourage invalid XML markup.

Featured in: "Importing Concatenated XML Documents" on page 30

XMLDATAFORM=ELEMENT | ATTRIBUTE

specifies whether the tag for the element to contain SAS variable information (name and data) is in open element or enclosed attribute format. For example, if the variable name is PRICE and the value of one observation is `1.98`, the generated output for ELEMENT is `<PRICE> 1.98 </PRICE>` and for ATTRIBUTE it is `<COLUMN name="PRICE" value="1.98" />`.

Default: ELEMENT

Restriction: Use this option when exporting an XML document only.

Restriction: Use this option for the GENERIC markup type only.

XMLDOUBLE=DISPLAY | INTERNAL

controls the results of importing or exporting numeric values.

DISPLAY

when exporting, the engine retrieves the stored value for the numeric variable, determines an appropriate display for the value in a readable form, and writes the display value to the XML document. In some cases, values might not be displayed completely, or they might show truncation or rounding artifacts.

When importing, the engine retrieves PCDATA (parsed character data) from the named element in the XML document and converts that value to numeric variable content.

Alias: FORMAT

INTERNAL

when exporting, the XML engine retrieves the stored value for the numeric variable and writes the raw value to a generated attribute-value pair (of the form `rawvalue="value"`). SAS uses a base64 encoding of a portable machine representation. (The base64 encoding method converts binary data into ASCII text, and vice versa, and is similar to the MIME format.)

When importing, the XML engine retrieves the value from the `rawvalue=` attribute from the named element in the XML document and converts that value to numeric variable content. The PCDATA content of the element is ignored.

Typically, you use XMLDOUBLE=INTERNAL to import or export an XML document when data content is more important than readability.

Alias: PRECISION

Default: For the GENERIC markup type, the default is XMLDOUBLE=DISPLAY.

Restriction: You can specify the XMLDOUBLE= option for the GENERIC markup type only.

Featured in: "Exporting Numeric Values" on page 10 and "Importing an XML Document with Numeric Values" on page 23

XMLENCODING='*encoding-value*'

overrides the SAS data set's encoding for the output file. If an encoding value contains a hyphen, enclose the value in quotation marks.

Restriction: Use this option when exporting an XML document only.

Requirement: Use this option with caution. If you are unfamiliar with character sets, encoding methods, or translation tables, do not use this option without proper technical advice.

Tip: When transferring an XML document across environments (for example, using FTP), you must be aware of the document's content to determine the appropriate transfer mode. If the document contains an encoding attribute in the XML declaration, or if a byte-order mark (BOM) precedes the XML declaration, transfer the XML document in binary mode. If the document contains neither of these, and you are transferring the document across similar environments, transfer the XML document in text mode.

Tip: The combination of the character set and translation table (encoding method) results in the file's encoding.

See: "XMLENCODING= Option" in *SAS National Language Support (NLS): Reference Guide*

XMLFILEREF=*fileref*

is the SAS name that is associated with the physical location of the XML document to be exported or imported. To assign the fileref, use the FILENAME statement. The XML engine can access any data referenced by a fileref. For example, the following code writes to the XML document Wilma.XML:

```
filename cartoon 'C:\XMLdata\wilma.xml';

libname bedrock xml xmlfileref=cartoon;

proc print data=bedrock.wilma;
run;
```

Tip: When using the URL access method to reference a fileref that is assigned to an XML document, you should also specify an XMLMap. Specifying an XMLMap causes the XML engine to process the XML document with a single pass. Whether you need to specify an XMLMap depends on your Web server. For an example, see "Referencing a Fileref Using the URL Access Method" on page 70.

XMLMAP=*fileref* | '*XMLMap*'

specifies an XML document that you create that contains specific XMLMap syntax. The syntax tells the XML engine how to interpret the XML markup for importing or exporting. The XMLMap syntax is itself XML markup.

fileref

is the SAS name that is associated with the physical location of the XMLMap. To assign a fileref, use the FILENAME statement.

Tip: To assign a fileref to an XMLMap using the URL access method, your Web server might require that the file extension be **.xml** instead of **.map**.

'*XMLMap*'

is the physical location of the XMLMap. Include the complete pathname and the filename. It is suggested that you use the filename extension .map. Enclose the physical name in single or double quotation marks.

For example, the following statements import an XML document named MY.XML and specify the XMLMap named MY.MAP, which contains specific XMLMap syntax. The XML engine interprets the XML document as a SAS data set named TEST.MY. In this example, XMLMAP= is used as an option in the LIBNAME statement:

```
libname test xml 'C:\XMLdata\my.xml' xmlmap='C:\XMLdata\my.map';

proc print data=test.my;
run;
```

Requirement: If you specify an XMLMap, specify XMLTYPE=XMLMAP or do not specify a markup type. If you explicitly specify a markup type other than XMLMAP (such as XMLTYPE=GENERIC), an error occurs.

See: Chapter 8, "XMLMap Files for the XML Engine," on page 99

Featured in: Chapter 5, "Importing XML Documents Using an XMLMap," on page 45

XMLMETA=DATA | SCHEMADATA | SCHEMA

specifies whether to include metadata-related information in the exported markup, or specifies whether to import metadata-related information that is included in the input XML document.

Metadata-related information is metadata that describes the characteristics (types, lengths, levels, and so on) of columns within the table markup. Including the metadata-related information can be useful when exporting an XML document from a SAS data set to process on an external product.

DATA

ignores metadata-related information. DATA includes only data content in the exported markup and imports only data content in the input XML document.

SCHEMADATA

includes both data content and metadata-related information in the exported markup and imports both data content and metadata-related information in the input XML document.

SCHEMA

ignores data content. SCHEMA includes only metadata-related information in the exported markup and imports only metadata-related information in the input XML document.

Alias:

DATA	NONE, NO, IGNORE
SCHEMADATA	FULL, YES

Default: DATA

Restriction: Use this option for the GENERIC and MSACCESS markup types only.

Interaction: If XMLMETA=SCHEMADATA and XMLSCHEMA= is specified, the data is written to the physical location of the XML document specified in the

LIBNAME statement. Separate metadata-related information is written to the physical location specified with XMLSCHEMA=. If XMLSCHEMA= is not specified, the metadata-related information is embedded with the data content in the XML document.

Tip: Before SAS 9, the functionality for the XMLMETA= option used the keyword XMLSCHEMA=. SAS 9 changed the option keyword XMLSCHEMA= to XMLMETA=. SAS 9.1 added new functionality using the XMLSCHEMA= option.

Featured in: "Exporting an XML Document with Separate Metadata" on page 15 and "Importing an XML Document Created by Microsoft Access" on page 26

XMLPROCESS=CONFORM | PERMIT

determines how the XML engine processes character data that does not conform to W3C specifications.

CONFORM

requires that the XML conform to W3C specifications. W3C specifications state that for character data, certain characters such as the left angle bracket (<), the ampersand (&), and the apostrophe (') must be escaped using character references or strings like `&amp;`. For example, to allow attribute values to contain both single and double quotation marks, the apostrophe or single-quotation mark character (') can be represented as `&apos;` and the double-quotation mark character (") can be represented as `&quot;`.

PERMIT

permits character data that does not conform to W3C specifications to be accepted. That is, in character data, non-escaped characters such as the apostrophe, double quotation marks, and the ampersand are accepted.

Restriction: Non-escaped angle brackets in character data are not accepted.

Restriction: Use XMLPROCESS=PERMIT cautiously. If an XML document consists of non-escaped characters, the content is not standard XML construction. The option is provided for convenience, not to encourage invalid XML markup.

Default: CONFORM

Featured in: "Importing an XML Document with Non-Escaped Character Data" on page 24

XMLSCHEMA=*fileref* | '*external-file*'

specifies an external file to contain metadata-related information.

fileref

is the SAS name that is associated with the physical location of the output file. To assign a fileref, use the FILENAME statement.

'*external-file*'

is the physical location of the file to contain the metadata-related information. Include the complete pathname and the filename. Enclose the physical name in single or double quotation marks.

Restriction: Use this option when exporting an XML document only.

Restriction: Use this option only for the GENERIC and MSACCESS markup types with XMLMETA=SCHEMADATA.

Interaction: If XMLMETA=SCHEMADATA and XMLSCHEMA= is specified, the data is written to the physical location of the XML document specified in the LIBNAME statement. Separate metadata-related information is written to the physical location specified with XMLSCHEMA=. If XMLSCHEMA= is not specified, the metadata-related information is embedded with the data content in the XML document.

Featured in: "Exporting an XML Document with Separate Metadata" on page 15

XMLTYPE=GENERIC | CDISCODM | EXPORT | MSACCESS | ORACLE | WSDL | XMLMAP

specifies the XML markup type:

Default: GENERIC

Tip: You can control the markup by specifying options such as INDENT=, XMLDATAFORM=, XMLMETA= (when applicable), and TAGSET=.

GENERIC

is a simple, well-formed XML markup type. The XML document consists of a root (enclosing) element and repeating element instances. GENERIC determines a variable's attributes from the data content.

Requirement: When importing, the GENERIC markup type requires a specific physical structure. See "Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type" on page 45.

Featured in: "Exporting an XML Document Containing SAS Dates, Times, and Datetimes" on page 9, "Exporting Numeric Values" on page 10, "Importing an XML Document Using the GENERIC Markup Type" on page 21, and several examples throughout the document.

CDISCODM XML Only

is the XML markup type for the markup standards that are defined in the Operational Data Model (ODM) that was created by the Clinical Data Interchange Standards Consortium (CDISC). The XML engine supports the ODM 1.2 schema specification. ODM supports the electronic acquisition, exchange, and archiving of clinical trials data and metadata for medical and biopharmaceutical product development.

Tip: Use the FORMATACTIVE=, FORMATNOREPLACE=, and FORMATLIBRARY= options to specify how display data are read and stored in the target environment.

Featured in: "Importing a CDISC ODM Document" on page 32 and "Exporting an XML Document in CDISC ODM Markup" on page 18

EXPORT XML Only

is an alias to specify the XML format that is most commonly used in the industry. For the XML engine, specifying XMLTYPE=EXPORT is the same as specifying XMLTYPE=OIMDBM, which was deprecated in SAS 9. Future releases will upgrade this format specification, as needed.

MSACCESS XML Only

is the XML markup type for the markup standards supported for a Microsoft Access database (.mdb). If the Microsoft Access file contains metadata-related information, then you must specify MSACCESS rather than the default GENERIC markup type. If there is an embedded XML schema, specifying MSACCESS and the XMLMETA=SCHEMADATA option causes a variable's attributes to be obtained from the embedded schema. If there is not an embedded schema, MSACCESS uses default values for attributes.

Featured in: "Importing an XML Document Created by Microsoft Access" on page 26

ORACLE XML Only

is the XML markup type for the markup standards equivalent to the Oracle 8i XML implementation. The number of columns to indent each nested element is one, and the enclosing element tag for the contents of the SAS data set is ROWSET.

Featured in: "Exporting an XML Document for Use by Oracle" on page 7

WSDL XML92 Only (Preproduction)

is the XML markup type that interacts with a WSDL file to invoke a Web service and import the Web service results. The WSDL markup type interacts with a WSDL file that supports the markup standards of WSDL 1.1.

Restriction: The functionality is for SAS 9.2 Phase 2 and later.

Restriction: Use this option to import Web service results only.

Restriction: The `import` element, which references other WSDL files, is not supported.

Restriction: For data types that are defined in the type= attribute in an XML schema `element` element, only the integer, string, and double data types are supported.

Restriction: For the `complexType` element, which determines the set of attributes and the content of an element, the name= attribute is not supported.

Restriction: The `simpleType` element is not supported.

Requirement: The Web service must specify a document-oriented SOAP operation. Therefore, the WSDL file must include the `soap:operation` element style= attribute that is either set or defaults to the document value. For information about the `soap:operation` element style= attribute, see the W3C WSDL 1.1 specification.

Requirement: The Web service must require both input and output parameters. A Web service that does not accept input parameters or has no return value is not supported.

See: "Importing Web Service Results Using the WSDL Markup Type (Preproduction)" on page 35

See also: "PARMS= Data Set Option" on page 117

XMLMAP XML92 Only

specifies that XML markup is determined by an XMLMap, which is an XML document that you create that contains specific XMLMap syntax. The XMLMap syntax tells the XML engine how to map the SAS data back into the specific XML document structure. To specify the XMLMap in the LIBNAME statement, see XMLMAP=.

Restriction: Exporting an XML document that is controlled by an XMLMap is limited to a single SAS data set.

Featured in: "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 41

CHAPTER 8

XMLMap Files for the XML Engine

Comparing the XMLMap Functionality

The most significant difference between the two XML engine versions of XMLMap functionality is that XML92 enables you to export an XML document by using an XMLMap.

The following table lists the available XMLMap functionality. The ✓ symbol indicates whether the functionality is available for importing or exporting, and whether the option is available for an engine nickname.

Table 8.1 XMLMap Functionality

Functionality	Description	Import	Export	XML	XML92
SXLEMAP	Primary (root) enclosing element	✓	✓	✓	✓
OUTPUT	Contains one or more HEADING elements		✓		✓
HEADING	Contains one or more ATTRIBUTE elements		✓		✓
ATTRIBUTE	Contains file attribute information		✓		✓
TABLEREF	Specifies the name of the table		✓		✓
TABLE	Contains a data set definition	✓	✓	✓	✓
TABLE-PATH	Specifies a location path for variables	✓	✓	✓	✓
TABLE-END-PATH	Specifies a location path to stop processing	✓	✓	✓	✓

Functionality	Description	Import	Export	XML	XML92
TABLE-DESCRIPTION	Specifies a SAS data set description	✓	✓	✓	✓
COLUMN name=	Specifies the variable name	✓	✓	✓	✓
COLUMN retain=	Determines the contents of the input buffer	✓	✓	✓	✓
COLUMN replace=	Controls the concatenation of data	✓	✓	✓	✓
COLUMN ordinal=	Determines whether the variable is a counter variable	✓	✓	✓	✓
COLUMN class=	Determines the type of variable	✓	✓		✓
TYPE	Specifies the SAS data type for the variable	✓	✓	✓	✓
DATATYPE	Specifies the type of data being read	✓	✓	✓	✓
DEFAULT	Specifies a default value for a missing value	✓	✓	✓	✓
ENUM	Contains a list of valid values for the variable	✓	✓	✓	✓
FORMAT	Specifies a SAS format for the variable	✓	✓	✓	✓
INFORMAT	Specifies a SAS informat for the variable	✓	✓	✓	✓
DESCRIPTION	Specifies a description for the variable	✓	✓	✓	✓
LENGTH	Determines the maximum field storage length for a character variable	✓	✓	✓	✓
PATH	Specifies a location path for the current variable	✓	✓	✓	✓
INCREMENT-PATH	Specifies a location path for incrementing the accumulated value for a counter variable	✓	✓	✓	✓
RESET-PATH	Specifies a location path for resetting the accumulated value for a counter variable to zero	✓	✓	✓	✓
DECREMENT-PATH	Specifies a location path to decrement the accumulated value for the counter variable by 1	✓	✓	✓	✓

XMLMap Syntax Version 1.9

Using XMLMap Syntax

The XML elements for the XMLMap syntax for version 1.9 are explained in this chapter. The elements are listed in the order in which you would typically include them in an XMLMap. That is:

□ The first element in the XMLMap is the SXLEMAP element, which is the primary (root) enclosing element that contains the definition. See “Elements for XMLMap” on page 101.

□ If you use an XMLMap for exporting, you must include the exporting elements. See “Elements for Exporting” on page 102.

- □ The table elements define the SAS data set. See “Elements for Tables” on page 103.
- □ The column elements define the variables for the SAS data set. See “Elements for Columns” on page 106.

CAUTION:
The XMLMap markup, as XML itself, is case sensitive. The tag names must be uppercase, and the element attributes must be lowercase. An example is **`<SXLEMAP version="1.9">`**. In addition, the supported XPath syntax is case sensitive as well. △

Elements for XMLMap

SXLEMAP version="*number*" name="*XMLMap*" description="*description*"
is the primary (root) enclosing element that contains the definition for the generated output file. The element provides the XML well-formed constraint for the definition.

Restriction: When importing an XML document, the definition can define more than one output SAS data set. When exporting an XML document from a SAS data set, the definition can define only one output XML document.

Requirement: The SXLEMAP element is required.

version="*number*"
specifies the version of the XMLMap syntax. The documented XMLMap syntax version is 1.9 and must be specified to obtain full functionality.

Default: The default version is the first version of XMLMap syntax. It is retained for compatibility with prior releases of the XMLMap syntax. It is recommended that you update existing XMLMaps to version 1.9.

Restriction: The XML92 engine nickname does not support XMLMap syntax before version 1.2. If the version is 1.0 or 1.1, an error occurs.

Tip: To update an XMLMap to version 1.9, load the existing XMLMap into SAS XML Mapper, change the version to 1.9 on the XMLMap Settings tab, and then save the XMLMap. For information about SAS XML Mapper, see “Using SAS XML Mapper to Generate and Update an XMLMap” on page 114.

name="*XMLMap*"
is an optional attribute that specifies the filename of the XMLMap.

description="*description*"
is an optional attribute that specifies a description of the XMLMap.

In the example below, the SXLEMAP element specifies all three attributes and contains two TABLE elements.

```
<?xml version="1.0" ?>
<SXLEMAP version="1.9" name="Myxmlmap"
         description="sample XMLMap">
    <TABLE name="test1">
       .
       .
       .
    </TABLE>
```

```
<TABLE name="test2">
   .
   .
   .
</TABLE>
</SXLEMAP>
```

Elements for Exporting

OUTPUT XML92 Only

is an optional element that contains one or more HEADING elements and one TABLEREF element for exporting a SAS data set as an XML document.

Requirement: If you specify version 1.9 in an XMLMap to export a SAS data set as an XML document, you must include the OUTPUT element in the XMLMap.

Featured in: "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 41

HEADING XML92 Only

is an optional element that contains one or more ATTRIBUTE elements.

Featured in: "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 41

ATTRIBUTE name="*name*" value="*value*" XML92 Only

is an optional element that contains additional file attribute information for the exported XML document, such as a schema reference, XML namespace, or other general attributes. The specified name-value pairs are added as attributes to the first generated element in the exported XML document, such as, **`<NHL description="Teams of the National Hockey League">`**.

Featured in: "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 41

name="*name*"

specifies a name for a file attribute, such as **`name="description"`**.

value="*value*"

specifies a value for the attribute, such as **`value="Teams of the National Hockey League"`**.

TABLEREF name="*name*" XML92 Only

is an optional element that specifies the name of the table in the XMLMap to be exported.

Restriction: You can specify one TABLEREF element only.

Requirement: The specified name must match a TABLE element name= attribute.

Featured in: "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 41

name="*name*"

specifies the name of the table in the XMLMap to be exported. The name must be unique in the XMLMap definition, and the name must be a valid SAS name, which can be up to 32 characters.

Elements for Tables

TABLE name="*data-set-name*"
is an element that contains a data set definition.

```
<TABLE name="channel">
```

Requirement: The TABLE element is required.

Interaction: The TABLE element can contain one or more of the following elements: TABLE-PATH, TABLE-END-PATH, TABLE-DESCRIPTION, and COLUMN.

name="*data-set-name*"
specifies the name for the SAS data set. The name must be unique in the XMLMap, and the name must be a valid SAS name, which can be up to 32 characters.

Requirement: The name= attribute is required.

TABLE-PATH syntax="*type*"
specifies a location path that tells the XML engine where in the XML document to locate and access specific elements in order to collect variables for the SAS data set. The location path defines the repeating element instances in the XML document, which is the SAS data set observation boundary. The observation boundary is translated into a collection of rows with a constant set of columns.

Requirement: The TABLE-PATH element is required.

syntax="*type*"
is an optional attribute that specifies the type of syntax in the location path. The syntax is valid XPath construction in compliance with the W3C specifications.

Default: The default is XPATH, that is, **`syntax="XPATH"`**.

Requirement: The XPath construction is a formal specification that puts a path description similar to UNIX on each element of the XML structure. *Note that XPath syntax is case sensitive. For example, if an element tag name is uppercase, it must be uppercase in the location path. If it is lowercase, it must be lowercase.* All location paths must begin with the root-enclosing element (denoted by a slash '/') or with the "any parent" variant (denoted by double slashes '//'). Other W3C documented forms are not currently supported.

For example, using the XML document RSS.XML, which is used in the example “Using an XMLMap to Import an XML Document as Multiple SAS Data Sets” on page 51, this TABLE-PATH element causes the following to occur:

```
<TABLE-PATH syntax="XPATH"> /rss/channel/item </TABLE-PATH>
```

1 The XML engine reads the XML markup until it encounters the <ITEM> start tag.

2 The XML engine clears the input buffer, sets the contents to MISSING (by default), and scans elements for variable names based on the COLUMN element definitions. As values are encountered, they are read into the input buffer. (Note that whether the XML engine resets to MISSING is determined by the DEFAULT element as well as the COLUMN element retain= attribute.)

3 When the </ITEM> end tag is encountered, the XML engine writes the completed input buffer to the SAS data set as a SAS observation.

4 The process is repeated for each <ITEM> start-tag and </ITEM> end-tag sequence until the end-of-file is encountered in the input stream or until the TABLE-END-PATH (if specified) is achieved, which results in six observations.

CAUTION:

Specifying the table location path, which is the observation boundary, can be tricky due to start-tag and end-tag pairing. The table location path determines which end tag causes the XML engine to write the completed input buffer to the SAS data set. If you do not identify the appropriate end tag, the result could be concatenated data instead of separate observations, or an unexpected set of columns. For examples, see "Determining the Observation Boundary to Avoid Concatenated Data" on page 62 and "Determining the Observation Boundary to Select the Best Columns" on page 64. △

TABLE-END-PATH syntax="*type*" beginend="BEGIN | END"

is an optional, optimization element that saves resources by stopping the processing of the XML document before the end of file. The location path tells the XML engine where in the XML document to locate and access a specific element in order to stop processing the XML document.

Default: Processing continues until the last end tag in the XML document.

Interaction: The TABLE-END-PATH element does not affect the observation boundary; that is determined with the TABLE-PATH element.

Tip: Specifying a location to stop processing is useful for an XML document that is hierarchical, but generally not appropriate for repeating instance data.

Featured in: "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 51

syntax="*type*"

is an optional attribute that specifies the type of syntax in the location path. The syntax is valid XPath construction in compliance with the W3C specifications. The XPath form supported by the XML engine allows elements and attributes to be individually selected for exclusion in the generated SAS data set.

Default: The default is XPath, that is, `syntax="XPATH"`.

Requirement: The XPath construction is a formal specification that puts a path description similar to UNIX on each element of the XML structure. *Note that XPath syntax is case sensitive. For example, if an element tag name is uppercase, it must be uppercase in the location path. If it is lowercase, it must be lowercase.* All location paths must begin with the root-enclosing element (denoted by a slash '/') or with the "any parent" variant (denoted by double slashes '//'). Other W3C documented forms are not currently supported.

Featured in: "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 51

beginend="BEGIN | END"

is an optional attribute that specifies to stop processing when either the element start tag is encountered or the element end tag is encountered.

Default: BEGIN

For example, using the XML document RSS.XML, which is used in the example "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 51, there is only one <CHANNEL> start-tag and one </CHANNEL> end-tag. With the TABLE-PATH location path,

```
<TABLE-PATH syntax="XPATH"> /rss/channel </TABLE-PATH>
```

the XML engine would process the entire XML document, even though it does not store new data in the input buffer after it encounters the first <ITEM> start tag, because the remaining elements no longer qualify. The following TABLE-END-PATH location path tells the XML engine to stop processing when the <ITEM> start tag is encountered:

```
<TABLE-END-PATH syntax="XPATH" beginend="BEGIN">
   /rss/channel/item </TABLE-END-PATH>
```

Therefore, with the two location path specifications, the XML engine processes only the highlighted data in the RSS.XML document for the CHANNEL data set, rather than the entire XML document:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
   <channel>
      <title>WriteTheWeb</title>
      <link>http://writetheweb.com</link>
      <description>News for web users that write back
         </description>
      <language>en-us</language>
      <copyright>Copyright 2000, WriteTheWeb team.
         </copyright>
      <managingEditor>editor@writetheweb.com
         </managingEditor>
      <webMaster>webmaster@writetheweb.com</webMaster>
      <image>
         <title>WriteTheWeb</title>
         <url>http://writetheweb.com/images/mynetscape88.gif
            </url>
         <link>http://writetheweb.com</link>
         <width>88</width>
         <height>31</height>
         <description>News for web users that write back
            </description>
         </image>
      <item>
         <title>Giving the world a pluggable Gnutella</title>
    <link>http://writetheweb.com/read.php?item=24</link>
      <description>WorldOS is a framework on which to build programs
          that work like Freenet or Gnutella-allowing distributed
          applications using  peer-to-peer routing.</description>
      </item>
      <item>
         .
         .
         .
   </channel>
</rss>
```

TABLE-DESCRIPTION

is an optional element that specifies a description for the SAS data set, which can be up to 256 characters. For example,

```
<TABLE-DESCRIPTION> Data Set contains TV channel
   information </TABLE-DESCRIPTION>
```

Elements for Columns

COLUMN name="*name*" retain="NO|YES" replace="NO|YES" ordinal="NO|YES" class="ORDINAL|FILENAME|FILEPATH"

is an element that contains a variable definition. For example,

```
<COLUMN name="Title">
```

Requirement: At least one COLUMN element is required.

Interaction: COLUMN can contain one or more of the following elements that describe the variable attributes: DATATYPE, DEFAULT, ENUM, FORMAT, INFORMAT, DESCRIPTION, LENGTH, TYPE, PATH, INCREMENT-PATH, DECREMENT-PATH, and RESET-PATH.

name="*name*"

specifies the name for the variable. The name must be a valid SAS name, which can be up to 32 characters.

Requirement: The name= attribute is required.

retain="NO|YES"

is an optional attribute that determines the contents of the input buffer at the beginning of each observation.

NO

sets the value for the beginning of each observation either to MISSING or to the value of the DEFAULT element if specified.

YES

keeps the current value until it is replaced by a new, nonmissing value. Specifying YES is much like the RETAIN statement in DATA step processing. It forces the retention of processed values after an observation is written to the output SAS data set.

Default: NO

Featured in: “Importing Hierarchical Data as Related Data Sets” on page 55

replace="NO|YES"

is an optional attribute that controls the concatenation of data.

NO

concatenates data until the end of the observation boundary.

YES

replaces existing content with new content. Specifying replace="YES" operates like the REPLACE statement in DATA step processing.

Default: NO

Tip: Concatenated data might be the result of an improperly identified end of the observation boundary, which is determined with the TABLE-PATH element.

ordinal="NO | YES"

is an optional attribute that determines whether the variable is a counter variable (similar to the _N_ automatic variable in DATA step processing). A counter variable keeps track of the number of times the location path, which is specified by the INCREMENT-PATH element or the DECREMENT-PATH element, is encountered. The counter variable increments its count by 1 each time the location path is encountered. Counter variables can be useful for identifying individual occurrences of like-named data elements, or for counting observations. The value for the ordinal= attribute determines which column location path to use for tracking the column's values.

Default: NO

Restriction: The ordinal= attribute is deprecated in SAS 9.2. This attribute will not be supported in future releases. This functionality is provided with the class="ORDINAL" attribute.

Requirement: For a counter variable, the TYPE element must specify the SAS data type as numeric, and the DATATYPE element must specify the type of data as integer.

NO

determines that the variable is not a counter variable. The PATH element is required. INCREMENT-PATH, DECREMENT-PATH, and RESET-PATH elements are not allowed.

YES

determines that the variable is a counter variable. Either the INCREMENT-PATH or the DECREMENT-PATH element is required. The RESET-PATH element is optional. The PATH element is not allowed.

class="ORDINAL | FILENAME | FILEPATH" XML92 Only

is an optional attribute that determines the type of variable.

ORDINAL

specifies that the variable is a numeric counter variable that keeps track of the number of times the location path, which is specified by the INCREMENT-PATH element or the DECREMENT-PATH element, is encountered. (This is similar to the _N_ automatic variable in DATA step processing.) The counter variable increments or decrements its count by 1 each time the location path is encountered. Counter variables can be useful for identifying individual occurrences of like-named data elements, or for counting observations.

Restriction: When exporting an XML document, variables with class="ORDINAL" are not included in the output XML document.

Requirement: You must use the INCREMENT-PATH element or the DECREMENT-PATH element. The PATH element is not allowed.

Requirement: The TYPE element must specify the SAS data type as numeric, and the DATATYPE element must specify the type of data as integer.

Tip: Specifying class="ORDINAL" is the same as specifying the COLUMN attribute ordinal="YES".

Featured in: "Including a Key Field with Generated Numeric Keys" on page 58

FILENAME

generates a character variable that contains the filename and extension of the input document. This functionality can be useful when you assign a libref for the XML engine that is associated with a physical location of a SAS library to determine which file contains a particular value.

Requirement: The TYPE element must specify the SAS data type as character, and the DATATYPE element must specify the type of data as string.

FILEPATH

generates a character variable that contains the pathname, filename, and extension of the input document. This functionality can be useful when you assign a libref for the XML engine that is associated with a physical location of a SAS library to determine which file contains a particular observation.

Requirement: The TYPE element must specify the SAS data type as character, and the DATATYPE element must specify the type of data as string.

TYPE

specifies the SAS data type (character or numeric) for the variable, which is how SAS stores the data. For example, the following specifies that the SAS data type for the variable is numeric:

```
<TYPE> numeric </TYPE>
```

Requirement: The TYPE element is required.

Tip: To assign a floating-point type, use

```
<DATATYPE> float </DATATYPE>
<TYPE> numeric </TYPE>
```

Tip: To apply output formatting in SAS, use the FORMAT on page 110 element.

Tip: To control data type conversion on input, use the INFORMAT on page 110 element.

```
<INFORMAT> datetime </INFORMAT>
```

DATATYPE

specifies the type of data being read from the XML document for the variable. For example, the following DATATYPE element specifies that the data contains alphanumeric characters:

```
<DATATYPE> string </DATATYPE>
```

Restriction: The values for previous versions of XMLMap syntax are not accepted by version 1.9.

Requirement: The DATATYPE element is required.

The type of data specification can be

string

specifies that the data contains alphanumeric characters and does not contain numbers used for calculations.

integer

specifies that the data contains whole numbers used for calculations.

double

specifies that the data contains floating-point numbers.

datetime

specifies that the input represents a valid datetime value, which is either

□ in the form of the XML specification ISO 8601 format. The default form is: **`yyyy-mm-ddThh:mm:ss[.nnnnnn]`**.

□ in a form for which a SAS informat (either supplied by SAS or user-written) properly translates the input into a valid SAS datetime value. See also the INFORMAT on page 110 element.

date

specifies that the input represents a valid date value, which is either

□ in the form of the XML specification ISO 8601 format. The default form is: **`yyyy-mm-dd`**.

□ in a form for which a SAS informat (either supplied by SAS or user-written) properly translates the input into a valid SAS date value. See also the INFORMAT on page 110 element.

time

specifies that the input represents a valid time value, which is either

□ in the form of the XML specification ISO 8601 format. The default form is: **`hh:mm:ss[.nnnnnn]`**.

□ in a form for which a SAS informat (either supplied by SAS or user-written) properly translates the input into a valid SAS date value. See also the INFORMAT on page 110 element.

DEFAULT

is an optional element that specifies a default value for a missing value for the variable. Use the DEFAULT element to assign a nonmissing value to missing data. For example, by including the following element, the XML engine assigns the value **`single`** when a missing value occurs:

```
<DEFAULT> single </DEFAULT>
```

Default: By default, the XML engine sets a missing value to MISSING.

Featured in: "Determining the Observation Boundary to Select the Best Columns" on page 64

ENUM

is an optional element that contains a list of valid values for the variable. The ENUM element can contain one or more VALUE elements in order to list the values. By using ENUM, values in the XML document are verified against the list of values. If a value is not valid, it is either set to MISSING (by default) or set to the value specified by the DEFAULT element. Note that a value specified for DEFAULT must be one of the ENUM values in order to be valid.

```
<COLUMN name="filing_status">
   .
   .
   .
   <DEFAULT> single </DEFAULT>
   .
   .
   .
   <ENUM>
      <VALUE> single </VALUE>
      <VALUE> married filing joint return </VALUE>
      <VALUE> married filing separate return </VALUE>
      <VALUE> head of household </VALUE>
      <VALUE> qualifying widow(er) </VALUE>
   </ENUM>
</COLUMN>
```

Featured in: "Determining the Observation Boundary to Select the Best Columns" on page 64

FORMAT width="*w*" ndec="*d*"

is an optional element that specifies a SAS format for the variable. A format name can be up to 31 characters for a character format and 32 characters for a numeric format. A SAS format is an instruction that SAS uses to write values. You use formats to control the written appearance of values. Do not include a period (.) as part of the format name. Specify a width and length as attributes, not as part of the format name.

For a list of the SAS formats, including the ISO 8601 SAS formats, see *SAS Language Reference: Dictionary*.

Featured in: "Determining the Observation Boundary to Select the Best Columns" on page 64

width="*w*"

is an optional attribute that specifies a format width, which for most formats is the number of columns in the output data.

ndec="*d*"

is an optional attribute that specifies a decimal scaling factor for numeric formats.

Here is an example:

```
<FORMAT> E8601DA </FORMAT>
<FORMAT width="8"> best </FORMAT>
<FORMAT width="8" ndec="2"> dollar </FORMAT>
```

INFORMAT width="*w*" ndec="*d*"

is an optional element that specifies a SAS informat for the variable. An informat name can be up to 30 characters for a character informat and 31 characters for a numeric informat. A SAS informat is an instruction that SAS uses to read values into a variable, that is, to store the values. Do not include a period (.) as part of the informat name. Specify a width and length as attributes, not as part of the informat name.

For a list of the SAS informats, including the ISO 8601 SAS informats, see *SAS Language Reference: Dictionary*.

Featured in: "Determining the Observation Boundary to Select the Best Columns" on page 64

width="*w*"

is an optional attribute that specifies an informat width, which for most informats is the number of columns in the input data.

ndec="*d*"

is an optional attribute that specifies a decimal scaling factor for numeric informats. SAS divides the input data by 10 to the power of this value.

Here is an example:

```
<INFORMAT> E8601DA </INFORMAT>
<INFORMAT width="8"> best </INFORMAT>
<INFORMAT width="8" ndec="2"> dollar </INFORMAT>
```

DESCRIPTION

is an optional element that specifies a description for the variable, which can be up to 256 characters. The following example shows that the description is assigned as the variable label.

```
<DESCRIPTION> Story link </DESCRIPTION>
```

LENGTH

is the maximum field storage length from the XML data for a character variable. The value refers to the number of bytes used to store each of the variable's values in the SAS data set. The value can be 1 to 32,767. During the input process, a maximum length of characters is read from the XML document and transferred to the observation buffer. For example,

```
<LENGTH> 200 </LENGTH>
```

Restriction: LENGTH is not valid for numeric data.

Requirement: For data that is defined as a STRING data type, the LENGTH element is required.

Tip: You can use LENGTH to truncate a long field.

PATH syntax="*type*"

specifies a location path that tells the XML engine where in the XML document to locate and access a specific tag for the current variable. In addition, the location path tells the XML engine to perform a function, which is determined by the location path form, to retrieve the value for the variable. The XPath forms that are supported allow elements and attributes to be individually included in the generated SAS data set.

Requirement: Whether the PATH element is required or allowed is determined by the class="ORDINAL" attribute for the COLUMN element on page 106. If the class="ORDINAL" attribute is not specified, which is the default, PATH is required and INCREMENT-PATH, DECREMENT-PATH, and RESET-PATH are not allowed. If the class="ORDINAL" attribute is specified, PATH is not allowed, INCREMENT-PATH or DECREMENT-PATH is required, and RESET-PATH is optional.

Featured in: "Specifying a Location Path on the PATH Element" on page 71

syntax="*type*"

is an optional attribute that specifies the type of syntax used in the location path. The syntax is valid XPath construction in compliance with the W3C specifications. The XPath form supported by the XML engine allows elements and attributes to be individually included in the generated SAS data set.

Default: XPATH; that is, **`syntax="XPATH"`**

Requirement: The XPath construction is a formal specification that puts a path description similar to UNIX on each element of the XML structure. *XPath syntax is case sensitive. For example, if an element tag name is uppercase, it must be uppercase in the location path. If it is lowercase, it must be lowercase in the location path.* All location paths must begin with the root-enclosing element (denoted by a slash '/'), or with the "any parent" variant (denoted by double slashes '//'). Other W3C-documented forms are not currently supported.

To specify the PATH location path, use one of the following forms:

CAUTION:

These forms are the only XPath forms that the XML engine supports. If you use any other valid W3C form, the results will be unpredictable. △

element-form

selects PCDATA (parsed character data) from a named element. The following element forms enable you to select from a named element, conditionally select from a named element based on a specific attribute value, or conditionally select from a named element based on a specific occurrence of the element using the position function:

```
<PATH> /LEVEL/ITEM </PATH>
<PATH> /LEVEL/ITEM[@attr="value"] </PATH>
<PATH> /LEVEL/ITEM[position()=n]|[n] </PATH>
```

The following examples illustrate the element forms. For more information about the examples, see "Specifying a Location Path on the PATH Element" on page 71.

□ The following location path tells the XML engine to scan the XML markup until it finds the CONFERENCE element. The XML engine retrieves the value between the <CONFERENCE> start tag and the </CONFERENCE> end tag.

```
<PATH> /NHL/CONFERENCE </PATH>
```

□ The following location path tells the XML engine to scan the XML markup until it finds the TEAM element where the value of the founded= attribute is 1993. The XML engine retrieves the value between the <TEAM> start tag and the </TEAM> end tag.

```
<PATH> /NHL/CONFERENCE/DIVISION/TEAM[@founded="1993"] </PATH>
```

□ The following location path uses the position function to tell the XML engine to scan the XML markup until it finds the fifth occurrence of the TEAM element. The XML engine retrieves the value between the <TEAM> start tag and the </TEAM> end tag.

```
<PATH> /NHL/CONFERENCE/DIVISION/TEAM[position()=5] </PATH>
```

You can use the following shorter version for the position function:

```
<PATH> /NHL/CONFERENCE/DIVISION/TEAM[5] </PATH>
```

attribute-form

selects values from an attribute. The following attribute forms enable you to select from a specific attribute, or conditionally select from a specific attribute based on the value of another attribute:

```
<PATH> /LEVEL/ITEM@attr </PATH>
<PATH> /LEVEL/ITEM@attr[attr2="value"] </PATH
```

The following examples illustrate the attribute forms. For more information about the examples, see "Specifying a Location Path on the PATH Element" on page 71.

- □ The following location path tells the XML engine to scan the XML markup until it finds the TEAM element. The XML engine retrieves the value from the abbrev= attribute.

```
<PATH syntax="XPATH"> /NHL/CONFERENCE/DIVISION/TEAM@abbrev </PATH>
```

- □ The following location path tells the XML engine to scan the XML markup until it finds the TEAM element. The XML engine retrieves the value from the founded= attribute where the value of the abbrev= attribute is ATL. The two attributes must be for the same element.

```
<PATH"> /NHL/CONFERENCE/DIVISION/TEAM@founded[@abbrev="ATL"] </PATH>
```

INCREMENT-PATH syntax="*type*" beginend="BEGIN | END"

specifies a location path for a counter variable, which is established by specifying the COLUMN on page 106 element attribute class="ORDINAL". The location path tells the XML engine where in the input data to increment the accumulated value for the counter variable by 1.

Requirement: If the variable is not a counter variable, PATH is required and INCREMENT-PATH and RESET-PATH are not allowed. If the variable is a counter variable, PATH is not allowed and either INCREMENT-PATH or DECREMENT-PATH is required.

Featured in: “Including a Key Field with Generated Numeric Keys” on page 58

syntax="*type*"

is an optional attribute that specifies the type of syntax in the location path. The syntax is valid XPath construction in compliance with the W3C specifications. The XPath form supported by the XML engine allows elements and attributes to be individually included in the generated SAS data set.

Default: The default is XPATH, that is, **`syntax="XPATH"`**.

Requirement: The XPath construction is a formal specification that puts a path description similar to UNIX on each element of the XML structure. *Note that XPath syntax is case sensitive. For example, if an element tag name is uppercase, it must be uppercase in the location path. If it is lowercase, it must be lowercase.* All location paths must begin with the root-enclosing element (denoted by a slash '/') or with the "any parent" variant (denoted by double slashes '//'). Other W3C documented forms are not currently supported.

beginend="BEGIN | END"

is an optional attribute that specifies to stop processing when either the element start tag is encountered or the element end tag is encountered.

Default: BEGIN

RESET-PATH syntax="*type*" beginend="BEGIN | END"

specifies a location path for a counter variable, which is established by specifying the COLUMN element attribute class="ORDINAL". The location path tells the XML engine where in the XML document to reset the accumulated value for the counter variable to 0.

Requirement: If the variable is not a counter variable, RESET-PATH is not allowed. If the variable is a counter variable, RESET-PATH is optional.

syntax="*type*"

is an optional attribute that specifies the type of syntax in the location path. The syntax is valid XPath construction in compliance with the W3C specifications. The XPath form supported by the XML engine allows elements and attributes to be individually included in the generated SAS data set.

Default: The default is XPATH, that is, `syntax="XPATH"`.

Requirement: The XPath construction is a formal specification that puts a path description similar to UNIX on each element of the XML structure. *Note that XPath syntax is case sensitive. For example, if an element tag name is uppercase, it must be uppercase in the location path. If it is lowercase, it must be lowercase.* All location paths must begin with the root-enclosing element (denoted by a slash '/') or with the "any parent" variant (denoted by double slashes '//'). Other W3C documented forms are not currently supported.

beginend="BEGIN | END"

is an optional attribute that specifies to stop processing when either the element start tag is encountered or the element end tag is encountered.

Default: BEGIN

DECREMENT-PATH syntax="*type*" beginend="BEGIN | END"

specifies a location path for a counter variable, which is established by specifying the COLUMN element attribute class="ORDINAL". The location path tells the XML engine where in the input data to decrement the accumulated value for the counter variable by 1.

Requirement: If the variable is not a counter variable, DECREMENT-PATH is not allowed. If the variable is a counter variable, either DECREMENT-PATH or INCREMENT-PATH is required.

syntax="*type*"

is an optional attribute that specifies the type of syntax in the location path. The syntax is valid XPath construction in compliance with the W3C specifications. The XPath form supported by the XML engine allows elements and attributes to be individually included in the generated SAS data set.

Default: XPATH; that is, `syntax="XPATH"`

Requirement: The XPath construction is a formal specification that puts a path description similar to UNIX on each element of the XML structure. *XPath syntax is case sensitive. For example, if an element tag name is uppercase, it must be uppercase in the location path. If it is lowercase, it must be lowercase in the location path.* All location paths must begin with the root-enclosing element (denoted by a slash '/'), or with the "any parent" variant (denoted by double slashes '//'). Other W3C-documented forms are not currently supported.

beginend="BEGIN | END"

is an optional attribute that specifies to stop processing when either the element start tag is encountered, or the element end tag is encountered.

Default: BEGIN

Using SAS XML Mapper to Generate and Update an XMLMap

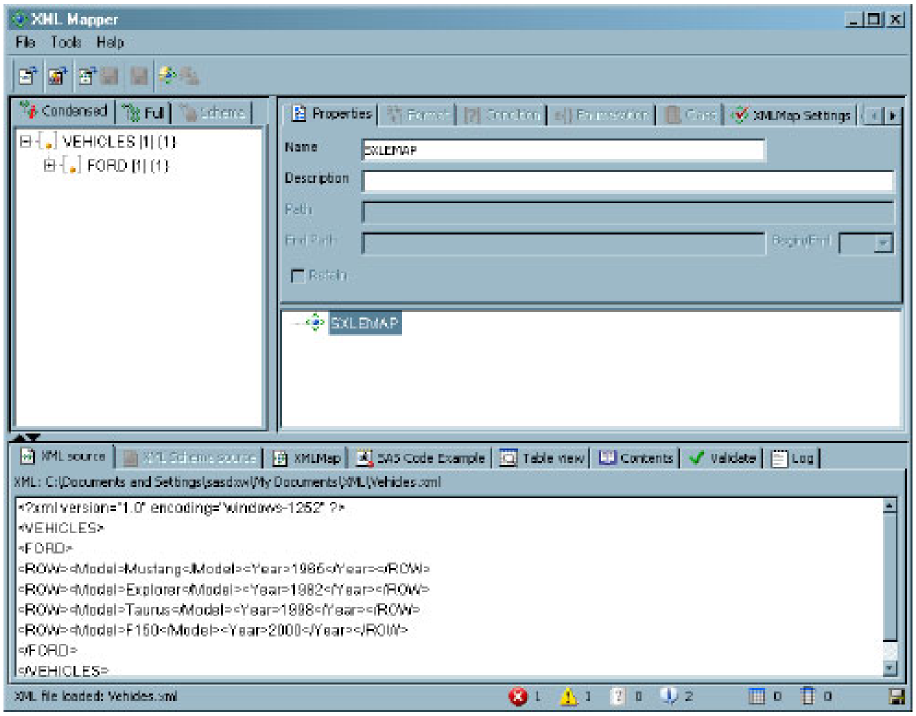
What Is SAS XML Mapper?

SAS XML Mapper is an XMLMap support tool for the XML engine. SAS XML Mapper is a Java-based, stand-alone application that removes the tedium when creating and modifying an XMLMap.

SAS XML Mapper provides a graphical interface that you can use to generate the appropriate XML elements. SAS XML Mapper analyzes the structure of an XML document or an XML schema and generates basic XML syntax for the XMLMap.

The interface consists of windows, a menu bar, and a toolbar. Using SAS XML Mapper, you can display an XML document or an XML schema, create and modify an XMLMap, and generate example SAS programs.

Display 8.1 SAS XML Mapper


XML Mapper
File Tools Help
Condensed Full Schema
VEHICLES [1] (1)
FORD [1] (1)
Properties Format Condition Enumeration Cases XMLMap Settings
Retain
SXLEMAP
XML source XML Schema source XMLMap SAS Code Example Table view Contents Validate Log
1 1 0 2 0 0

Name: SXLEMAP

Description:

Path:

End Path:

Begin/End:

XML: C:\Documents and Settings\sasdxw\My Documents\XML\Vehicles.xml

```
<?xml version="1.0" encoding="windows-1252" ?>
<VEHICLES>
<FORD>
<ROW><Model>Mustang</Model><Year>1965</Year></ROW>
<ROW><Model>Explorer</Model><Year>1982</Year></ROW>
<ROW><Model>Taurus</Model><Year>1998</Year></ROW>
<ROW><Model>F150</Model><Year>2000</Year></ROW>
</FORD>
</VEHICLES>
```

XML file loaded: Vehicles.xml

Using the Windows

The XML window and the XMLMap window are the two primary windows. The XML window, which is on the left, displays an XML document in a tree structure. The XMLMap window, which is on the right, displays an XMLMap in a tree structure. The map tree displays three layers: the top level is the map itself, the second tier includes tables, and the leaf nodes are columns. The detail area at the top displays information about the currently selected item, such as attributes for the table or column. The information is subdivided into tabs.

There are several source windows on the bottom of the interface, such as the XML source window, the XMLMap source window, the SAS code example window, and so on.

Using the Menu Bar

The menu bar provides menus in order to request functionality. For example, select the **File** menu, and then **Open XML** in order to display a browser so that you can select an XML document to open.

Using the Toolbar

The toolbar contains icons for shortcuts to several items on the menu bar. For example, the first icon from the left is the **Open an XML file** icon. Select it to display a browser so that you can select an XML document to open.

How Do I Get SAS XML Mapper?

SAS XML Mapper can be installed from the installation media for Windows and UNIX platforms, or it can be downloaded from the SAS Web site **http://www.sas.com/apps/demosdownloads/92_SDL_sysdep.jsp?packageID=000513**.

The latest version of SAS XML Mapper, which is SAS 9.2, can be downloaded and used with SAS 9.2 or with versions of SAS before SAS 9.2. There are some features that can be used only with SAS 9.2 XML Mapper, such as the 1.9 XMLMap version.

SAS XML Mapper has online Help attached, which includes usage examples. From the menu bar, select **Help**, and then **Help Topics**.

CHAPTER

9

SAS Data Set Options for the XML Engine

PARMS= Data Set Option

Specifies the input parameters SAS data set for importing Web service results.

Valid in: DATA step

Category: Data Set Control

Restriction: Use with the WSDL markup type only

See: "Importing Web Service Results Using the WSDL Markup Type (Preproduction)" on page 35

Syntax

PARMS=*input-parameters-dataset*

Syntax Description

input-parameters-dataset
the name of the SAS data set that contains the input parameters. This data set is submitted to the Web service.

Details

Specifying the WSDL markup type enables the XML engine to invoke a Web service and import the Web service results.

For an example of the PARMS= data set option, see "Importing Web Service Results" on page 36.

PART 3

Appendixes

APPENDIX 1

Sample XML Document

Example CDISC ODM Document

Here is an example of an XML document that is in CDISC ODM format. This document is used in “Importing a CDISC ODM Document” on page 32 and in “Exporting an XML Document in CDISC ODM Markup” on page 18.

```
<?xml version="1.0"  encoding=’’windows-1252’’ ?>
<!--
      Clinical Data Interchange Standards Consortium (CDISC)
      Operational Data Model (ODM) for clinical data interchange

      You can learn more about CDISC standards efforts at
      http://www.cdisc.org/standards/index.html
  -->

<ODM xmlns="http://www.cdisc.org/ns/odm/v1.2"
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.cdisc.org/ns/odm/v1.2 ODM1-2-0.xsd"

     ODMVersion="1.2"
     FileOID="000-00-0000"
     FileType="Snapshot"
     Description="Adverse events from the CTChicago file"

     AsOfDateTime="2005-05-18T14:01:41"
     CreationDateTime="2005-05-18T14:01:41"
     SourceSystem="SAS 9.1"
     SourceSystemVersion="9.01.01M3D05172005">

   <Study OID="STUDY.StudyOID">

      <!--
            GlobalVariables is a REQUIRED section in ODM markup
        -->
      <GlobalVariables>
         <StudyName>CDISC Connect-A-Thon Test Study III</StudyName>
         <StudyDescription>This file contains test data from a previous CDISC Connect-A-Thon.</StudyDescription>
         <ProtocolName>CDISC-Protocol-00-000</ProtocolName>
      </GlobalVariables>

      <BasicDefinitions />

      <!--
            Internal ODM markup required metadata
        -->
      <MetaDataVersion OID="v1.1.0" Name="Version 1.1.0">
         <Protocol>
            <StudyEventRef StudyEventOID="SE.VISIT1" OrderNumber="1" Mandatory="Yes" />
         </Protocol>

         <StudyEventDef OID="SE.VISIT1" Name="Study Event Definition" Repeating="Yes" Type="Common">
            <FormRef FormOID="FORM.AE" OrderNumber="1" Mandatory="No" />
         </StudyEventDef>

         <FormDef OID="FORM.AE" Name="Form Definition" Repeating="Yes">
            <ItemGroupRef ItemGroupOID="IG.AE" Mandatory="No" />
         </FormDef>



         <!--
               Columns defined in the table
           -->
         <ItemGroupDef OID="IG.AE" Repeating="Yes"
                       SASDatasetName="AE"
                       Name="Adverse Events"
                       Domain="AE"
                       Comment="Some adverse events from this trial">
            <ItemRef ItemOID="ID.TAREA"    OrderNumber="1"  Mandatory="No" />
            <ItemRef ItemOID="ID.PNO"      OrderNumber="2"  Mandatory="No" />
            <ItemRef ItemOID="ID.SCTRY"    OrderNumber="3"  Mandatory="No" />
            <ItemRef ItemOID="ID.F_STATUS" OrderNumber="4"  Mandatory="No" />
            <ItemRef ItemOID="ID.LINE_NO"  OrderNumber="5"  Mandatory="No" />
            <ItemRef ItemOID="ID.AETERM"   OrderNumber="6"  Mandatory="No" />
            <ItemRef ItemOID="ID.AESTMON"  OrderNumber="7"  Mandatory="No" />
            <ItemRef ItemOID="ID.AESTDAY"  OrderNumber="8"  Mandatory="No" />
            <ItemRef ItemOID="ID.AESTYR"   OrderNumber="9"  Mandatory="No" />
            <ItemRef ItemOID="ID.AESTDT"   OrderNumber="10" Mandatory="No" />
            <ItemRef ItemOID="ID.AEENMON"  OrderNumber="11" Mandatory="No" />
            <ItemRef ItemOID="ID.AEENDAY"  OrderNumber="12" Mandatory="No" />
            <ItemRef ItemOID="ID.AEENYR"   OrderNumber="13" Mandatory="No" />
            <ItemRef ItemOID="ID.AEENDT"   OrderNumber="14" Mandatory="No" />
            <ItemRef ItemOID="ID.AESEV"    OrderNumber="15" Mandatory="No" />
            <ItemRef ItemOID="ID.AEREL"    OrderNumber="16" Mandatory="No" />
            <ItemRef ItemOID="ID.AEOUT"    OrderNumber="17" Mandatory="No" />
            <ItemRef ItemOID="ID.AEACTTRT" OrderNumber="18" Mandatory="No" />
            <ItemRef ItemOID="ID.AECONTRT" OrderNumber="19" Mandatory="No" />
         </ItemGroupDef>



         <!--
               Column attributes as defined in the table
           -->
         <ItemDef OID="ID.TAREA"    SASFieldName="TAREA"    Name="Therapeutic Area"                      DataType="text"    Length="4">
            <CodeListRef CodeListOID="CL.$TAREAF" />
         </ItemDef>
         <ItemDef OID="ID.PNO"      SASFieldName="PNO"      Name="Protocol Number"                       DataType="text"    Length="15" />
         <ItemDef OID="ID.SCTRY"    SASFieldName="SCTRY"    Name="Country"                               DataType="text"    Length="4">
            <CodeListRef CodeListOID="CL.$SCTRYF" />
         </ItemDef>
         <ItemDef OID="ID.F_STATUS" SASFieldName="F_STATUS" Name="Record status, 5 levels, internal use" DataType="text"    Length="1">
            <CodeListRef CodeListOID="CL.$F_STATU" />
         </ItemDef>
         <ItemDef OID="ID.LINE_NO"  SASFieldName="LINE_NO"  Name="Line Number"                           DataType="integer" Length="2" />
         <ItemDef OID="ID.AETERM"   SASFieldName="AETERM"   Name="Conmed Indication"                     DataType="text"    Length="100" />
         <ItemDef OID="ID.AESTMON"  SASFieldName="AESTMON"  Name="Start Month - Enter Two Digits 01-12"  DataType="integer" Length="2" />
         <ItemDef OID="ID.AESTDAY"  SASFieldName="AESTDAY"  Name="Start Day - Enter Two Digits 01-31"    DataType="integer" Length="2" />
         <ItemDef OID="ID.AESTYR"   SASFieldName="AESTYR"   Name="Start Year - Enter Four Digit Year"    DataType="integer" Length="4" />
         <ItemDef OID="ID.AESTDT"   SASFieldName="AESTDT"   Name="Derived Start Date"                    DataType="date" />
         <ItemDef OID="ID.AEENMON"  SASFieldName="AEENMON"  Name="Stop Month - Enter Two Digits 01-12"   DataType="integer" Length="2" />
         <ItemDef OID="ID.AEENDAY"  SASFieldName="AEENDAY"  Name="Stop Day - Enter Two Digits 01-31"     DataType="integer" Length="2" />
         <ItemDef OID="ID.AEENYR"   SASFieldName="AEENYR"   Name="Stop Year - Enter Four Digit Year"     DataType="integer" Length="4" />
         <ItemDef OID="ID.AEENDT"   SASFieldName="AEENDT"   Name="Derived Stop Date"                     DataType="date" />
         <ItemDef OID="ID.AESEV"    SASFieldName="AESEV"    Name="Severity"                              DataType="text"    Length="1">
            <CodeListRef CodeListOID="CL.$AESEV" />
         </ItemDef>
         <ItemDef OID="ID.AEREL"    SASFieldName="AEREL"    Name="Relationship to study drug"            DataType="text"    Length="1">
            <CodeListRef CodeListOID="CL.$AEREL" />
         </ItemDef>
         <ItemDef OID="ID.AEOUT"    SASFieldName="AEOUT"    Name="Outcome"                               DataType="text"    Length="1">
            <CodeListRef CodeListOID="CL.$AEOUT" />
         </ItemDef>
         <ItemDef OID="ID.AEACTTRT" SASFieldName="AEACTTRT" Name="Actions taken re study drug"           DataType="text"    Length="1">
            <CodeListRef CodeListOID="CL.$AEACTTR" />
         </ItemDef>
         <ItemDef OID="ID.AECONTRT" SASFieldName="AECONTRT" Name="Actions taken, other"                  DataType="text"    Length="1">
            <CodeListRef CodeListOID="CL.$AECONTR" />
         </ItemDef>


         <!--
               Translation to ODM markup for any PROC FORMAT style
               user defined or SAS internal formatting specifications
               applied to columns in the table
           -->
         <CodeList OID="CL.$TAREAF" SASFormatName="$TAREAF" Name="$TAREAF" DataType="text">
            <CodeListItem CodedValue=’ONC’>
               <Decode>
                  <TranslatedText xml:lang="en">Oncology</TranslatedText>
               </Decode>
            </CodeListItem>
         </CodeList>

         <CodeList OID="CL.$SCTRYF" SASFormatName="$SCTRYF" Name="$SCTRYF" DataType="text">
            <CodeListItem CodedValue=’USA’>
               <Decode>
```

```
         <TranslatedText xml:lang="en">United States</TranslatedText>
      </Decode>
   </CodeListItem>
</CodeList>

<CodeList OID="CL.$F_STATU" SASFormatName="$F_STATU" Name="$F_STATU" DataType="text">
   <CodeListItem CodedValue=’S’>
      <Decode>
         <TranslatedText xml:lang="en">Source verified, not queried</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’V’>
      <Decode>
         <TranslatedText xml:lang="en">Source verified, queried</TranslatedText>
      </Decode>
   </CodeListItem>
</CodeList>

<CodeList OID="CL.$AESEV" SASFormatName="$AESEV" Name="$AESEV" DataType="text">
   <CodeListItem CodedValue=’1’>
      <Decode>
         <TranslatedText xml:lang="en">Mild</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’2’>
      <Decode>
         <TranslatedText xml:lang="en">Moderate</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’3’>
      <Decode>
         <TranslatedText xml:lang="en">Severe</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’4’>
      <Decode>
         <TranslatedText xml:lang="en">Life Threatening</TranslatedText>
      </Decode>
   </CodeListItem>
</CodeList>

<CodeList OID="CL.$AEREL" SASFormatName="$AEREL" Name="$AEREL" DataType="text">
   <CodeListItem CodedValue=’0’>
      <Decode>
         <TranslatedText xml:lang="en">None</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’1’>
      <Decode>
         <TranslatedText xml:lang="en">Unlikely</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’2’>
      <Decode>
```

```
         <TranslatedText xml:lang="en">Possible</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’3’>
      <Decode>
         <TranslatedText xml:lang="en">Probable</TranslatedText>
      </Decode>
   </CodeListItem>
</CodeList>

<CodeList OID="CL.$AEOUT" SASFormatName="$AEOUT" Name="$AEOUT" DataType="text">
   <CodeListItem CodedValue=’1’>
      <Decode>
         <TranslatedText xml:lang="en">Resolved, no residual effects</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’2’>
      <Decode>
         <TranslatedText xml:lang="en">Continuing</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’3’>
      <Decode>
         <TranslatedText xml:lang="en">Resolved, residual effects</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’4’>
      <Decode>
         <TranslatedText xml:lang="en">Death</TranslatedText>
      </Decode>
   </CodeListItem>
</CodeList>

<CodeList OID="CL.$AEACTTR" SASFormatName="$AEACTTR" Name="$AEACTTR" DataType="text">
   <CodeListItem CodedValue=’0’>
      <Decode>
         <TranslatedText xml:lang="en">None</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’1’>
      <Decode>
         <TranslatedText xml:lang="en">Discontinued permanently</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’2’>
      <Decode>
         <TranslatedText xml:lang="en">Reduced</TranslatedText>
      </Decode>
   </CodeListItem>
   <CodeListItem CodedValue=’3’>
      <Decode>
         <TranslatedText xml:lang="en">Interrupted</TranslatedText>
      </Decode>
   </CodeListItem>
```

```
      </CodeList>

      <CodeList OID="CL.$AECONTR" SASFormatName="$AECONTR" Name="$AECONTR" DataType="text">
         <CodeListItem CodedValue=’0’>
            <Decode>
               <TranslatedText xml:lang="en">None</TranslatedText>
            </Decode>
         </CodeListItem>
         <CodeListItem CodedValue=’1’>
            <Decode>
               <TranslatedText xml:lang="en">Medication required</TranslatedText>
            </Decode>
         </CodeListItem>
         <CodeListItem CodedValue=’2’>
            <Decode>
               <TranslatedText xml:lang="en">Hospitalization required or prolonged</TranslatedText>
            </Decode>
         </CodeListItem>
         <CodeListItem CodedValue=’3’>
            <Decode>
               <TranslatedText xml:lang="en">Other</TranslatedText>
            </Decode>
         </CodeListItem>
      </CodeList>
   </MetaDataVersion>
</Study>


<!--
      Administrative metadata
  -->
<AdminData />


<!--
      Clinical Data   : AE
                        Adverse Events
                        Some adverse events from this trial
  -->
<ClinicalData StudyOID="STUDY.StudyOID" MetaDataVersionOID="v1.1.0">
   <SubjectData SubjectKey="001">
      <StudyEventData StudyEventOID="SE.VISIT1" StudyEventRepeatKey="1">
         <FormData FormOID="FORM.AE" FormRepeatKey="1">
            <ItemGroupData ItemGroupOID="IG.AE" ItemGroupRepeatKey="1">
               <ItemData ItemOID="ID.TAREA" Value="ONC" />
               <ItemData ItemOID="ID.PNO" Value="143-02" />
               <ItemData ItemOID="ID.SCTRY" Value="USA" />
               <ItemData ItemOID="ID.F_STATUS" Value="V" />
               <ItemData ItemOID="ID.LINE_NO" Value="1" />
               <ItemData ItemOID="ID.AETERM" Value="HEADACHE" />
               <ItemData ItemOID="ID.AESTMON" Value="06" />
               <ItemData ItemOID="ID.AESTDAY" Value="10" />
               <ItemData ItemOID="ID.AESTYR" Value="1999" />
               <ItemData ItemOID="ID.AESTDT" Value="1999-06-10" />
```

```
                  <ItemData ItemOID="ID.AEENMON" Value="06" />
                  <ItemData ItemOID="ID.AEENDAY" Value="14" />
                  <ItemData ItemOID="ID.AEENYR" Value="1999" />
                  <ItemData ItemOID="ID.AEENDT" Value="1999-06-14" />
                  <ItemData ItemOID="ID.AESEV" Value="1" />
                  <ItemData ItemOID="ID.AEREL" Value="0" />
                  <ItemData ItemOID="ID.AEOUT" Value="1" />
                  <ItemData ItemOID="ID.AEACTTRT" Value="0" />
                  <ItemData ItemOID="ID.AECONTRT" Value="1" />
               </ItemGroupData>
               <ItemGroupData ItemGroupOID="IG.AE" ItemGroupRepeatKey="2">
                  <ItemData ItemOID="ID.TAREA" Value="ONC" />
                  <ItemData ItemOID="ID.PNO" Value="143-02" />
                  <ItemData ItemOID="ID.SCTRY" Value="USA" />
                  <ItemData ItemOID="ID.F_STATUS" Value="V" />
                  <ItemData ItemOID="ID.LINE_NO" Value="2" />
                  <ItemData ItemOID="ID.AETERM" Value="CONGESTION" />
                  <ItemData ItemOID="ID.AESTMON" Value="06" />
                  <ItemData ItemOID="ID.AESTDAY" Value="11" />
                  <ItemData ItemOID="ID.AESTYR" Value="1999" />
                  <ItemData ItemOID="ID.AESTDT" Value="1999-06-11" />
                  <ItemData ItemOID="ID.AEENMON" Value="" />
                  <ItemData ItemOID="ID.AEENDAY" Value="" />
                  <ItemData ItemOID="ID.AEENYR" Value="" />
                  <ItemData ItemOID="ID.AEENDT" Value="" />
                  <ItemData ItemOID="ID.AESEV" Value="1" />
                  <ItemData ItemOID="ID.AEREL" Value="0" />
                  <ItemData ItemOID="ID.AEOUT" Value="2" />
                  <ItemData ItemOID="ID.AEACTTRT" Value="0" />
                  <ItemData ItemOID="ID.AECONTRT" Value="1" />
               </ItemGroupData>
            </FormData>
         </StudyEventData>
      </SubjectData>
   </ClinicalData>
</ODM>
```

APPENDIX 2

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *The Little SAS Book: A Primer*
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- SAS Companion that is specific to your operating environment
- Base SAS focus area at **`support.sas.com/base`**
- For information about XML (Extensible Markup Language), see the Web site **`www.w3.org/XML`**

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **`sasbook@sas.com`**
Web address: **`support.sas.com/pubs`**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

encoding
a set of characters (letters, logograms, digits, punctuation, symbols, control characters, and so on) that have been mapped to numeric values (called code points) that can be used by computers. The code points are assigned to the characters in the character set by applying an encoding method. Some examples of encodings are wlatin1, wcyrillic, and shift-jis.

fileref (file reference)
a short name (or alias) for the full physical name of an external file. A SAS FILENAME statement maps the fileref to the full physical name.

libref (library reference)
a valid SAS name that serves as a shortcut name to associate with the physical location of an XML document.

markup language
a set of codes that are embedded in text in order to define layout and certain content.

metadata
a description or definition of data or information.

observation
a row in a SAS data set. All of the data values in an observation are associated with a single entity such as a customer or a state. Each observation contains either one data value or a missing-value indicator for each variable.

SAS data file
a type of SAS data set that contains data values as well as descriptor information that is associated with the data. The descriptor information includes information such as the data types and lengths of the variables, as well as the name of the engine that was used to create the data. See also SAS data set and SAS data view.

SAS data set
a file whose contents are in one of the native SAS file formats. There are two types of SAS data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats.

SAS data view

a type of SAS data set that retrieves data values from other files. A SAS data view contains only descriptor information such as the data types and lengths of the variables (columns) plus other information that is required for retrieving data values from other SAS data sets or from files that are stored in other software vendors' file formats. SAS data views can be created by the SAS DATA step and by the SAS SQL procedure.

SAS library
a collection of one or more files that are recognized by SAS and that are referenced and stored as a unit. Each file is a member of the library.

SAS XML LIBNAME engine
the SAS engine that processes XML documents. The engine exports an XML document from a SAS data set by translating the proprietary SAS file format to XML markup. The engine also imports an external XML document by translating XML markup to a SAS data set.

SAS XML Mapper
a graphical interface that you can use to create and modify XMLMaps for use by the SAS XML LIBNAME engine. The SAS XML Mapper analyzes the structure of an XML document and generates basic XML markup for the XMLMap.

variable
a column in a SAS data set or in a SAS data view. The data values for each variable describe a single characteristic for all observations.

XML (Extensible Markup Language)
a markup language that structures information by tagging it for content, meaning, or use. Structured information contains both content (for example, words or numbers) and an indication of what role the content plays. For example, content in a section heading has a different meaning from content in a database table.

XML engine
See SAS XML LIBNAME engine.

XMLMap
a file that contains XML tags that tell the SAS XML LIBNAME engine how to interpret an XML document.

Index

Your Turn

We welcome your feedback.

- □ If you have comments about this book, please send them to **yourturn@sas.com**. Include the full title and page numbers (if applicable).
- □ If you have comments about the software, please send them to **suggest@sas.com**.