## Contents

*What’s New in the SAS 9.4 XML LIBNAME Engine* ................................................. v

### PART 1 Usage  1

**Chapter 1 • Getting Started with the XML Engine** ........................................ 3
- What Does the XML LIBNAME Engine Do? .................................................. 3
- Understanding How the XML LIBNAME Engine Works .................................. 4
- SAS Processing Supported by the XML Engine ............................................. 5
- Transferring an XML Document across Environments .................................. 6
- Frequently Asked Questions ......................................................................... 6
- Accessibility Features of the XML LIBNAME Engine ..................................... 7

**Chapter 2 • Exporting XML Documents** ....................................................... 9
- Understanding How to Export an XML Document ....................................... 9
- Exporting an XML Document for Use by Oracle .......................................... 9
- Exporting an XML Document Containing SAS Dates, Times, and Datetimes ... 11
- Exporting Numeric Values ........................................................................... 12
- Exporting an XML Document with Separate Metadata ............................... 17
- Exporting an XML Document in CDISC ODM Markup ................................. 22

**Chapter 3 • Importing XML Documents** ..................................................... 23
- Understanding How to Import an XML Document ...................................... 23
- Importing an XML Document Using the GENERIC Markup Type .................. 23
- Importing an XML Document with Numeric Values .................................... 25
- Importing an XML Document with Non-Escaped Character Data .................. 27
- Importing an XML Document Created by Microsoft Access ......................... 29
- Importing Concatenated XML Documents ................................................... 33
- Importing a CDISC ODM Document ........................................................... 36

**Chapter 4 • Exporting XML Documents Using an XMLMap** ...................... 39
- Why Use an XMLMap when Exporting? ..................................................... 39
- Using an XMLMap to Export an XML Document with a Hierarchical Structure . 39

**Chapter 5 • Importing XML Documents Using an XMLMap** ....................... 43
- Why Use an XMLMap When Importing? .................................................... 43
- Understanding the Required Physical Structure for an XML
  Document to Be Imported Using the GENERIC Markup Type ......................... 44
- Using an XMLMap to Import an XML Document as One SAS Data Set ........... 47
- Using an XMLMap to Import an XML Document as Multiple SAS Data Sets .... 50
- Importing Hierarchical Data as Related Data Sets ....................................... 54
- Including a Key Field with Generated Numeric Keys .................................... 58
- Determining the Observation Boundary to Avoid Concatenated Data ............. 62
- Determining the Observation Boundary to Select the Best Columns ............... 64
- Using ISO 8601 SAS Informats and Formats to Import Dates .................... 67
- Using ISO 8601 SAS Informats and Formats to Import Time Values with a Time Zone . 68
- Referencing a Fileref Using the URL Access Method ................................... 71
Contents

Specifying a Location Path on the PATH Element ........................................... 71
Including Namespace Elements in an XMLMap ........................................... 74
Importing an XML Document Using the AUTOMAP= Option to
  Generate an XMLMap ............................................................................. 77

Chapter 6 • Understanding and Using Tagsets for the XML Engine ................ 83
What Is a Tagset? ......................................................................................... 83
Creating Customized Tagsets ..................................................................... 83
Exporting an XML Document Using a Customized Tagset ....................... 84

PART 2 LIBNAME Statement Reference 91

Chapter 7 • LIBNAME Statement: Overview ............................................. 93
  Using the LIBNAME Statement .................................................................. 93
  Understanding the XML LIBNAME Engine Versions: XML and XMLV2 ...... 93
  LIBNAME Statement Options .................................................................... 95

Chapter 8 • LIBNAME Statement Syntax .................................................. 97
  Dictionary ................................................................................................. 97

PART 3 XMLMap File Reference 111

Chapter 9 • XMLMap Syntax: Overview .................................................. 113
  Using XMLMap Syntax ........................................................................... 113
  Comparing the XMLMap Syntax ............................................................... 113

Chapter 10 • XMLMap Syntax Version 2.1 ................................................ 117
  Dictionary ................................................................................................. 117

Chapter 11 • Using SAS XML Mapper to Generate and Update an XMLMap .... 135
  What Is SAS XML Mapper? .................................................................... 135
  Using the Windows ................................................................................ 136
  Using the Menu Bar ............................................................................. 136
  Using the Toolbar ................................................................................ 137
  How Do I Get SAS XML Mapper? ........................................................... 137
  To Start SAS XML Mapper .................................................................... 137

PART 4 Appendixes 139

Appendix 1 • Example CDISC ODM Document ........................................ 141
  Recommended Reading ........................................................................ 147
  Glossary ................................................................................................. 149
  Index ...................................................................................................... 155
What’s New in the SAS 9.4 XML LIBNAME Engine

Overview

In SAS 9.4, XMLV2 functionality is no longer preproduction for the z/OS environment. XMLV2 functionality is production in all SAS 9.4 operating environments.

Enhanced LIBNAME Statement

The LIBNAME statement for the XMLV2 engine nickname supports the PREFIXATTRIBUTES= option to specify whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element. For more information, see the PREFIXATTRIBUTES= option on page 104.
SAS XML LIBNAME Engine
Part 1

Usage

Chapter 1
  Getting Started with the XML Engine .......................... 3

Chapter 2
  Exporting XML Documents ......................................... 9

Chapter 3
  Importing XML Documents ........................................ 23

Chapter 4
  Exporting XML Documents Using an XMLMap .................. 39

Chapter 5
  Importing XML Documents Using an XMLMap ................. 43

Chapter 6
  Understanding and Using Tagsets for the XML Engine ....... 83
What Does the XML LIBNAME Engine Do?

The XML LIBNAME 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 LIBNAME Engine Works

**Assigning a Libref**

The XML LIBNAME engine works much like other SAS engines. That is, you execute a LIBNAME statement 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 LIBNAME engine can be associated with either a specific XML document or, for the XMLV2 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:

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

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

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

**Output 1.1** DATASETS Procedure Output for MyXml Library

The PRINT procedure results in the following output:

```sas
proc print data=myxml.students;
run;
```
### 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:

```sas
libname myfiles 'C:\My Documents\';
libname myxml xml 'C:\My Documents\XML\Singers.xml';
```

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

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

### Output 1.3  XML Document Singers.XML

```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 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?

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

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 5, “Importing XML Documents Using an XMLMap,” on page 43.

What Are the XML Engine Nicknames?

SAS provides two versions of XML LIBNAME engine functionality by supporting the engine nicknames XML and XMLV2 in the LIBNAME statement. See “Understanding the XML LIBNAME Engine Versions: XML and XMLV2” on page 93.

Accessibility Features of the XML LIBNAME Engine

The XML LIBNAME engine is a command-based product. For this release, no features were added to address accessibility, but the product might be compliant to accessibility standards because it does not have a graphical user interface, and all of its features are available to anyone who can type or otherwise produce a command. If you have specific questions about the accessibility of SAS products, send them to accessibility@sas.com or call SAS Technical Support.
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.
The following SAS program exports an XML document from the SAS data set MyFiles.Class:

```sas
libname myfiles 'SAS-library';

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

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

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.
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.

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

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

data trans.test;
   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.

```
<?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, and then exports two XML documents to show the difference in output:

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

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

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

data format.dbltest;
  set npi;
run;

data prec.rawtest;
  set npi;
run;

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

title 'Keeps the Precision';
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, the PRINT procedure generates the XML document Format.XML, which contains numeric values controlled by the SAS format. See Output 2.5 on page 14.
7. For the PRINT procedure output, a format was specified to show the precision loss. In the output, the decimals after the second digit are zeros. See Output 2.6 on page 15.
From the data set Prec.RawTest, the PRINT procedure generates the XML document Precision.XML, which contains the stored numeric values. See Output 2.7 on page 16.

For the PRINT procedure output, a format was specified to show the retained precision. See Output 2.8 on page 17.

Output 2.5 XML Document Format.XML

```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>
```
### Output 2.6  PRINT Procedure Output for Format.DblTest

<table>
<thead>
<tr>
<th>Obs</th>
<th>N_PI</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.1400000000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6.2800000000</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9.4200000000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12.5700000000</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>15.7100000000</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>18.8500000000</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>21.9900000000</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>25.1300000000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>28.2700000000</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>31.4200000000</td>
<td>10</td>
</tr>
</tbody>
</table>
<?xml version="1.0" encoding="iso-8859-1" ?>
<TABLE>
    <RAWTEST>
        <n rawvalue="QRAAAAAAAAA=">1</n>
        <n_pi rawvalue="QTJD9qIWjA=">3.14</n_pi>
    </RAWTEST>
    <RAWTEST>
        <n rawvalue="QSAAAAAAAAA=">2</n>
        <n_pi rawvalue="QWSH7VEQCGA=">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>
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.

First, here is the CONTENTS procedure output for the SAS data set Input.Suppliers:
The following SAS program exports an XML document from the SAS data set Input.Suppliers:

```sas
libname input 'C:\My Documents\myfiles';
filename xsd 'C:\My Documents\XML\suppliers.xsd';
libname output xml 'C:\My Documents\XML\suppliers.xml' xmltype=msaccess xmlmeta=schemadata xmlschema=xsd;
data output.suppliers;
  set input.suppliers;
run;
```

The first LIBNAME statement assigns the libref Input to the physical location of the SAS library that stores the SAS data set Suppliers.

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

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.

Part of the resulting XML document is shown in Output 2.10 on page 19. The separate metadata information is shown in Output 2.11 on page 20.

Output 2.10  XML Document Suppliers.XML

```xml
<?xml version="1.0" encoding="windows-1252" ?>
<dataroot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
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>
```
<?xml version="1.0" encoding="windows-1252" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:od="urn:schemas-microsoft-com:officedata">
    <xsd:element name="dataroot">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="SUPPLIERS" minOccurs="0" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="SUPPLIERS">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="HOMEPAGE" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
                    <xsd:simpleType>
                        <xsd:restriction base="xs:string">
                            <xsd:maxLength value="94" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="FAX" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
                    <xsd:simpleType>
                        <xsd:restriction base="xs:string">
                            <xsd:maxLength value="15" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="PHONE" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
                    <xsd:simpleType>
                        <xsd:restriction base="xs:string">
                            <xsd:maxLength value="15" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="COUNTRY" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
                    <xsd:simpleType>
                        <xsd:restriction base="xs:string">
                            <xsd:maxLength value="11" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="POSTALCODE" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
                    <xsd:simpleType>
                        <xsd:restriction base="xs:string">
                            <xsd:maxLength value="8" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
                <xsd:element name="REGION" minOccurs="0" od:jetType="text" od:sqlSType="nvarchar">
                    <xsd:simpleType>
                        <xsd:restriction base="xs:string">
                            <xsd:maxLength value="8" />
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
<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 is imported in “Importing a CDISC ODM Document” on page 36 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:

```sas
filename output 'C:\myoutput.xml';
libname output xml xmltype=CDISCODM formatactive=yes;
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 uses 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 Appendix 1, “Example CDISC ODM Document,” on page 141.
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 44.

```xml
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
  <CLASS>
    <Name> Alfred </Name>
    <Gender> M </Gender>
  </CLASS>
</TABLE>
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
<th>Height</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>F</td>
<td>13</td>
<td>56.5</td>
<td>84</td>
</tr>
<tr>
<td>William</td>
<td>M</td>
<td>15</td>
<td>66.5</td>
<td>112</td>
</tr>
</tbody>
</table>

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

```sas
libname trans xml 'XML-document';
libname myfiles 'SAS-library';
data myfiles.class;
  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:

```sas
proc print data=myfiles.class;
run;
```
Importing an XML Document with Numeric Values

This example imports the XML document Precision.XML, which was exported in “Exporting Numeric Values” on page 12. 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 parsed character data (PCDATA) from the element:

```sas
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  PRINT Procedure Output for Default.RawTest

```
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 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
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
<?xml version="1.0" ?>
<PERMIT>
  <CHARS>
    <accept>OK</accept>
  </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:

```sas
libname permit xmlv2 'C:\My Documents\XML\permit.xml';
proc print data=permit.chars;
run;
```

**Log 3.1  SAS Log Output**

```
ERROR: There is an illegal character in the entity name.
encountered during XMLInput parsing,
ocurred 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 xmlv2 'C:\My Documents\XML\permit.xml' xmlprocess=permit;

proc print data=permit.chars;
run;

Output 3.4  PRINT Procedure Output for PermitChars

---

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>accept</th>
<th>status</th>
<th>ampersand</th>
<th>squote</th>
<th>dquote</th>
<th>less</th>
<th>greater</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OK</td>
<td>proper escape sequence</td>
<td>&amp;</td>
<td>'</td>
<td>&quot;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>2</td>
<td>OK</td>
<td>unescaped character in CDATA</td>
<td>Abbott &amp; Costello</td>
<td>Logan's Run</td>
<td>This is &quot;realworld&quot; stuff</td>
<td>e &lt; pi</td>
<td>pen &gt; sword</td>
</tr>
<tr>
<td>3</td>
<td>NO</td>
<td>single unescaped character</td>
<td>&amp;</td>
<td>'</td>
<td>&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>NO</td>
<td>unescaped character in string</td>
<td>Dunn &amp; Bradstreet</td>
<td>Isn't this silly?</td>
<td>Quoth the raven, &quot;Nevermore!&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

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
<?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">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="HOMEPAGE" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</root>
```
Importing an XML Document Created by Microsoft Access 31

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SUPPLIERS" type="xs:complexType">
    <xs:complexType>
      <xs:sequence>
        <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>
</dataroot>
The following SAS program interprets the XML document as a SAS data set:

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

proc print data=access.suppliers (obs=2);
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 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 3.5** PRINT Procedure Output for Access.Suppliers

```
<table>
<thead>
<tr>
<th>Obs</th>
<th>CONTACTNAME</th>
<th>COMPANYNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Charlotte Smith</td>
<td>Exotic Flowers</td>
</tr>
<tr>
<td>2</td>
<td>Shelley Martin</td>
<td>New Orleans Cajun Foods</td>
</tr>
</tbody>
</table>
```

Using the CONTENTS procedure, 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;
```
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
<?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:

```sas
libname concat xml '/u/My Documents/XML/ConcatStudents.xml';
proc datasets library=concat;
```
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 3.7**  PRINT Procedure Output for Concat.Students

<table>
<thead>
<tr>
<th></th>
<th>CITY</th>
<th>SCHOOL</th>
<th>NAME</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Houston</td>
<td>Bellaire</td>
<td>Linda Kay</td>
<td>1345</td>
</tr>
<tr>
<td>2</td>
<td>Houston</td>
<td>Sam Houston</td>
<td>Chas Wofford</td>
<td>2456</td>
</tr>
<tr>
<td>3</td>
<td>Houston</td>
<td>Sharpstown</td>
<td>Jerry Kolar</td>
<td>3667</td>
</tr>
<tr>
<td>4</td>
<td>Austin</td>
<td>Reagan</td>
<td>Brad Martin</td>
<td>1234</td>
</tr>
<tr>
<td>5</td>
<td>Austin</td>
<td>Westwood</td>
<td>Zac Havell</td>
<td>2345</td>
</tr>
<tr>
<td>6</td>
<td>Austin</td>
<td>Bowie</td>
<td>Walter Smith</td>
<td>3456</td>
</tr>
</tbody>
</table>
Importing a CDISC ODM Document

This example imports the XML document that is shown in Appendix 1, “Example CDISC ODM Document,” on page 141. 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= option, FORMATLIBRARY= option, and FORMATNOREPLACE= option.

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

```sas
filename odm 'C:\Documents and Settings\myid\My Documents\CDISC\AE.XML';
libname odm xml xmltype=CDISCODM FormatActive=YES FormatNoReplace=NO FormatLibrary="Work";
proc contents data=odm.AE varnum;
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 Odm to reference the XML document and specifies the XML engine. If the fileref matches the libref, you do not need to specify the physical location of the XML document in the LIBNAME statement. By default, the XML engine expects GENERIC markup, so you must include the XMLTYPE= option 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 output from the CONTENTS procedure 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.
### Output 3.8  CONTENTS Procedure Output for Odm.AE

#### Data Set Name: ODM.AE

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>ODM.AE</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
</tr>
<tr>
<td>Engine</td>
<td>JOL</td>
<td>Indexes</td>
</tr>
<tr>
<td>Created</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Last Modified</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Bonded</td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td></td>
<td>Default</td>
</tr>
<tr>
<td>Encoding</td>
<td></td>
<td>Default</td>
</tr>
</tbody>
</table>

#### Variables in Creation Order

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Informat</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>_STUDYID</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_STUDYID</td>
</tr>
<tr>
<td>2</td>
<td>_METADATAVERSIONID</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_METADATAVERSIONID</td>
</tr>
<tr>
<td>3</td>
<td>_SUBJECTKEY</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_SUBJECTKEY</td>
</tr>
<tr>
<td>4</td>
<td>_STUDYEVENTID</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_STUDYEVENTID</td>
</tr>
<tr>
<td>5</td>
<td>_STUDYEVENTREPEATKEY</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_STUDYEVENTREPEATKEY</td>
</tr>
<tr>
<td>6</td>
<td>_FORMID</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_FORMID</td>
</tr>
<tr>
<td>7</td>
<td>_FORMREPEATKEY</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_FORMREPEATKEY</td>
</tr>
<tr>
<td>8</td>
<td>_TBGROUPOID</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_TBGROUPOID</td>
</tr>
<tr>
<td>9</td>
<td>_TBGROUPREPEATKEY</td>
<td>Char</td>
<td>100</td>
<td></td>
<td></td>
<td>_TBGROUPREPEATKEY</td>
</tr>
<tr>
<td>10</td>
<td>TAREA</td>
<td>Char</td>
<td>4</td>
<td>STAREA</td>
<td>Therapeutic Area</td>
<td>Therapeutic Area</td>
</tr>
<tr>
<td>11</td>
<td>PNO</td>
<td>Char</td>
<td>15</td>
<td></td>
<td>Protocol Number</td>
<td>Protocol Number</td>
</tr>
<tr>
<td>12</td>
<td>SITE</td>
<td>Char</td>
<td>4</td>
<td>SITE</td>
<td>Country</td>
<td>Country</td>
</tr>
<tr>
<td>13</td>
<td>E_STOP</td>
<td>Char</td>
<td>1</td>
<td>E_STOP</td>
<td>Record status, 5 levels, Internal use</td>
<td>Record status, 5 levels, Internal use</td>
</tr>
<tr>
<td>14</td>
<td>LINE_NO</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>2</td>
<td>Line Number</td>
</tr>
<tr>
<td>15</td>
<td>AETERMII</td>
<td>Char</td>
<td>100</td>
<td></td>
<td>Conmed indication</td>
<td>Conmed indication</td>
</tr>
<tr>
<td>16</td>
<td>AESTHON</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>2</td>
<td>Start Month - Enter Two Digits 01-12</td>
</tr>
<tr>
<td>17</td>
<td>AESTDAY</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>2</td>
<td>Start Day - Enter Two Digits 01-31</td>
</tr>
<tr>
<td>18</td>
<td>AESTYR</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>4</td>
<td>Start Year - Enter Four Digits Year</td>
</tr>
<tr>
<td>19</td>
<td>AESTOT</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>DATE</td>
<td>Derived Start Date</td>
</tr>
<tr>
<td>20</td>
<td>AESTMED</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>2</td>
<td>Stop Month - Enter Two Digits 01-12</td>
</tr>
<tr>
<td>21</td>
<td>AESTDAY</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>2</td>
<td>Stop Day - Enter Two Digits 01-31</td>
</tr>
<tr>
<td>22</td>
<td>AESTYR</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>4</td>
<td>Stop Year - Enter Four Digits Year</td>
</tr>
<tr>
<td>23</td>
<td>AESTOT</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>DATE</td>
<td>Derived Stop Date</td>
</tr>
<tr>
<td>24</td>
<td>ASEEV</td>
<td>Char</td>
<td>1</td>
<td>ASEEV</td>
<td>Severity</td>
<td>Severity</td>
</tr>
<tr>
<td>25</td>
<td>AERELE</td>
<td>Char</td>
<td>1</td>
<td>AERELE</td>
<td>Relationship to study drug</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>AEROUT</td>
<td>Char</td>
<td>1</td>
<td>AEROUT</td>
<td>Outcome</td>
<td>Outcome</td>
</tr>
<tr>
<td>27</td>
<td>AECSTRTRT</td>
<td>Char</td>
<td>1</td>
<td>AECSTRTRT</td>
<td>Actions taken re study drug</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>AECSTRATR</td>
<td>Char</td>
<td>1</td>
<td>AECSTRATR</td>
<td>Actions taken, other</td>
<td></td>
</tr>
</tbody>
</table>
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 set back into the specific XML document structure.

To export an XML document using an XMLMap, specify the XML engine nickname XMLV2 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 47.

First, here is the SAS data set named Nhl.Teams to be exported:
Output 4.1 PRINT Procedure Output of Data Set Nhl.Teams

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
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
  <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>
  ...
</TABLE>
```

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.
To use an XMLMap to export the SAS data set as an XML document, you must specify 1.9 or 2.1 as the XMLMap version number.

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.

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

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:

```sas
<OUTPUT>
  <HEADING>
    <ATTRIBUTE name="description" value="Teams of the National Hockey League" />
  </HEADING>
  <TABLEREF name="TEAMS" />
</OUTPUT>

<TABLE name="TEAMS">
  <TABLE-PATH syntax="XPath">/NHL/CONFERENCE/DIVISION/TEAM</TABLE-PATH>

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

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

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

  <COLUMN name="DIVISION" retain="YES">
    <PATH syntax="XPath">/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 or 2.1 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.
libname nhl 'C:\My Documents\myfiles';
filename out 'C:\My Documents\XML\nhlout.xml';
libname out xmlv2 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 -->
<!-- Created 2012-09-21T13:47:16 -->

<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>
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).

As an alternative to you creating an XMLMap by coding XMLMap syntax, the SAS XML Mapper can generate XMLMap syntax. SAS XML Mapper removes the tedium of creating and modifying an XMLMap by providing a GUI that generates the appropriate XML elements for you. SAS XML Mapper analyzes the structure of an XML document or an XML schema, and generates basic syntax for the XMLMap. See Chapter 11, “Using SAS XML Mapper to Generate and Update an XMLMap,” on page 135.

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

The AUTOMAP= option in the LIBNAME statement provides another alternative to creating an XMLMap. AUTOMAP= specifies to automatically generate an XMLMap file to import an XML document. See the AUTOMAP= option on page 100.

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

What Is the Required Physical Structure?

In order 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
<?xml version="1.0" encoding="windows-1252" ?>
<LIBRARY>
  <STUDENTS>
    <ID> 0755 </ID>
    <NAME> Brad Martin </NAME>
    <ADDRESS> 1611 Glengreen </ADDRESS>
    <CITY> Huntsville </CITY>
    <STATE> Texas </STATE>
  </STUDENTS>
  <STUDENTS>
    <ID> 1522 </ID>
    <NAME> Zac Harvell </NAME>
    <ADDRESS> 11900 Glenda </ADDRESS>
    <CITY> Houston </CITY>
    <STATE> Texas </STATE>
  </STUDENTS>
</LIBRARY>
```
Understanding the Required Physical Structure for an XML Document to Be Imported

Using the GENERIC Markup Type

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\students.xml';

proc print data=test.students;
run;

Output 5.1 PRINT Procedure Output for Test.Students

<table>
<thead>
<tr>
<th>Obs</th>
<th>STATE</th>
<th>CITY</th>
<th>ADDRESS</th>
<th>NAME</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Texas</td>
<td>Huntsville</td>
<td>1611 Glengreen</td>
<td>Brad Martin</td>
<td>755</td>
</tr>
<tr>
<td>2</td>
<td>Texas</td>
<td>Houston</td>
<td>11900 Glenda</td>
<td>Zac Harvell</td>
<td>1522</td>
</tr>
</tbody>
</table>

Why Is a Specific Physical Structure Required?

Well-formed XML is determined by structure, not content. Therefore, although 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
<?xml version="1.0" encoding="windows-1252" ?>
<CLIMATE>
  <HIGHTEMP>
    <PLACE> Libya </PLACE>
    <DATE> 1922-09-13 </DATE>
    <DEGREE-F> 136 </DEGREE-F>
  </HIGHTEMP>
</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 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:

```sas
libname climate xml 'C:\My Documents\climate.xml';
proc datasets library=climate;
quit;
```
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 to successfully import the document. See “Why Use an XMLMap When Importing?” on page 43.

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.

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
<?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>
</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
<?xml version="1.0" ?>
<SXLEMAP version="2.1">
  <TABLE name="TEAMS">
    <TABLE-PATH syntax="XPath">
      /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>
    </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 50.)

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:

```sas
filename nhl 'C:\My Documents\XML\Nhl.xml';
filename map 'C:\My Documents\XML\Nhl.map';
libname nhl xmlv2 xmlmap=map;
proc print data=nhl.teams;
run;
```
1. The first FILENAME statement assigns the file reference Nhl to the physical location (complete pathname, filename, and file extension) of the XML document named Nhl.xml.

2. The second FILENAME statement assigns the file reference Map to the physical location of the XMLMap named Nhl.map.

3. The LIBNAME statement uses the file reference Nhl to reference the XML document. It specifies the XMLV2 engine and uses the file reference Map to reference the XMLMap.

4. The PRINT procedure produces output, verifying that the import was successful.

**Output 5.3** PRINT Procedure Output for Nhl.Teams

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th>ABBREV</th>
<th>CONFERENCE</th>
<th>DIVISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thrashers</td>
<td>ATL</td>
<td>Eastern</td>
<td>Southeast</td>
</tr>
<tr>
<td>2</td>
<td>Hurricanes</td>
<td>CAR</td>
<td>Eastern</td>
<td>Southeast</td>
</tr>
<tr>
<td>3</td>
<td>Panthers</td>
<td>FLA</td>
<td>Eastern</td>
<td>Southeast</td>
</tr>
<tr>
<td>4</td>
<td>Lightning</td>
<td>TB</td>
<td>Eastern</td>
<td>Southeast</td>
</tr>
<tr>
<td>5</td>
<td>Capitals</td>
<td>WSH</td>
<td>Eastern</td>
<td>Southeast</td>
</tr>
<tr>
<td>6</td>
<td>Stars</td>
<td>DAL</td>
<td>Western</td>
<td>Pacific</td>
</tr>
<tr>
<td>7</td>
<td>Kings</td>
<td>LA</td>
<td>Western</td>
<td>Pacific</td>
</tr>
<tr>
<td>8</td>
<td>Ducks</td>
<td>ANA</td>
<td>Western</td>
<td>Pacific</td>
</tr>
<tr>
<td>9</td>
<td>Coyotes</td>
<td>PHX</td>
<td>Western</td>
<td>Pacific</td>
</tr>
<tr>
<td>10</td>
<td>Sharks</td>
<td>SJ</td>
<td>Western</td>
<td>Pacific</td>
</tr>
</tbody>
</table>

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.

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'Reilly Network weblog, I argue that the next generation of RSS should be based on RDF.</description>
    </item>
    - <item>
      <title>UK bloggers get organised</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's now a UK-specific page on weblogs.com, and a mailing list at egroups.</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
<?xml version="1.0" encoding="UTF-8"?>
<SXLEMAP name="SXLEMap" version="2.1"> 1
   <TABLE name="CHANNEL"> 2
      <TABLE-PATH syntax="XPath">/rss/channel</TABLE-PATH> 3
      <COLUMN name="title"> 4
         <PATH syntax="XPath">/rss/channel/title</PATH>
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>200</LENGTH>
      </COLUMN>
      <COLUMN name="link"> 5
         <PATH syntax="XPath">/rss/channel/link</PATH>
         <DESCRIPTION>Story link</DESCRIPTION>
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>200</LENGTH>
      </COLUMN>
      <COLUMN name="description"> 6
         <PATH syntax="XPath">/rss/channel/description</PATH>
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>1024</LENGTH>
      </COLUMN>
      <COLUMN name="language"> 7
         <PATH syntax="XPath">/rss/channel/language</PATH>
         <TYPE>character</TYPE>
         <DATATYPE>string</DATATYPE>
         <LENGTH>8</LENGTH>
      </COLUMN>
      <COLUMN name="version"> 8
         <PATH syntax="XPath">/rss/version</PATH>
         <TYPE>character</TYPE>
      </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 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.
5. Subsequent COLUMN elements define the variables Link, Description, and Language for the Channel data set.
6. 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.
7. Element for the Items data set definition.
8. Element containing the attributes for the Title variable in the Items data set.
9. 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.

```sas
filename rss 'C:\My Documents\rss.xml';
filename map 'C:\My Documents\rss.map';
libname rss xmlv2 xmlmap=map access=readonly;
proc datasets library=rss;
run;
quit;
```

**Output 5.4**  DATASETS Procedure Output for RSS Library Showing Two Data Sets

---

### 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. 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.

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
<?xml version="1.0" ?>
```
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 Person data set imports the name and address of each customer, and the Prescription data set 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.
SXLEMAP is the root-enclosing element for the two SAS data set definitions.

First TABLE element defines the Person data set.

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

Second TABLE element defines the Prescription data set.
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.)

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:

```sas
filename pharm 'C:\My Documents\Pharmacy.xml';
filename map 'C:\My Documents\Pharmacy.map';
libname pharm xmlv2 xmlmap=map;
```

The DATASETS procedure verifies that SAS interprets the XML document Pharmacy.xml as two SAS data sets: Pharm.Person and Pharm.Prescription.

```sas
proc datasets library=pharm;
quit;
```

Output 5.5 DATASETS Procedure Output for Pharm Library

<table>
<thead>
<tr>
<th>The SAS System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory</td>
</tr>
<tr>
<td>Libref</td>
</tr>
<tr>
<td>Engine</td>
</tr>
<tr>
<td>Physical Name</td>
</tr>
<tr>
<td>XMLType</td>
</tr>
<tr>
<td>Version</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Member Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PERSON</td>
<td>DATA</td>
</tr>
<tr>
<td>2</td>
<td>PRESCRIPTION</td>
<td>DATA</td>
</tr>
</tbody>
</table>

Here is the PRINT procedure output for both of the imported SAS data sets.
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 54. This example continues with the XMLMap by adding a key field with generated numeric key values to provide a relationship between the two data sets. (A key field holds unique data 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 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
<?xml version="1.0" encoding="UTF-8"?>
<!-- 2011-01-10T14:39:38 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 903000.1.0.20101208190000_v930 -->
<!-- ### Validation report -->
<!-- XMLMap validation completed successfully. -->
<SXLEMAP name="AUTO_GEN" version="2.1">
  <NAMESPACES count="0"/>
  <TABLE description="PERSON" name="PERSON">
    <TABLE-PATH syntax="XPath">/PHARMACY/PERSON</TABLE-PATH>
    <COLUMN name="KEY" retain="YES" class="ORDINAL">
      <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 description="PRESCRIPTION" name="PRESCRIPTION"/>
</SXLEMAP>
```

Including a Key Field with Generated Numeric Keys
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 to generate the key field values for the Person data set:
   1. 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.
   2. 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.
   3. The process is repeated for each <PERSON> start tag (from INCREMENT-PATH) and </PERSON> end tag (from TABLE-PATH) sequence.
   4. 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 to generate the key field values for the Prescription data set:
   1. 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.
2. 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. 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 data set Key variable is the same as the Person data set Key variable. Although 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.

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

4. The result is three variables and three observations.

The following SAS statements import the XML document:

```sas
filename pharm 'C:\My Documents\XML\Pharmacy.xml';
filename map 'C:\My Documents\XML\PharmacyOrdinal.map';
libname pharm xmlv2 xmlmap=map;
```

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

**Output 5.8  PRINT Procedure Output for Pharm.Person**

<table>
<thead>
<tr>
<th>Obs</th>
<th>KEY</th>
<th>NAME</th>
<th>STREET</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>001</td>
<td>Brad Martin</td>
<td>11900 Glenda Court</td>
<td>Austin</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>Jim Spano</td>
<td>1611 Glengreen</td>
<td>Austin</td>
</tr>
</tbody>
</table>

**Output 5.9  PRINT Procedure Output for Pharm.Prescription**

<table>
<thead>
<tr>
<th>Obs</th>
<th>KEY</th>
<th>NUMBER</th>
<th>DRUG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>001</td>
<td>1234</td>
<td>Tetracycline</td>
</tr>
<tr>
<td>2</td>
<td>001</td>
<td>1245</td>
<td>Lomotil</td>
</tr>
<tr>
<td>3</td>
<td>002</td>
<td>1268</td>
<td>Nexium</td>
</tr>
</tbody>
</table>
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 to import the file successfully. Without an XMLMap, the XML engine would import a data set named Ford with variables Row0, Model0, Year0, Row1, Model1, Year1, and so on.

```xml
<?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:

```xml
<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.

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="2.1" 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.
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 the columns Title, Acquired, and 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 the columns Title, Acquired, Topic, and Major.

```xml
<?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>
</Library>
```
Here is the XMLMap syntax to use in order to import the previous XML document:

```xml
<?xml version="1.0" ?>
<SXLEMAP version="1.2">
  <TABLE name="Publication">
    <TABLE-PATH syntax="XPath">
      /Library/Publication/Topic
    </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>
      <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>
        <VALUE>Y</VALUE>
        <VALUE>N</VALUE>
      </ENUM>
    </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 value 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 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 5.11 PRINT Procedure Output for Rep.Publication Data Set*

```
<table>
<thead>
<tr>
<th>Title</th>
<th>Acquired</th>
<th>Topic</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer's Almanac</td>
<td>12/11/2000</td>
<td>JAVA</td>
<td>Y</td>
</tr>
<tr>
<td>Inside Visual C++</td>
<td>06/19/1998</td>
<td>C</td>
<td>Y</td>
</tr>
<tr>
<td>Inside Visual C++</td>
<td>06/19/1998</td>
<td>Reference</td>
<td>N</td>
</tr>
<tr>
<td>Core Servlets</td>
<td>05/30/2001</td>
<td>JAVA</td>
<td>Y</td>
</tr>
<tr>
<td>Core Servlets</td>
<td>05/30/2001</td>
<td>Servlets</td>
<td>N</td>
</tr>
<tr>
<td>Core Servlets</td>
<td>05/30/2001</td>
<td>Reference</td>
<td>N</td>
</tr>
</tbody>
</table>
```
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.

Here is the XML document:

```xml
<?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
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- 2011-01-11T13:20:17 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 903000.1.0.20101208190000_v930 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
< SXLEMAP description="Reading a Basic and Extended format ISO date field" name="ISOdate" version="2.1" >
  <NAMESPACES count="0"/>
  <!-- ############################################################ -->
  <!-- SAS XML Libname Engine Map -->
  <!-- Generated by XML Mapper, 903000.1.0.20101208190000_v930 -->
  <!-- Validation report -->
  <!-- XMLMap validation completed successfully. -->
  <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>
      <INFORMAT width="8">B8601DA</INFORMAT>
    </COLUMN>
    <COLUMN name="EXTENDED">
      <PATH syntax="XPath">/Root/ISODATE/EXTENDED</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>date</DATATYPE>
    </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 \texttt{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 \texttt{yyyymmdd}.

   \textbf{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 \texttt{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 \texttt{yyyy-mm-dd}.

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

```sas
filename dates 'C:\My Documents/XML\isodate.xml';
filename map 'C:\My Documents/XML\isodate.map';
libname dates xmlv2 xmlmap=map;
proc print data=dates.isodate;
run;
```

\textbf{Output 5.12} \textit{PRINT Procedure Output for Imported Data Set Dates.Isodate}

![The SAS System Table]

<table>
<thead>
<tr>
<th>Obs</th>
<th>BASIC</th>
<th>EXTENDED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2001-09-11</td>
<td>2001-09-11</td>
</tr>
</tbody>
</table>

\textbf{Using ISO 8601 SAS Informs 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.

Here is an XML document that contains a variety of time values:

```xml
<?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
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 903000.1.0.20101208190000_y930 -->
<!-- ############################################################ -->
<!-- ### Validation report                                   ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- XMLMap name="ISOtime" version="2.1" -->

<NAMESPACE count="0"/>

<!-- ############################################################ -->
<!-- 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>
  <FORMAT width="8">E8601TM</FORMAT>
  <INFORMAT width="8">E8601TM</INFORMAT>
</COLUMN>

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

<COLUMN name="UTC">
  <PATH syntax="XPath">/Root/TIME/UTC</PATH>
  <TYPE>numeric</TYPE>
  <DATATYPE>time</DATATYPE>
  <FORMAT width="9">E8601TZ</FORMAT>
</COLUMN>
```
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 \textit{hh:mm:ss.ffffff}. 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 \textit{hh:mm:ss.ffffff}. 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 \textit{hh:mm:ss+|-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.

\textit{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 \textit{hh:mm:ss+|-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 \textit{hh:mm:ss+|-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:

```sas
filename timzn 'C:\My Documents\XML\Time.xml';
filename map 'C:\My Documents\XML\Time.map';
libname timzn xmlv2 xmlmap=map;
proc print data=timzn.time;
```
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';
filename map 'C:\My Documents\XML\Nhl.map';
libname nhl xml xmlmap=map;
proc copy indd=nhl outdd=work;
select nhl;
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 PROC COPY 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 COPY procedure or a DATA step.

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.

Here is the XML document Nhl.xml to be imported:

```xml
<?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
<?xml version="1.0" ?>
<SXLEMAP version="2.1">
  <TABLE name="TEAMS">
    <TABLE-PATH syntax="XPath">
      /NHL/CONFERENCE/DIVISION/TEAM
    </TABLE-PATH>
    <COLUMN name="ABBREV">
      <PATH syntax="XPath">
        /NHL/CONFERENCE/DIVISION/TEAM/@abbrev
      </PATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>3</LENGTH>
    </COLUMN>
    <COLUMN name="FOUNDED">
      <PATH syntax="XPath">
        /NHL/CONFERENCE/DIVISION/TEAM/@founded[@abbrev="ATL"]
      </PATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>10</LENGTH>
    </COLUMN>
    <COLUMN name="CONFERENCE" retain="YES">
      <PATH syntax="XPath">
        /NHL/CONFERENCE
      </PATH>
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>10</LENGTH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```
The Abbrev variable uses the attribute form that selects values from a specific attribute. The engine scans the XML markup until it finds the TEAM element. The engine retrieves the value from the abbrev= attribute, which results in each team abbreviation.

The Founded variable uses the attribute form that conditionally selects from a specific attribute based on the value of another attribute. The engine scans the XML markup until it finds the TEAM element. The 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.

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

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

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 engine to scan the XML markup until it finds the fifth occurrence of the TEAM element. The 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 NhlShort.xml and specify the XMLMap named Nhl1.map. The PRINT procedure shows the resulting variables with selected values:

```sql
filename nhl 'C:\My Documents\XML\NhlShort.xml';
filename map 'C:\My Documents\XML\Nhl1.map';
libname nhl xmlv2 xmlmap=map;
```
proc print data=nhl.teams noobs;
run;

Output 5.14  PRINT Procedure Output Showing Resulting Variables with Selected Values

The SAS System

<table>
<thead>
<tr>
<th>ABBREV</th>
<th>FOUNDED</th>
<th>CONFERENCE</th>
<th>TEAM</th>
<th>TEAM5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL</td>
<td>1999</td>
<td>Eastern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAR</td>
<td></td>
<td>Eastern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLA</td>
<td></td>
<td>Eastern</td>
<td>Panthers</td>
<td></td>
</tr>
<tr>
<td>TB</td>
<td></td>
<td>Eastern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WSH</td>
<td></td>
<td>Eastern</td>
<td>Capitals</td>
<td></td>
</tr>
</tbody>
</table>

Including Namespace Elements in an XMLMap

This example illustrates the XMLMap namespace elements. The XMLMap namespace elements enable you to import an XML document with like-named elements that are qualified with XML namespaces. The XMLMap namespace elements maintain XML namespaces from the imported XML document to export an XML document with namespaces from the SAS data set.

Here is an XML document named NSSample.xml to be imported. The XML document contains three XML namespaces. The namespaces distinguish ADDRESS elements by qualifying them with references to unique URIs. The ADDRESS elements are highlighted below in the first PERSON repeating element:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PEOPLE xmlns:HOME="http://sample.url.org/home"
         xmlns:IP="http://sample.url.org/ip"
         xmlns:WORK="http://sample.url.org/work">
  <PERSON>
    <NAME>Joe Smith</NAME>
    <HOME:ADDRESS>1234 Elm Street</HOME:ADDRESS>
    <HOME:PHONE>999-555-0011</HOME:PHONE>
    <WORK:ADDRESS>2001 Office Drive, Box 101</WORK:ADDRESS>
    <WORK:PHONE>999-555-0101</WORK:PHONE>
    <IP:ADDRESS>192.168.1.1</IP:ADDRESS>
  </PERSON>
  <PERSON>
    <NAME>Jane Jones</NAME>
    <HOME:ADDRESS>9876 Main Street</HOME:ADDRESS>
    <HOME:PHONE>999-555-0022</HOME:PHONE>
    <WORK:ADDRESS>2001 Office Drive, Box 102</WORK:ADDRESS>
    <WORK:PHONE>999-555-0102</WORK:PHONE>
  </PERSON>
</PEOPLE>
```
Here is the XMLMap that was used to import the XML document. Notations describe the namespace elements.

```
<IP:ADDRESS>172.16.1.2</IP:ADDRESS>
</PERSON>

<PERSON>
  <NAME>Pat Perkinson</NAME>
  <HOME:ADDRESS>1395 Half Way</HOME:ADDRESS>
  <HOME:PHONE>999-555-0033</HOME:PHONE>
  <WORK:ADDRESS>2001 Office Drive, Box 103</WORK:ADDRESS>
  <WORK:PHONE>999-555-0103</WORK:PHONE>
</PERSON>

Here is the XMLMap that was used to import the XML document. Notations describe the namespace elements.

```
<IP:ADDRESS>172.16.1.2</IP:ADDRESS>
</PERSON>

<PERSON>
  <NAME>Pat Perkinson</NAME>
  <HOME:ADDRESS>1395 Half Way</HOME:ADDRESS>
  <HOME:PHONE>999-555-0033</HOME:PHONE>
  <WORK:ADDRESS>2001 Office Drive, Box 103</WORK:ADDRESS>
  <WORK:PHONE>999-555-0103</WORK:PHONE>
</PERSON>

<IP:ADDRESS>10.0.1.3</IP:ADDRESS>
</PERSON>

</PEOPLE>

Here is the XMLMap that was used to import the XML document. Notations describe the namespace elements.

```
<NS id="1" prefix="HOME">http://sample.url.org/home</NS>
<NS id="2" prefix="IP">http://sample.url.org/ip</NS>
<NS id="3" prefix="WORK">http://sample.url.org/work</NS>

<TABLE description="PERSON" name="PERSON">
  <TABLE-PATH syntax="XPath">/PEOPLE/PERSON</TABLE-PATH>

  <COLUMN name="NAME">
    <PATH syntax="XPath">/PEOPLE/PERSON/NAME</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>13</LENGTH>
  </COLUMN>

  <COLUMN name="ADDRESS">
    <PATH syntax="XPath"/PEOPLE/PERSON/{1}ADDRESS</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>16</LENGTH>
  </COLUMN>

  <COLUMN name="PHONE">
    <PATH syntax="XPath"/PEOPLE/PERSON/{1}PHONE</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>12</LENGTH>
  </COLUMN>

  <COLUMN name="ADDRESS1">
    <PATH syntax="XPath"/PEOPLE/PERSON/{3}ADDRESS</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>26</LENGTH>
  </COLUMN>

  <COLUMN name="PHONE1">
    <PATH syntax="XPath"/PEOPLE/PERSON/{3}PHONE</PATH>
    <TYPE>character</TYPE>
```

Including Namespace Elements in an XMLMap

75
A NAMESPACES element contains NS elements for defining XML namespaces. The count= attribute specifies that there are three defined XML namespaces.

Three NS elements define the XML namespaces by referencing unique URIs. The id= attribute specifies the identification numbers 1, 2, and 3 for the three XML namespaces. The prefix= attribute assigns the names HOME, WORK, and IP to the referenced URIs.

The XMLMap TABLE element contains the data set definition for the PERSON repeating element.

XMLMap COLUMN elements contain variable definitions for each nested element within PERSON, which includes NAME, ADDRESS, PHONE, ADDRESS1, PHONE1, and ADDRESS2.

In the PATH element for each COLUMN element, the type of syntax is specified as XPathENR (XPath with Embedded Namespace Reference). This type indicates that the syntax is not compliant with the XPath specification. In addition, the identification number is included in the location path preceding the element that is being defined. The identification number is enclosed in braces. For example, this is the PATH element for the ADDRESS element: `<PATH syntax="XPathENR">/PEOPLE/PERSON/{1}ADDRESS</PATH>`.

The following SAS statements import the XML document and specify an XMLMap named NSSample.map. The PRINT procedure shows the resulting SAS data set:

```sas
filename ns 'C:\My Documents\XML\NSSample.xml';
filename nsmap 'C:\My Documents\XML\NSSample.map';
libname ns xmlv2 xmlmap=nsmap;
proc print data=ns.person noobs;
run;
```
This example illustrates how to import an XML document using the AUTOMAP= option on page 100 to automatically generate an XMLMap file. By specifying the AUTOMAP= option in the LIBNAME statement, SAS analyzes the structure of the specified XML document and generates XMLMap syntax that describes how to interpret the XML markup into a SAS data set or data sets, variables (columns), and observations (rows).

Here is the XML document Nhl.xml to be imported. If you try to import the document without an XMLMap, an error indicates that the data is not in a supported format.

```xml
<?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>
```
The following SAS statements import the XML document Nhl.xml:

```sas
filename nhl 'C:\My Documents\XML\Nhl.xml';
filename map 'C:\My Documents\XML\NhlGenerate.map';
libname nhl xmlv2 automap=replace xmlmap=map;
proc print data=nhl.team;
run;
```

1. The first FILENAME statement assigns the file reference Nhl to the physical location (complete pathname, filename, and file extension) of the XML document named Nhl.xml to be imported.

2. The second FILENAME statement assigns the file reference Map to the physical location of the XMLMap named NhlGenerate.map to be generated.

3. The LIBNAME statement includes the following arguments:
   - The LIBNAME statement assigns the library reference Nhl, which matches the file reference that is assigned in the first FILENAME statement. Because the library reference and file reference match, the physical location of the XML document to be imported does not have to be specified in the LIBNAME statement.
   - The XMLV2 engine is specified.
   - The AUTOMAP=REPLACE option requests an XMLMap file to be generated and to overwrite the filename, if it exists. The default setting PREFIXATTRIBUTES=YES specifies that the element name is concatenated to the attribute name in each generated XMLMap COLUMN element, which defines the SAS variable name.
   - The XMLMAP= option specifies the file reference map, which matches the file reference that is assigned in the second FILENAME statement. The file reference is associated with the physical location of the XMLMap to be generated.

4. PROC PRINT produces output, verifying that the import was successful.

Here is the generated XMLMap NhlGenerate.map:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- 2012-09-21T10:30:48 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904000.4.0.20120905190000_v940 -->
<!-- Validation report -->
<!-- XMLMap validation completed successfully. -->
<SESSION MAP="$AUTO_GEN" version="2.1">
  <NAMESPACES count="0"/>
  <TABLE description="NHL" name="NHL">
    <TABLE-PATH syntax="XPath">/NHL</TABLE-PATH>
  </TABLE>
</SESSION>
```
Importing an XML Document Using the AUTOMAP= Option to Generate an XMLMap

```xml
<TABLE description="CONFERENCE" name="CONFERENCE">
  <TABLE-PATH syntax="XPath">/NHL/CONFERENCE</TABLE-PATH>

  <COLUMN class="ORDINAL" name="NHL_ORDINAL">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/NHL</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN class="ORDINAL" name="CONFERENCE_ORDINAL">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/NHL/CONFERENCE</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN name="CONFERENCE">
    <PATH syntax="XPath">/NHL/CONFERENCE</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>12</LENGTH>
  </COLUMN>

</TABLE>

<!-- ############################################################ -->

<!-- ############################################################ -->

<TABLE description="DIVISION" name="DIVISION">
  <TABLE-PATH syntax="XPath">/NHL/CONFERENCE/DIVISION</TABLE-PATH>

  <COLUMN class="ORDINAL" name="CONFERENCE_ORDINAL">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/NHL/CONFERENCE</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN class="ORDINAL" name="DIVISION_ORDINAL">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/NHL/CONFERENCE/DIVISION</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN name="DIVISION">
    <PATH syntax="XPath">/NHL/CONFERENCE/DIVISION</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>28</LENGTH>
  </COLUMN>

</TABLE>
```

Here is the PRINT procedure output for Nhl.Team
### The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>DIVISION_ORDINAL</th>
<th>TEAM_ORDINAL</th>
<th>TEAM_name</th>
<th>TEAM_abbrev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Thrashers</td>
<td>ATL</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Hurricanes</td>
<td>CAR</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>Panthers</td>
<td>FLA</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>Lightning</td>
<td>TB</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Capitals</td>
<td>WSH</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td>Stars</td>
<td>DAL</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>Kings</td>
<td>LA</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td>Ducks</td>
<td>ANA</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>9</td>
<td>Coyotes</td>
<td>PHX</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10</td>
<td>Sharks</td>
<td>SJ</td>
</tr>
</tbody>
</table>
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.

```sas
proc template;
/* +------------------------------------------------+ 
|                                                | 
+------------------------------------------------+ */
define tagset tagsets.custom ;
notes "SAS XML Engine output event model(interface)";
indent = 3;
map = '<>&"''';
mapsub = '/</>/&/"''/';
/* +------------------------------------------------+ 
|                                                | 
+------------------------------------------------+ */
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;
```

Exporting an XML Document Using a Customized Tagset

```plaintext
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 */
brbreak / if ^cmp("Print", name);
eval $is_procprint 1; /* PROC PRINT */
eval $regex prxparse("\/(.+)/");
eval $match prxmatch($regex, value);
set $TABLENAME prxposn($regex, 1, value);
brbreak;
finish:
  break;
end;

define event output;
start:
brbreak / if $is_procprint ;
eval $is_engine 0; /* NOT ENGINE */
set $TABLENAME name / if name; /* TABLE VIEWER */
brbreak;
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 */
brbreak;
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;
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; 1
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 ;
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.

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.

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

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;
set sashelp.class (obs=3); run;

filename xmlout "C:\My Documents\XML\engine92.xml";
libname xmlout xml xmltype=GENERIC tagset=tagsets.custom;
data xmlout.class;
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
<?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

LIBNAME Statement Reference

Chapter 7
LIBNAME Statement: Overview ........................................ 93

Chapter 8
LIBNAME Statement Syntax ............................................ 97
Chapter 7
LIBNAME Statement: Overview

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 to import or export an XML document.


Understanding the XML LIBNAME Engine Versions: XML and XMLV2

About the XML Engine Versions
SAS provides two versions of 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 XMLV2, you access XML engine functionality with enhancements and changes after SAS 9.1.3. For example, the XMLV2 version provides enhanced LIBNAME statement functionality, new XMLMap functionality, and diagnostics of obsolete syntax.

The major differences between the versions include the following:

• The XMLV2 version is XML compliant.
• LIBNAME statement functionality for XMLV2 includes the XMLMAP markup type, additional options, and the ability to assign a libref to a SAS library.

• XMLMap functionality for XMLV2 includes the ability to use an XMLMap for exporting and support for XML namespaces.

Comparing the XML Engine Versions: XMLV2 and XML

XML Compliance
The XMLV2 version is XML compliant, which means that XMLV2 requires XML markup to be well-formed and in valid construction that is in compliance with the W3C specifications. Because the XMLV2 version is XML compliant, using XMLV2 could affect the following situations:

• XML documents that are imported with the XML version might not pass the more strict parsing rules in the XMLV2 version. For example, like XML markup, the XMLV2 version is case sensitive. Opening and closing tags must be written in the same case, such as `<BODY> ... </BODY>` and `<Message> ... </Message>`. For the XMLV2 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 XMLV2 version. The XMLV2 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 `<SXMLMAP version="2.1">`. 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 XMLV2 version supports XMLMap files starting with XMLMap version 1.2. The documented XMLMap syntax version is 2.1. See Chapter 9, “XMLMap Syntax: Overview,” on page 113.

LIBNAME Statement Functionality Enhancements for XMLV2
The XMLV2 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.

• The XMLMAP markup type.

• Additional options. For a list of the LIBNAME statement options that are available for the XML and XMLV2 nicknames, see “LIBNAME Statement Options” on page 95.

• Using the XMLV2 nickname 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:

```sas
libname stones xmlv2 'c:\Grades.xml';

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

LIBNAME Statement 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**

<table>
<thead>
<tr>
<th>Task</th>
<th>Option</th>
<th>XML</th>
<th>XMLV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically generate an XMLMap file to import an XML document</td>
<td>AUTOMAP= on page 100</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>Determine whether SAS formats are used with the GENERIC markup type</td>
<td>FORMATACTIVE= on page 101</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>When importing or exporting a CDISC ODM XML document with the CDISCODM markup type</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Determine whether SAS formats are used</td>
<td>FORMATACTIVE= on page 101</td>
<td>■</td>
<td></td>
</tr>
<tr>
<td>Specify the libref to create a format catalog</td>
<td>FORMATLIBRARY= on page 102</td>
<td>■</td>
<td></td>
</tr>
<tr>
<td>Replace existing format entries in the format catalog</td>
<td>FORMATNOREPLACE = on page 102</td>
<td>■</td>
<td></td>
</tr>
<tr>
<td>Indent nested elements in exported XML document</td>
<td>INDENT= on page 102</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Specify the character set to use for the output file</td>
<td>ODSCHARSET= on page 102</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Control the generation of a record separator</td>
<td>ODSRECSEP= on page 103</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Task</td>
<td>Option</td>
<td>XML</td>
<td>XMLV2</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------------------------------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>Specify the translation table to use for the output file</td>
<td>ODSTRANTAB= on page 103</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element</td>
<td>PREFIXATTRIBUTES= on page 104</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Override the default tagset</td>
<td>TAGSET= on page 104</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Import concatenated XML documents</td>
<td>XMLCONCATENATE= on page 104</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Specify the tag format to contain SAS variable information</td>
<td>XMLDATAFORM= on page 105</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Control the results of numeric values</td>
<td>XMLDOUBLE= on page 105</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Override the SAS data set's encoding for the output file</td>
<td>XMLENCODING= on page 106</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Specify a fileref for the XML document</td>
<td>XMLFILEREF= on page 106</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Specify an XMLMap</td>
<td>XMLMAP= on page 107</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Determine whether metadata-related information is included</td>
<td>XMLMETA= on page 107</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Determine whether to process nonconforming character data</td>
<td>XMLPROCESS= on page 108</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Specify an external file to contain exported metadata-related information</td>
<td>XMLSCHEMA= on page 109</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Specify the XML markup type</td>
<td>XMLTYPE= on page 109</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>
Chapter 8
LIBNAME Statement Syntax

Processes an XML document.

Valid in: Anywhere
Category: Data Access

Syntax

LIBNAME libref engine 'SAS-library | XML-document-path' <option(s)>;

Summary of Optional Arguments

AUTOMAP=REPLACE | REUSE XMLV2 Only
specifies to automatically generate an XMLMap file to import an XML document.

FORMATACTIVE=NO | YES
determines whether SAS formats are used.

FORMATLIBRARY=libref XML Only
specifies the libref of an existing SAS library in which to create the format catalog.

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.

INDENT=integer
specifies the number of columns to indent each nested element in the
exported XML document.

**ODSCHARSET=character-set**
specifies the character set to use for the output file.

**ODSRECSEP=DEFAULT | NONE | YES**  
XML Only  
controls the generation of a record separator that marks the end of a line in
the output XML document.

**ODSTRACTAB=table-name**
specifies the translation table to use for the output file.

**PREFIXATTRIBUTES=YES | NO**  
XMLV2 Only  
specifies whether the element name is concatenated to the attribute name
when generating each XMLMap COLUMN element, which defines the SAS
variable name.

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

**XMLCONCATENATE=NO | YES**
specifies whether the file to be imported contains multiple, concatenated
XML documents

**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.

**XMLDOUBLE=DISPLAY | INTERNAL**
controls the results of importing or exporting numeric values.

**XMLENCODING='encoding-value'**
overrides the SAS data set's encoding for the output file.

**XMLFILEREF=fileref**
is the SAS name that is associated with the physical location of the XML
document to be exported or imported.

**XMLMAP=fileref | 'XMLMap'**
specifies an XML document that you create that contains specific XMLMap
syntax.

**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.

**XMLPROCESS=CONFORM | PERMIT**
determines how the XML engine processes character data that does not
conform to W3C specifications.

**XMLSCHEMA=fileref | 'external-file'**
specifies an external file to contain metadata-related information.

**XMLTYPE=GENERIC | CDISCODM | MSACCESS | ORACLE | XMLMAP**
specifies the XML markup type.

---

**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 *XML Only*.

XMLV2

specifies the XML engine nickname that accesses the engine functionality since SAS 9.2. The syntax for functionality that is available only for the XMLV2 engine nickname is labeled with *XMLV2 Only*.

Alias

| XML92 |

Restriction

When exporting an XML document using the XMLV2 engine, you can specify up to 19 SAS data sets to export.

Note

If syntax is not labeled with either *XML Only* or *XMLV2 Only*, 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

*XMLV2 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.

*Interactions*

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:

```sas
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 106. For example, the following code writes to the XML document Wilma.XML:

```sas
filename cartoon 'C:\XMLdata\wilma.xml';
libname bedrock xml xmlfileref=cartoon;
proc print data=bedrock.wilma;
run;
```

### Optional Arguments

**AUTOMAP=** `REPLACE | REUSE`  
XMLV2 Only  
specifies to automatically generate an XMLMap file to import an XML document. The XMLMap file contains specific syntax that describes how to interpret the XML markup into a SAS data set or data sets, variables (columns), and observations (rows). XMLMap syntax is generated by analyzing the structure of the specified XML document. To automatically generate the XMLMap file, you must specify an existing XML document and the physical location for the output XMLMap file.

- **REPLACE**  
  overwrites an existing XMLMap file.
  
  If an XMLMap file exists at the specified physical location, the generated XMLMap file overwrites the existing one. If an XMLMap file does not exist at the specified physical location, the generated XMLMap file is written to the specified pathname and filename.

- **REUSE**  
  does not overwrite an existing XMLMap file.
  
  If an XMLMap file exists at the specified physical location, the existing XMLMap file is used. If an XMLMap file does not exist at the specified physical location, the generated XMLMap file is written to the specified pathname and filename.

### Restriction

Use this option when importing only.

### Requirements

You must specify the physical location of an existing XML document with either the complete pathname, filename, and file extension, or with a file reference that is associated with the physical location for the DISK or TEMP device type only. The XML document must exist on disk. The AUTOMAP= option does not support accessing an XML document by using access methods such as FTP, SFTP, URL, or WebDAV.

You must include the `XMLMAP=` option on page 107 to specify the physical location of the generated XMLMap file with either the complete pathname, filename, and file extension, or with a file reference that is associated with the physical location for the DISK or TEMP device type only. The AUTOMAP= option does not support accessing an XMLMap file by using access methods such as FTP, SFTP, URL, or WebDAV.
Interaction

To specify whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element, which defines the SAS variable name, use the **PREFIXATTRIBUTES=** option on page 104. By default, element names are attached to the resulting SAS variable names.

Tips

The functionality to automatically generate an XMLMap file is also available with SAS XML Mapper. The **AUTOMAP=** option provides the functionality for the XMLV2 LIBNAME engine so that the XMLMap is created and used with a single LIBNAME statement.

You can assign the generated XMLMap file to a temporary location, which is deleted at the end of the SAS session. To do this, associate a file reference to a physical location and specify the **TEMP** device type.

Example

“Importing an XML Document Using the **AUTOMAP=** Option to Generate an XMLMap” on page 77

---

**FORMATACTIVE=** 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.

<table>
<thead>
<tr>
<th>NO</th>
<th>XML Only</th>
<th>causes formatting controls to be suppressed for both importing and exporting.</th>
</tr>
</thead>
</table>

| 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. |

Tips

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 on page 102.

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**.

Example

“Exporting an XML Document in CDISC ODM Markup” on page 22

---

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

<table>
<thead>
<tr>
<th>NO</th>
<th>writes the actual data value to the XML markup.</th>
</tr>
</thead>
</table>

| 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.

Restrictions Use this option when importing an XML document only.

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

Restrictions Use this option when importing an XML document only.

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 indentation) 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.

**ODSRECSEP=DEFAULT | NONE | YES** XML Only

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 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.


**PREFIXATTRIBUTES=**YES | NO  XMLV2 Only

specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element, which defines the SAS variable name. If you specify PREFIXATTRIBUTES=NO, the element name is not attached to the resulting SAS variable name.

Default YES

Restrictions Use this option when importing an XML document only.

Example “Importing an XML Document Using the AUTOMAP= Option to Generate an XMLMap” on page 77

**TAGSET=tagset-name**

specifies the name of a tagset 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 the TEMPLATE procedure 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 Chapter 6, “Understanding and Using Tagsets for the XML Engine,” on page 83

Example “Exporting an XML Document Using a Customized Tagset” on page 84

**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.

**XMLCONCATENATE=**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 or response as in a web form).

Alias XMLCONCAT=

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.

Example

“Importing Concatenated XML Documents” on page 33

**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 is `<COLUMN name="PRICE" value="1.98" />`.

<table>
<thead>
<tr>
<th>Default</th>
<th>ELEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrictions</td>
<td>Use this option when exporting an XML document only.</td>
</tr>
<tr>
<td></td>
<td>Use this option for the GENERIC markup type only.</td>
</tr>
</tbody>
</table>

**XMLDOUBLE=DISPLAY | INTERNAL**

controls the results of importing or exporting numeric values.

**DISPLAY**

when exporting, the SAS XML LIBNAME 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. The display value is affected by the specified engine nickname and whether a format is assigned.

- The XML engine nickname uses an assigned format. The maximum value is 16 digits. For example, if a numeric variable has an assigned format width that is 20 digits, such as BEST20., the engine truncates the exported value. If there is not an assigned format, the engine displays the value using BEST10.
- The XMLV2 engine nickname ignores any assigned format and displays the value using BEST16.

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

**Alias** FORMAT

**INTERNAL**

when exporting, the SAS XML LIBNAME 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 the 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 SAS XML LIBNAME engine retrieves the stored value from the `rawvalue=` attribute from the named element in the XML document. It converts that value into numeric variable content. The PCDATA content of the element is ignored. When importing, XMLDOUBLE=INTERNAL is not supported for the XMLV2 engine nickname.

**Alias** PRECISION

**Tip**

Typically, you use XMLDOUBLE=INTERNAL to import or export an XML document when content is more important than readability.
Default | DISPLAY
---|---
**Restriction** | You can specify the XMLDOUBLE= option for the GENERIC markup type only.

**Examples** | “Exporting Numeric Values” on page 12

| “Importing an XML Document with Numeric Values” on page 25

**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.

**Tips** | 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.

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


**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:

```sas
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 71.
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;

Restrictions The XMLV2 engine nickname supports XMLMap syntax versions 1.2, 1.9, and 2.1. The XMLV2 engine nickname does not support XMLMap versions 1.0 or 1.1.

The XML engine nickname supports XMLMap syntax versions 1.0, 1.1, and 1.2. The XML engine nickname does not support XMLMap syntax versions 1.9 or 2.1.

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 9, “XMLMap Syntax: Overview,” on page 113

Example Chapter 5, “Importing XML Documents Using an XMLMap,” on page 43

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.

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
Prior to 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.

Examples
“Exporting an XML Document with Separate Metadata” on page 17
“Importing an XML Document Created by Microsoft Access” on page 29

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.

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

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

Example  “Importing an XML Document with Non-Escaped Character Data” on page 27

<table>
<thead>
<tr>
<th>XMLSCHEMA=fileref</th>
<th>'external-file'</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies an external file to contain metadata-related information.</td>
<td></td>
</tr>
</tbody>
</table>

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.

Restrictions  Use this option when exporting an XML document only.

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.

Example  “Exporting an XML Document with Separate Metadata” on page 17

XMLTYPE=GENERIC | CDISCODM | MSACCESS | ORACLE | XMLMAP
specifies the XML markup type.

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 44

Examples  “Exporting an XML Document Containing SAS Dates, Times, and Datetimes” on page 11

“Exporting Numeric Values” on page 12

“Importing an XML Document Using the GENERIC Markup Type” on page 23

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.

Examples

“Importing a CDISC ODM Document” on page 36

“Exporting an XML Document in CDISC ODM Markup” on page 22

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.

Example

“Importing an XML Document Created by Microsoft Access” on page 29

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.

Example

“Exporting an XML Document for Use by Oracle” on page 9

XMLMAP XMLV2 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, use the XMLMAP= option on page 107.

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

Example

“Using an XMLMap to Export an XML Document with a Hierarchical Structure” on page 39

Default GENERIC

Tip You can control the markup by specifying options such as INDENT=, XMLDATAFORM=, XMLMETA= (when applicable), and TAGSET=.
Part 3

XMLMap File Reference

Chapter 9
XMLMap Syntax: Overview ............................................. 113

Chapter 10
XMLMap Syntax Version 2.1 ............................................. 117

Chapter 11
Using SAS XML Mapper to Generate and Update an XMLMap . . . 135
Chapter 9
XMLMap Syntax: Overview

Using XMLMap Syntax
The XML elements for the XMLMap syntax for version 2.1 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 for the generated output file. See “SXLEMAP Element” on page 117.
- The namespace elements define XML namespaces, which distinguish element and attribute names by qualifying them with Uniform Resource Identifier (URIs). See “Elements for XML Namespaces” on page 118.
- If you use an XMLMap for exporting, you must include the exporting elements. See “Elements for Exporting” on page 120.
- The table elements define the SAS data set. See “Elements for Tables” on page 121.
- The column elements define the variables for the SAS data set. See “Elements for Columns” on page 125.

CAUTION:
The XMLMap markup, as XML itself, is case sensitive. The tag names must be uppercase, and the element attributes must be lowercase. For example, `<SXLEMAP version="2.1">`. In addition, the supported XPath syntax is case sensitive as well.

Comparing the XMLMap Syntax
The following table lists the available XMLMap syntax. The ■ symbol indicates whether the syntax is available for importing or exporting, and whether the syntax is available for an engine nickname.
### Table 9.1 XMLMap Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Import</th>
<th>Export</th>
<th>XML</th>
<th>XMLV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXLEMAP on page 117</td>
<td>Primary (root) enclosing element</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>NAMESPACES on page 118</td>
<td>Contains one or more NS elements for defining XML namespaces</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>NS on page 119</td>
<td>Defines an XML namespace by referencing a unique URI</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>OUTPUT on page 120</td>
<td>Contains one or more HEADING elements and one TABLEREF element for exporting a SAS data set</td>
<td>■</td>
<td>■</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HEADING on page 120</td>
<td>Contains one or more ATTRIBUTE elements</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>ATTRIBUTE on page 120</td>
<td>Contains file attribute information</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLEREF on page 120</td>
<td>Specifies the name of the table</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE on page 121</td>
<td>Contains a data set definition</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE-PATH on page 121</td>
<td>Specifies a location path for variables</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE-END-PATH on page 122</td>
<td>Specifies a location path to stop processing</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE-DESCRIPTION on page 124</td>
<td>Specifies a SAS data set description</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>COLUMN name= on page 125</td>
<td>Specifies the variable name</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>COLUMN retain= on page 125</td>
<td>Determines the contents of the input buffer</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>COLUMN class= on page 125</td>
<td>Determines the type of variable</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TYPE on page 126</td>
<td>Specifies the SAS data type for the variable</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>DATATYPE on page 126</td>
<td>Specifies the type of data being read</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>DEFAULT on page 127</td>
<td>Specifies a default value for a missing value</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
</tbody>
</table>

114 Chapter 9 • XMLMap Syntax: Overview
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Import</th>
<th>Export</th>
<th>XML</th>
<th>XMLV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENUM on page 127</td>
<td>Contains a list of valid values for the variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>FORMAT on page 128</td>
<td>Specifies a SAS format for the variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>INFORMTAT on page 128</td>
<td>Specifies a SAS informat for the variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>DESCRIPTION on page 129</td>
<td>Specifies a description for the variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>LENGTH on page 129</td>
<td>Determines the maximum field storage length for a character variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>PATH on page 129</td>
<td>Specifies a location path for the current variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>INCREMENT-PATH on page 131</td>
<td>Specifies a location path for incrementing the accumulated value for a counter variable</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>RESET-PATH on page 132</td>
<td>Specifies a location path for resetting the accumulated value for a counter variable to zero</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>DECREMENT-PATH on page 133</td>
<td>Specifies a location path to decrement the accumulated value for the counter variable by 1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Chapter 10
XMLMap Syntax Version 2.1

Dictionary

SXLEMAP Element

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.

Syntax

SXLEMAP version="number" name="XMLMap" description="description"

**Attributes**

version="number"

specifies the version of the XMLMap syntax. The documented XMLMap syntax version is 2.1 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 2.1.

**Restrictions**
The XMLV2 engine nickname supports XMLMap syntax versions 1.2, 1.9, and 2.1. The XMLV2 engine nickname does not support XMLMap versions 1.0 or 1.1.
The XML engine nickname supports XMLMap syntax versions 1.0, 1.1, and 1.2. The XML engine nickname does not support XMLMap syntax versions 1.9 or 2.1.

### Tip
To update an XMLMap to version 2.1, load the existing XMLMap into SAS 9.3 XML Mapper, and then save the XMLMap. For information about SAS XML Mapper, see Chapter 11, “Using SAS XML Mapper to Generate and Update an XMLMap,” on page 135.

- **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. The description can be up to 256 characters.

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

```xml
<?xml version="1.0" ?>
<SXLEMAP version="2.1" name="Myxmlmap" description="sample XMLMap">
  <TABLE name="test1">
    .
    .
    .
  </TABLE>
  <TABLE name="test2">
    .
    .
    .
  </TABLE>
</SXLEMAP>
```

---

### Elements for XML Namespaces
Define XML namespaces.

#### Syntax

- **NAMESPACE** count="number" 
- **NS** id="number" <prefix="name"/>

#### Elements

- **NAMESPACE** count="number"  XML2 Only
  - is an optional element that contains one or more NS elements for defining XML namespaces. For example, `<NAMESPACE count="2">`.

XMLMap namespace elements enable you to import an XML document with like-named elements that are qualified with XML namespaces. In addition, XMLMap
namespace elements maintain XML namespaces from the imported XML document to export an XML document from the SAS data set.

An XML namespace is a W3C specification that distinguishes element and attribute names by qualifying them with Uniform Resource Identifiers (URIs). For example, if an XML document contains a CUSTOMER element and a PRODUCT element, and both elements contain a nested ID element, XML namespaces make each nested ID element unique.

count="number"
specifies the number of defined XML namespaces.

Requirements The count= attribute is required. The specified value must match the total number of NS elements.

Example “Including Namespace Elements in an XMLMap” on page 74

NS id="number" <prefix="name"> XMLV2 Only
is an optional element that defines an XML namespace by referencing a unique URI. The URI is a string of characters that identifies a resource on the Internet. The URI is treated by an XML parser as simply a string. Specifying the URI does not require that it be used to retrieve information. The most common URI is the Uniform Resource Locator (URL), which identifies an Internet domain address. The use of URIs as namespace names must follow the same rules as the W3C specification for XML namespaces. For example, <NS id="1" prefix="freq"> http://www.hurricanefrequency.com </NS>.

Note: It is recommended that you do not use non-escaped characters in a URI.

id="number"
specifies an identification number for the XML namespace.

Requirements The id= attribute is required.

prefix="name"
specifies a qualified name that is associated with the referenced URI. The prefix is used with each element or attribute to indicate to which XML namespace it belongs. Prefix names must follow the same rules as the W3C specification for element names.

Requirements The referenced URI must be unique.

The total number of NS elements must match the specified value in the NAMESPACES count= attribute.

Tip It is recommended that you do not use non-escaped characters in a URI.

Example “Including Namespace Elements in an XMLMap” on page 74
Elements for Exporting

Export an XML document from a SAS data set by using the XMLMap that was created to import the XML document.

**Restriction:** The engine supports exporting from one SAS data set only.

**Syntax**

```plaintext
OUTPUT
HEADING
ATTRIBUTE name="name" value="value"
TABLEREF name="name"
```

**Elements**

**OUTPUT**  XMLV2 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 or 2.1 in an XMLMap to export a SAS data set as an XML document, you must include the OUTPUT element in the XMLMap.

**Example**

“Using an XMLMap to Export an XML Document with a Hierarchical Structure” on page 39

**HEADING**  XMLV2 Only

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

**ATTRIBUTE name="name" value="value"**  XMLV2 Only

is an optional element that contains additional file attribute information for the exported XML document, such as a schema reference 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">`.

```plaintext
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"**  XMLV2 Only

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

```plaintext
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.
```

**Restriction**

You can specify one TABLEREF element only.
Elements for Tables

Define the SAS data set.

Syntax

TABLE description="description" name="data-set-name"
TABLE-PATH syntax="type"
TABLE-END-PATH syntax="type" beginend="BEGIN | END"
TABLE-DESCRIPTION

Elements

TABLE description="description" name="data-set-name"
is an element that contains a data set definition. For example, <TABLE name="channel">.

description="description"
is an optional attribute that specifies a description of the SAS data set. The description can be up to 256 characters.

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.

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.

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.

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 50, 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.

**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. 
For example, **syntax="XPath"**.

<table>
<thead>
<tr>
<th>Default</th>
<th>XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirements</strong></td>
<td>The value must be XPath or XPathENR.</td>
</tr>
</tbody>
</table>
| If an XML namespace is defined with the NAMESPACES 
element, you must specify the type of syntax as XPathENR 
(XPath with Embedded Namespace Reference). This is because 
the syntax is different from the XPath specification. For 
example, **syntax="XPathENR"**. |

**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>
<thead>
<tr>
<th>Requirements</th>
<th>The TABLE-PATH element is required.</th>
</tr>
</thead>
</table>
| If an XML namespace is defined with the NAMESPACES 
element, you must include the identification number in the location path 
preceding the element that is being defined. The identification 
number is enclosed in braces. For example, 
**<TABLE-PATH syntax="XPathENR">/Table/{1}Hurricane</TABLE-PATH>**. |

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.

**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 the 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.

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 50, 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 TABLE-END-PATH location path <TABLE-END-PATH syntax="XPath" beginend="BEGIN">/rss/channel/item </TABLE-END-PATH> tells the XML engine to stop processing when the <ITEM> start tag is encountered.

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
<?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>
    ...
    ...
  </channel>
</rss>
```
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. For example, syntax="XPath".

**Default**

XPath

**Requirements**

The value must be XPath or XPathENR.

If an XML namespace is defined with the NAMESPACES element, you must specify the type of syntax as XPathENR (XPath with Embedded Namespace Reference). This is because the syntax is different from the XPath specification. For example, syntax="XPathENR".

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

**Requirements**

If an XML namespace is defined with the NAMESPACES element, you must include the identification number in the location path preceding the element that is being defined. The identification number is enclosed in braces. For example, <TABLE-END-PATH syntax="XPathENR">/Table/{1}Hurricane</TABLE-END-PATH>.

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.

**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.

**Example**

"Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 50

**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
Define the variables for the SAS data set.

Syntax

COLUMN name="name" retain="NO | YES" class="ORDINAL"
TYPE
DATATYPE
DEFAULT
ENUM
FORMAT width="w" ndec="d"
INFORMAT width="w" ndec="d"
DESCRIPTION
LENGTH
PATH syntax="type"
INCREMENT-PATH syntax="type" beginend="BEGIN | END"
RESET-PATH syntax="type" beginend="BEGIN | END"
DECLCREMENT-PATH syntax="type" beginend="BEGIN | END"

Elements

COLUMN name="name" retain="NO | YES" class="ORDINAL"
is an element that contains a variable definition. For example, <COLUMN
name="Title">

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

Example “Importing Hierarchical Data as Related Data Sets” on page 54

class="ORDINAL" XMLV2 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.

**Requirements**

You must use the INCREMENT-PATH element or the DECREMENT-PATH element. The PATH element is not allowed.

The TYPE element must specify the SAS data type as numeric, and the DATATYPE element must specify the type of data as integer.

**Example**

“Including a Key Field with Generated Numeric Keys” on page 58

---

**TYPE**

specifies the SAS data type (character or numeric) for the variable, which is how SAS stores the data. For example, `<TYPE> numeric </TYPE>` specifies that the SAS data type for the variable is numeric.

**Requirement**

The TYPE element is required.

**Tips**

To assign a floating-point type, use

```xml
<TYPE> float </TYPE>
</DATATYPE>
```

To apply output formatting in SAS, use the FORMAT element.

To control data type conversion in input, use the INFORMAT element. For example, `<INFORMAT> datetime </INFORMAT>`.

**DATATYPE**

specifies the type of data being read from the XML document for the variable. For example, `<DATATYPE> string </DATATYPE>` specifies that the data contains alphanumeric characters.

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.ffffff.
• 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 element on page 128.

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 element on page 128.

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.ffffff.
• 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 element on page 128.

Restriction
The values for previous versions of XMLMap syntax are not
accepted by versions 1.9 and 2.1.

Requirement
The DATATYPE element is required.

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, `<DEFAULT> single </DEFAULT>` assigns the value single when
a missing value occurs.

Default
By default, the XML engine sets a missing value to MISSING.

Example
“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 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.

```xml
<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>
```

Example  “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 Formats and Informats: Reference*.

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:

```xml
<FORMAT> E8601DA </FORMAT>
<FORMAT width="8"> best </FORMAT>
<FORMAT width="8" ndec="2"> dollar </FORMAT>
```

Example  “Determining the Observation Boundary to Select the Best Columns” on page 64

**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 Formats and Informats: Reference*.

Here is an example:

```xml
<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.

```xml
<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. Multi-byte character strings that are longer than the specified length are truncated on a character boundary, not on a byte boundary.

**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.

**syntax="type"**

is an 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

Requirements The value must be XPath or XPathENR.

If an XML namespace is defined with the NAMESPACES element, you must specify the type of syntax as XPathENR (XPath with Embedded Namespace Reference). This is because the syntax is different from the XPath specification. For example, syntax="XPathENR".

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:


**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>
```
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>`

**Requirements**

Whether the PATH element is required or allowed is determined by the class="ORDINAL" attribute for the COLUMN element. 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.

If an XML namespace is defined with the NAMESPACES element, you must include the identification number in the location path preceding the element that is being defined. The identification number is enclosed in braces. For example, `<PATH syntax="XPathENR">/Table/Hurricane/{1}Month</PATH>`. See “Including Namespace Elements in an XML Map” on page 74.

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.

**Example**

“Specifying a Location Path on the PATH Element” on page 71

**INCREMENT-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 increment the accumulated value for the counter variable by 1.

**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. For example,

**syntax="XPath"**
Default | XPath
---|---
Requirements | The value must be XPath or XPathENR.

If an XML namespace is defined with the NAMESPACES element, you must specify the type of syntax as XPathENR (XPath with Embedded Namespace Reference). This is because the syntax is different from the XPath specification. For example, syntax="XPathENR".

```
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
---|---
Requirements | If an XML namespace is defined with the NAMESPACES element, you must include the identification number in the location path preceding the element that is being defined. The identification number is enclosed in braces. For example, `<INCREMENT-PATH syntax="XPathENR">/Table/Hurricane/{1}Month</INCREMENT-PATH>`.

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.

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.

Example | “Including a Key Field with Generated Numeric Keys” on page 58
---|---

**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 zero.

```
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. For example, syntax="XPath".

Default | XPath
---|---
Requirements | The value must be XPath or XPathENR.
If an XML namespace is defined with the NAMESPACES element, you must specify the type of syntax as XPathENR (XPath with Embedded Namespace Reference). This is because the syntax is different from the XPath specification. For example, \texttt{syntax="XPathENR"}.

\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

**Requirements**

If the variable is not a counter variable, RESET-PATH is not allowed. If the variable is a counter variable, RESET-PATH is optional.

If an XML namespace is defined with the NAMESPACES element, you must include the identification number in the location path preceding the element that is being defined. The identification number is enclosed in braces. For example, \texttt{<RESET-PATH syntax="XPathENR">/Table/Hurricane/{1}Month</RESET-PATH>}.

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.

**DECREMENT-PATH** \texttt{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.

\texttt{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. For example, \texttt{syntax="XPath"}.

**Default** XPath

**Requirements**

The value must be XPath or XPathENR.

If an XML namespace is defined with the NAMESPACES element, you must specify the type of syntax as XPathENR (XPath with Embedded Namespace Reference). This is because the syntax is different from the XPath specification. For example, \texttt{syntax="XPathENR"}.
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

Requirements 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.

If an XML namespace is defined with the NAMESPACES element,
you must include the identification number in the location path
preceding the element that is being defined. The identification
number is enclosed in braces. For example, `<DECREMENT-PATH
syntax="XPathENR">/Table/Hurricane/{1}Month</
DECREMENT-PATH>`.

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.
Chapter 11

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 of 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.
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 to request functionality. For example, select the File menu, and then Open XML 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 website http://support.sas.com/demosdownloads/setupcat.jsp?cat=Base+SAS+Software.

The latest version of SAS XML Mapper, which is SAS 9.4, can be downloaded and used with SAS 9.4 or with versions of SAS prior to SAS 9.4. There are some features that can be used only with SAS 9.3 and SAS 9.4 SAS XML Mapper, such as the 2.1 XMLMap version.

SAS XML Mapper has online Help attached, which includes usage examples. From the menu bar, select Help, and then Help Topics.

For a quick tutorial of SAS XML Mapper, see the video How to Automatically Generate XMLMap Files (video) on the Base SAS XML LIBNAME Engine Focus Area page at http://support.sas.com/rnd/base/xmlengine. Look for the heading XML Mapper and click on the link to the video.

To Start SAS XML Mapper

To start SAS XML Mapper:

- In a Windows environment, launch SAS XML Mapper from your desktop. Typically, you can launch SAS XML Mapper by selecting Start ⇒ All Programs ⇒ SAS ⇒ SAS XML Mapper 9.4.

- In a UNIX environment, start SAS XML Mapper from the UNIX command prompt.
Part 4

Appendixes

Appendix 1

Example CDISC ODM Document........................................ 141
Appendix 1
Example CDISC ODM Document

Here is an example of an XML document that is in the CDISC ODM format. This document is used in “Importing a CDISC ODM Document” on page 36 and in “Exporting an XML Document in CDISC ODM Markup” on page 22.
<?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 CTCChicago file"
AsOfDateTime="2009-03-31T14:01:41"
CreationDateTime="2009-03-31T14:01:41">
- <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.AEREL" OrderNumber="15" Mandatory="No" />
      <ItemRef ItemOID="ID.AEOUT" OrderNumber="16" Mandatory="No" />
      <ItemRef ItemOID="ID.AEACTTRT" OrderNumber="17" Mandatory="No" />
      <ItemRef ItemOID="ID.AECONTRT" OrderNumber="18" Mandatory="No" />
      <ItemRef ItemOID="ID.AECONTTRT" OrderNumber="19" Mandatory="No" />
    </ItemGroupDef>
  </MetaDataVersion>
</Study>
</ODM>
<table>
<thead>
<tr>
<th>ItemDef OID</th>
<th>SASFieldName</th>
<th>Name</th>
<th>DataType</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID.TAREA</td>
<td>TAREA</td>
<td>Therapeutic Area</td>
<td>text</td>
<td>4</td>
</tr>
<tr>
<td>ID.PNO</td>
<td>PNO</td>
<td>Protocol Number</td>
<td>text</td>
<td>15</td>
</tr>
<tr>
<td>ID.SCTRY</td>
<td>SCTRY</td>
<td>Country</td>
<td>text</td>
<td>4</td>
</tr>
<tr>
<td>ID.F_STATUS</td>
<td>F_STATUS</td>
<td>Record status, 5 levels, internal use</td>
<td>text</td>
<td>1</td>
</tr>
<tr>
<td>ID.LINE_NO</td>
<td>LINE_NO</td>
<td>Line Number</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>ID.AETERM</td>
<td>AETERM</td>
<td>Conmed Indication</td>
<td>text</td>
<td>100</td>
</tr>
<tr>
<td>ID.AESTMON</td>
<td>AESTMON</td>
<td>Start Month - Enter Two Digits 01-12</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>ID.AESTDAY</td>
<td>AESTDAY</td>
<td>Start Day - Enter Two Digits 01-31</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>ID.AESTYR</td>
<td>AESTYR</td>
<td>Start Year - Enter Four Digit Year</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>ID.AESTDT</td>
<td>AESTDT</td>
<td>Derived Start Date</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>ID.AEENMON</td>
<td>AEENMON</td>
<td>Stop Month - Enter Two Digits 01-12</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>ID.AEENDAY</td>
<td>AEENDAY</td>
<td>Stop Day - Enter Two Digits 01-31</td>
<td>integer</td>
<td>2</td>
</tr>
<tr>
<td>ID.AEENYR</td>
<td>AEENYR</td>
<td>Stop Year - Enter Four Digit Year</td>
<td>integer</td>
<td>4</td>
</tr>
<tr>
<td>ID.AEENDT</td>
<td>AEENDT</td>
<td>Derived Stop Date</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>ID.AESEV</td>
<td>AESEV</td>
<td>Severity</td>
<td>text</td>
<td>1</td>
</tr>
<tr>
<td>ID.AEREL</td>
<td>AEREL</td>
<td>Relationship to study drug</td>
<td>text</td>
<td>1</td>
</tr>
<tr>
<td>ID.AEOUT</td>
<td>AEOUT</td>
<td>Outcome</td>
<td>text</td>
<td>1</td>
</tr>
<tr>
<td>ID.AEACTTRT</td>
<td>AEACTTRT</td>
<td>Actions taken re study drug</td>
<td>text</td>
<td>1</td>
</tr>
<tr>
<td>ID.AECONTRT</td>
<td>AECONTRT</td>
<td>Actions taken, other</td>
<td>text</td>
<td>1</td>
</tr>
</tbody>
</table>

```
Example CDISC ODM Document
143
```
Oncology

- CodeList OID="CL.$SCTRYF" SASFormatName="$SCTRYF" Name="$SCTRYF" DataType="text"
  - CodeListItem CodedValue="USA"

- CodeList OID="CL.$F_STATU" SASFormatName="$F_STATU" Name="$F_STATU" DataType="text"
  - CodeListItem CodedValue="S"
  - CodeListItem CodedValue="V"

- CodeList OID="CL.$AESEV" SASFormatName="$AESEV" Name="$AESEV" DataType="text"
  - CodeListItem CodedValue="1"
  - CodeListItem CodedValue="2"
  - CodeListItem CodedValue="3"
  - CodeListItem CodedValue="4"

- CodeList OID="CL.$AEREL" SASFormatName="$AEREL" Name="$AEREL" DataType="text"
  - CodeListItem CodedValue="0"
  - CodeListItem CodedValue="1"
  - CodeListItem CodedValue="2"
  - CodeListItem CodedValue="3"
  - CodeListItem CodedValue="4"
- <CodeList OID="CL.$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>
Administrative metadata

Clinical Data

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="06" />
          - <ItemData ItemOID="ID.AEENDAY" Value="11" />
          - <ItemData ItemOID="ID.AEENYR" Value="1999" />
          - <ItemData ItemOID="ID.AEENDT" Value="1999-06-11" />
          - <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>
Recommended Reading

Here is the recommended reading list for this title:

- *The Little SAS Book: A Primer*
- *SAS Language Reference: Concepts*
- *SAS Statements: Reference*
- *SAS Data Set Options: Reference*
- 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 website www.w3.org/XML

For a complete list of SAS publications, go to [sas.com/store/books](http://sas.com/store/books). If you have questions about which titles you need, please contact a SAS Representative:

**SAS Books**
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: [sas.com/store/books](http://sas.com/store/books)
American Standard Code for Information Interchange (ASCII)
a 7-bit encoding standard that provides a basic set of 128 characters, supporting a
variety of computer systems. ASCII encodes the uppercase and lowercase letters of
the English alphabet, punctuation marks, the digits 0-9, and control characters. This
set of 128 characters is also included in most other encodings. See also EBCDIC.

ASCII
See American Standard Code for Information Interchange.

CDISC
See Clinical Data Interchange Standards Consortium.

CDISC ODM
See CDISC Operational Data Model.

CDISC Operational Data Model (CDISC ODM)
a vendor-neutral, platform-independent format that supports the electronic
acquisition, exchange, and archival of clinical trials data for the medical and
biopharmaceutical industries.

Clinical Data Interchange Standards Consortium (CDISC)
an organization that develops pharmaceutical industry standards. The standards
support the electronic acquisition, exchange, and archival of clinical trials data and
metadata for medical and biopharmaceutical product development. CDISC defines
several data models for the interoperability of clinical data exchange.

Coordinated Universal Time (UTC)
by international agreement, the time standard by which clocks and time are regulated
(for example, for computers and online services). Specifically, it is the local time at
the prime meridian, which passes through Greenwich, England. UTC is the term that
now supersedes Greenwich Mean Time.

data set
See SAS data set.

data view
See SAS data view.
Document Object Model (DOM)  
a cross-platform and language-independent convention for representing and 
interacting with objects in documents. Traditionally, the DOM applies to HTML, 
XHTML, and XML documents.

Document Type Definition (DTD)  
a set of markup declarations that define a document type for a markup language 
(SGML, XML, or HTML). The DTD specifies how the tags should be interpreted by 
an application that displays or prints the documents.

DOM  
See Document Object Model.

DTD  
See Document Type Definition.

EBCDIC (Extended Binary Coded Decimal Interchange Code)  
a family of single-byte and multi-byte encodings for the representation of data on 
IBM mainframe and mid-range computers.

coding  
a mapping of a coded character set to code values.

Extended Binary Coded Decimal Interchange Code  
See EBCDIC.

Extensible Markup Language (XML)  
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.

file reference  
See fileref.

File Transfer Protocol (FTP)  
a telecommunications protocol that is used for transferring files from one computer 
to another over a network.

fileref (file reference)  
a name that is temporarily assigned to an external file or to an aggregate storage 
location such as a directory or a folder. The fileref identifies the file or the storage 
location to SAS. See also libref.

format  
See SAS format.

FTP  
See File Transfer Protocol.

graphics template  
See ODS template.

hierarchical database  
a database that is organized as a tree structure of segments. A DL/I database has a 
hierarchical data structure.
informat
See SAS informat.

key field
See sequence field.

library reference
See libref.

libref (library reference)
a SAS name that is associated with the location of a SAS library. For example, in the name MYLIB.MYFILE, MYLIB is the libref, and MYFILE is a file in the SAS library. See also SAS library.

markup language
a set of codes that are embedded in text in order to define layout and certain content.

metadata
descriptive data about data that is stored and managed in a database, in order to facilitate access to captured and archived data for further use.

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.

ODS template (graphics template)
a description of how output should appear when it is formatted. ODS templates are stored as compiled entries in a template store, also known as an item store. Common template types include STATGRAPH, STYLE, CROSSTABS, TAGSET, and TABLE.

parsed character data (PCDATA)
a data definition that is used in XML to specify mixed content—that is, an element containing character data and child elements in arbitrary order and number of occurrences.

PCDATA
See parsed character data.

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, SAS data view.

SAS data set (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 view (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 format (format)
a type of SAS language element that is used to write or display data values according to the data type: numeric, character, date, time, or timestamp.

SAS informat (informat)
a type of SAS language element that is used to read data values according to the data's type: numeric, character, date, time, or timestamp.

SAS library
one or more files that are defined, recognized, and accessible by SAS, and that are referenced and stored as a unit. Each file is a member of the library.

SAS variable (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 (rows).

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.

SAX
See Simple API for XML.

sequence field (key field)
a field that identifies and provides access to segments in a database. It contains the record's key, which is located in the same position in each record of a key-sequenced data set.

Simple API for XML (SAX)
an event-based, sequential access parser API that was developed for XML documents by members of the XML-DEV group. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the Document Object Model (DOM).

tagset
a template that defines how to create a type of markup language output from a SAS format. Tagsets produce markup output such as Hypertext Markup Language (HTML), Extensible Markup Language (XML), and LaTeX. See also markup language.

Uniform Resource Identifier
See URI.

Uniform Resource Locator (URL)
a character string that is used by a web browser or other software application to access or identify a resource on the Internet or on an intranet. The resource could be a web page, an electronic image file, an audio file, a JavaServer page, or any other type of electronic object. The full form of a URL specifies which communications protocol to use for accessing the resource, as well as the directory path and filename of the resource.

URI (Uniform Resource Identifier)
a string that identifies resources such as files, images, and services on the World Wide Web. A URL is a type of URI. See also Uniform Resource Locator.
URL
   See Uniform Resource Locator.

UTC
   See Coordinated Universal Time.

variable
   See SAS variable.

W3C

World Wide Web Consortium (W3C)
an international community that develops open standards to ensure the long-term

XML
   See Extensible Markup Language.

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.

XML namespace
   a W3C specification that provides a simple method for qualifying element and
attribute names used in Extensible Markup Language (XML) documents by
associating them with namespaces identified by URI references. XML namespaces
avoid element name conflicts. For example, if an XML document contains a
CUSTOMER element and a PRODUCT element, and both elements contain a nested
ID element, XML namespaces make each nested ID element unique.

XML Path Language (XPath)
a query language that describes how to locate and process items in an XML
document. An XPath specification assigns a path description, similar to what is used
in UNIX, to each element of the XML structure.

XML schema
   defines the structure, content, and semantics of an XML document. An XML schema
is typically expressed in terms of constraints on the structure and content of the
document type.

XMLMap file
   a file that contains XML tags that tell the SAS XML LIBNAME engine how to
interpret an XML document.

XPath
   See XML Path Language.

XPath with Embedded Namespace Reference (XPathENR)
an XMLMap element that is used by the XMLV2 engine for XML namespace
support.

XPathENR
   See XPath with Embedded Namespace Reference.
Index

A
Access documents
  importing 29
  ampersand
    importing XML documents with 27, 108
apostrophe (')
  importing XML documents with 27, 108
ATTRIBUTE element 120
AUTOMAP= option
  LIBNAME statement 100

B
beginend= attribute
  DECREMENT-PATH element 134
  INCREMENT-PATH element 132
  RESET-PATH element 133
  TABLE-END-PATH element 124

count= attribute
  NAMESPACES element 119
create processing 6

D
data investigation 48
data sets, exporting XML documents from
  See exporting XML documents
  See exporting XML documents with XMLMap
data sets, importing XML documents as
  See importing XML documents
  See importing XML documents with XMLMap
DATASETS procedure 4
DATATYPE element 126
date values
  exporting XML documents containing 11
  ISO 8601 informats and formats for importing 67
datet ime values
  exporting XML documents containing 11
  DECREMENT-PATH element 133
DEFAULT element 127
DESCRIPTION element 129
description= attribute
  SXLEMAP element 118
  TABLE element 121
DOM application
  XML engine as 6
double quotation marks
    importing XML documents with 27, 108

E
elements for columns 125
elements for exporting 120
elements for tables 121
elements for XML namespaces 118
enclosed attribute format 105
encoding 6, 106
engine nicknames 7, 93, 99
ENUM element 127

when importing XML documents not
created with SAS 7
exporting elements
syntax 120
exporting XML documents 5, 9
CDISC ODM markup 22
customized tagset for 84
date, time, and datetime values 11
for Oracle 9
metadata information in separate file
17, 107
numeric values 12
exporting XML documents with
XMLMap 39
hierarchical structure 39
external files
for metadata-related information 109

F
fилerefs 106
URL access method for referencing 71
format catalog
libref for 102
replacing format entries 102
FORMAT element 128
FORMATACTIVE= option
LIBNAME statement 101
FORMATLIBRARY= option
LIBNAME statement 102
FORMATNOREPLACE= option
LIBNAME statement 102
formats
importing dates 67

G
generated numeric keys
including key field with 58
generating XMLMap 77
GENERIC markup 94, 109
exporting XML documents containing
date, time, and datetime values 11
importing XML documents 23
physical structure for importing XML
documents 44

H
HEADING element 120
hierarchical data

importing as related data sets 54
hierarchical structure
exporting XML documents with 39

I
id= attribute
NS element 119
importing XML documents 4, 23
CDISC ODM markup 36
concatenated documents 33
ersors when not created with SAS 7
GENERIC markup 23
Microsoft Access documents 29
non-escaped character data 27, 108
numeric values 25
importing XML documents with
XMLMap 43
as multiple data sets 50
as one data set 47
automatically generating XMLMap 77
avoiding concatenated data 62
columns. selecting best 64
importing hierarchical data as related
data sets 54
ISO 8601 informats and formats for
importing dates 67
ISO 8601 informats and formats for
importing time values with time
zone 68
key field with generated numeric keys
58
location path on PATH element 71
namespace elements 74
observation boundary 62, 64
physical structure for GENERIC
markup 44
URL access method for referencing
filerefs 71
INCREMENT-PATH element 131
INDENT= option
LIBNAME statement 102
INFORMAT element 128
informats
importing dates 67
input processing 6
installation
SAS XML Mapper 137
ISO 8601 informats and formats
importing dates 67
importing time values with a time zone
68

K
key field
including with generated numeric keys 58

L
LENGTH element 129
LIBNAME statement, XML 93
engine nicknames 93, 99
exporting XML documents from data sets 5
functionality enhancements for XMLV2 94
importing XML documents as data sets 4
  options 95
  required arguments 98
  syntax 97
librefs 98
  assigning 4
  for format catalog 102
location path, specifying on PATH element 71

M
map files 94
markup languages
  See tagsets
menu bar
  SAS XML Mapper 136
metadata
  exporting XML documents with separate metadata 17, 107
  external file for 109
Microsoft Access documents
  importing 29
MSACCESS markup 110
  importing XML documents 29

N
name= attribute
  ATTRIBUTE element 120
  COLUMN element 125
  SXLEMAP element 118
  TABLE element 121
  TABLEREF element 120
namespace elements
  importing XML documents with XMLMap 74
  NAMESPACES element 118
ndec= attribute
  FORMAT element 128
  INFORMAT element 129
nicknames for XML engine 7, 93
nicknames for XML LIBNAME engine 99
non-escaped character data 27, 108
NS element 119
numeric keys
  including key field with 58
  numeric values 105
  exporting 12
  importing XML documents with 25

O
observation boundary
  avoiding concatenated data 62
  selecting best columns 64
ODS MARKUP destination
  XML engine versus 7
ODSCHARSET= option
  LIBNAME statement 102
ODSRECSEP= option
  LIBNAME statement 103
ODSTRANTAB= option
  LIBNAME statement 103
one-to-many relationship 54
open element format 105
Oracle
  exporting XML documents for 9
  ORACLE markup 110
OUTPUT element 120
  output files
    translation tables for 103
  output processing 6
overriding tagsets 104

P
PATH element 129
  specifying location path on 71
physical structure
  importing XML documents with GENERIC markup 44
prefix= attribute
  NS element 119
PREFIXATTRIBUTES= option
  LIBNAME statement 104

R
read processing 6
record separators
  XML documents 103
RESET-PATH element 132
retain= attribute
  COLUMN element 125
SAS processing supported by XML engine 5
SAS XML Mapper 135
    generating and updating XMLMap 135
    installing 137
    menu bar 136
    starting 137
    toolbar 137
    windows 136
SAX application
    XML engine as 6
    sequential access engine 6
Simple API for XML (SAX) 6
    single quotation mark (') importing XML documents with 27, 108
    special characters
        importing XML documents with 27, 108
starting SAS XML Mapper 137
SXLEMAP element
    syntax 117
    syntax= attribute
        DECREMENT-PATH element 133
        INCREMENT-PATH element 131
        PATH element 129
        RESET-PATH element 132
        TABLE-END-PATH element 124
        TABLE-PATH element 122
TABLE element 121
    table elements
        syntax 121
        TABLE-DESCRIPTION element 124
        TABLE-END-PATH element 122
        TABLE-PATH element 121
        TABLEREF element 120
        TAGSET= option
            LIBNAME statement 104
tagsets 83
    customized 83
    customized, defining with TEMPLATE procedure 84
    customized, exporting XML documents 84
    overriding 104
    TEMPLATE procedure 83
        defining customized tagsets 84
    time values
        exporting XML documents containing 11
        importing with time zone 68
time zones importing time values with 68
tool bar
    SAS XML Mapper 137
    transferring XML documents across environments 6
    translation tables
        for output files 103
    TYPE element 126
update processing 6
    updating XMLMap 135
URL access method
    referencing filerefs 71
validating XML documents 6
    value= attribute
        ATTRIBUTE element 120
    version= attribute
        SXLEMAP element 117
    versions, XML engine 93
W3C specifications 27, 94
    width= attribute
        FORMAT element 128
        INFORMAT element 129
windows
    SAS XML Mapper 136
XML documents 3
    CDISC ODM format 141
    concatenated 104
    not in required physical structure 47
    record separators 103
    transferring across environments 6
    validating 6
XML documents, exporting
    See exporting XML documents
    See exporting XML documents with XMLMap
XML documents, importing
    See importing XML documents
    See importing XML documents with XMLMap
XML engine 3
    as DOM and SAX applications 6
    as sequential access engine 6
    how it works 4
    nicknames 7
ODS MARKUP destination versus 7
SAS processing supported by 5
versions 93
XML engine version 93
XML map files 94
XML Mapper
See SAS XML Mapper
XML markup 107, 109
XML namespace elements
importing XML documents with
XMLMap 74
syntax 118
XML window 136
XMLCONCATENATE= option
LIBNAME statement 33, 104
XMLDATAFORM= option
LIBNAME statement 105
XMLDOUBLE= option
LIBNAME statement 26, 105
XMLENCODING= option
LIBNAME statement 106
XMLFILEREF= option
LIBNAME statement 106
XMLMap
See also exporting XML documents with XMLMap
See also importing XML documents with XMLMap
column elements 125
comparing functionality 113
exporting elements 120
generating with AUTOMAP= option 100
generating with SAS XML Mapper 135
SXLEMAP element 117
syntax 107
table elements 121
updating with SAS XML Mapper 135
XML namespace elements 118
XMLMAP markup 94, 110
XMLMap window 136
XMLMAP= option
LIBNAME statement 107
XMLMETA= option
LIBNAME statement 17, 107
XMLPROCESS= option
LIBNAME statement 27, 108
XMLSCHEMA= option
LIBNAME statement 17, 109
XMLTYPE= option
LIBNAME statement 109
XMLV2 engine version 93
XPath attribute
DECREMENT-PATH element 133
PATH element 129
RESET-PATH element 132
TABLE-END-PATH element 124
TABLE-PATH element 122
XPathENR attribute
DECREMENT-PATH element 133
PATH element 129
RESET-PATH element 132
TABLE-END-PATH element 124
TABLE-PATH element 122
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.