SAS® 9.4 XMLV2 and XML LIBNAME Engines: User's Guide

SAS® Documentation
March 7, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2013. *SAS® 9.4 XMLV2 and XML LIBNAME Engines: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® 9.4 XMLV2 and XML LIBNAME Engines: User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

March 2024

9.4-P11:engxml

Contents

What's New in the SAS 9.4 XMLV2 Engine

Overview

The XMLV2 LIBNAME statement supports new options and provides more functionality in z/OS environments. Several documentation enhancements have been made.

Behavior Change for the AUTOMAP= LIBNAME Statement Option

Beginning with SAS 9.4M7, or if you apply a hot fix to SAS 9.4 or to SAS Viya, the behavior of the AUTOMAP= LIBNAME statement option is changed. Some XML entities are not supported. See "AUTOMAP= LIBNAME Statement Option" on page 85.

New XMLV2 LIBNAME Statement Options

In SAS 9.4M6, the following LIBNAME statement options are new:

- The "CHARMULTIPLIER= LIBNAME Statement Option" expands column (variable) lengths by a multiplier value. These column (variable) lengths are specified in an XMLMap.

- The "DERIVECHARMULTIPLIER= LIBNAME Statement Option" expands column (variable) lengths by a default multiplier value that is based on the session encoding. These column (variable) lengths are specified in an XMLMap.

In SAS 9.4M4, the new "PREFIXATTRIBUTES= LIBNAME Statement Option" specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element.

Documentation Enhancements

In SAS 9.4M6, the *XMLV2 engine* is supported throughout the documentation, except when a feature is supported by the XML engine only. In the syntax documentation, each language element is labeled as *XMLV2 only* or *XML only* or *XMLV2 and XML*. Examples that are supported for the XML engine only and not supported for the XMLV2 engine are now in an appendix: Appendix 1, "Usage Not Supported for the XMLV2 Engine," on page 135. For more information, see "Comparing the XMLV2 and XML Engines" on page 76.

In SAS 9.4M6, information about the encoding of imported XML documents is added in "Transferring an XML Document across Environments" on page 6.

In SAS 9.4M5, documentation for SAS Viya is merged with SAS 9.4M5 documentation.

Operating Environment Support

In SAS 9.4, XMLV2 engine functionality for the z/OS environment changed from preproduction to production. The engine is production in all SAS 9.4 operating environments.

PART 1

Usage

1

Getting Started with the XMLV2 Engine

What Does the XMLV2 Engine Do?

The XMLV2 LIBNAME engine processes an XML document. The engine can do the following:

- 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 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 XMLV2 Engine Works

Assigning a Libref

The XMLV2 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 XMLV2 engine can be assigned to either a specific XML document or to the physical location of a SAS library in a directory-based environment. (The older XML engine can assign a libref to a specific document, but not to a library.) 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 XMLV2 engine:

```
libname myxml xmlv2 'C:\Example\Students.xml';
```

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

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

Output 1.1 *DATASETS Procedure Output for myxml Library*

The SAS System

Directory	
Libref	MYXML
Engine	XMLV2
Physical Name	C:\Example\Students.xml
XMLType	SAS XML Generic
Version	.

#	Name	Member Type
1	STUDENTS	DATA

The PRINT procedure results in the following output:

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

Output 1.2 *PRINT Procedure Output for myxml.students*

The SAS System

Obs	ID	NAME	ADDRESS	CITY	STATE
1	755	Brad Martin	1611 Glengreen	Huntsville	Texas
2	1522	Zac Harvell	11900 Glenda	Houston	Texas

Exporting an XML Document

To export an XML document from a SAS data set, the LIBNAME statement for the XMLV2 engine assigns a libref to the XML document to be created.

In the following code, the first LIBNAME statement assigns the libref `myfiles` to the SAS library that contains the SAS data set `singers`. The second LIBNAME statement assigns the libref `myxml` to the physical location of the XML document that is to be exported from `myfiles.singers`:

```
libname myfiles 'c:\example\';

libname myxml xmlv2 'c:\output\singers.xml';
```

Executing these statements creates the XML document named `singers.xml`:

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

Output 1.3 *XML Document singers.xml*

```
<?xml version="1.0" encoding="utf-8" ?>
<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 XMLV2 Engine

The XMLV2 engine supports the following processing:

- The engine supports input (read) and output (create) processing. The engine does not support update processing.
- The 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 engine does not provide random (direct) access, which is required for some SAS applications and features. For example, 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 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 import an XML document that was created in a different character encoding, be aware of the possibility of transcoding errors. If an XML document or XMLMap does not specify an ENCODING= attribute in the XML declaration, then the engine attempts to identify the encoding from a byte-order mark. This behavior is consistent with World Wide Web Consortium (W3C) specifications. If neither an ENCODING= attribute nor a byte-order mark is found, the default for the XMLV2 engine is the session encoding. If the document's encoding is not compatible with the session encoding, a transcoding error could occur.

When you export an XML document using the XMLV2 engine, by default, the XML document contains an encoding attribute in the XML declaration from the SAS data set's encoding. Here is an example:

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

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 XMLV2 Engine a DOM or SAX Application?

The XMLV2 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 XMLV2 Engine Validate an XML Document?

The XMLV2 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. Successful processing without log messages does not ensure that the XML markup is valid. You must check the data to ensure that it is complete.

What Is the Difference between Using the XMLV2 Engine and Using ODS MARKUP?

The XMLV2 engine creates and reads XML documents. ODS MARKUP creates, but does not read, XML documents. Typically, you use the 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 XMLV2 engine reads 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 can 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 engine how to interpret the XML markup into a SAS data set or data sets, variables (columns), and observations (rows). See Chapter 4, “Importing XML Documents Using an XMLMap,” on page 21.

What Are the XMLV2 and XML Engines?

SAS provides two versions of XML LIBNAME engine functionality by supporting the two engine names **XMLV2** and **XML** in the LIBNAME statement. See “Comparing the XMLV2 and XML Engines” on page 76.

Accessibility Features of the XMLV2 Engine

The XMLV2 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.

2

Importing XML Documents

Understanding How to Import an XML Document

Importing an XML document is the process of reading an external XML document as a SAS data set. The XMLV2 engine translates the input XML document to the SAS proprietary file format.

To import an XML document, you execute the LIBNAME statement for the XMLV2 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 22.

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

```
<TABLE>
   <CLASS>
      <Name> Alfred </Name>
      <Gender> M </Gender>
      <Age> 14 </Age>
      <Height> 69 </Height>
      <Weight> 112.5 </Weight>
   </CLASS>
   <CLASS>
      <Name> Alice </Name>
      <Gender> F </Gender>
      <Age> 13 </Age>
      <Height> 56.5 </Height>
      <Weight> 84 </Weight>
   </CLASS>
.
.
.
   <CLASS>
      <Name> William </Name>
      <Gender> M </Gender>
      <Age> 15 </Age>
      <Height> 66.5 </Height>
      <Weight> 112 </Weight>
   </CLASS>
</TABLE>
```

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

```
libname trans xmlv2 'c:\example\class.xml'; /* 1 */

libname myfiles 'c:\output\'; /* 2 */

data myfiles.class; /* 3 */
   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 XMLV2 engine. By default, the XMLV2 engine expects GENERIC markup.
2. The second LIBNAME statement assigns the libref `myfiles` to the physical location of the SAS library that will store the resulting SAS data set. The V9 engine is the default.
3. The DATA step reads the XML document and writes its content in SAS proprietary format.

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

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

Output 2.1 *PRINT Procedure Output for myfiles.class*

The SAS System

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

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 ampersand (&) and the apostrophe (') must be escaped using character references or strings like `&` and `'`. For example, to allow attribute values to contain both single and double quotation marks, the apostrophe or single-quotation character (') can be represented as `'` and the double-quotation character (") as `"`.

To import an XML document that contains non-escaped characters, you can specify the LIBNAME statement option XMLPROCESS=PERMIT. When that option is specified, the XMLV2 engine accepts non-escaped characters like the apostrophe, double quotation marks, and the ampersand in character data. (Non-escaped angle

brackets are not supported.) See "XMLPROCESS= LIBNAME Statement Option" on page 101.

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 `strings.xml`, which contains non-escaped character data:

```
<?xml version="1.0" ?>
<STRINGS>
   <CHARS>
      <status>proper escape sequence</status>
      <ampersand>&amp;</ampersand>
      <squote>&apos;</squote>
      <dquote>&quot;</dquote>
      <greater>&gt;</greater>
   </CHARS>
   <CHARS>
       <status>unescaped character in CDATA</status>
       <ampersand><![CDATA[Abbott & Costello]]></ampersand>
       <squote><![CDATA[Logan's Run]]></squote>
       <dquote><![CDATA[As Benjamin Franklin advised, "Well done is
               better than well said."]]></dquote>
       <greater><![CDATA[ x > y ]]></greater>
  </CHARS>
   <CHARS>
      <status>unescaped character in string</status>
      <ampersand>Dunn & Bradstreet</ampersand>
      <squote>Isn't this silly?</squote>
      <dquote>Quoth the raven, "Nevermore!"</dquote>
      <greater> > </greater>
   </CHARS>
</STRINGS>
```

The example code below shows the default behavior, which is XMLPROCESS=CONFORM. Under the default, the XMLV2 engine expects XML markup to conform to W3C specifications. The following LIBNAME statement results in errors, and the `myfiles` libref is not assigned.

```
libname myfiles xmlv2 'c:\example\strings.xml';
```

Example Code 2.1 *SAS Log Output*

```
1    libname myfiles xmlv2 'c:\example\strings.xml';
ERROR: There is an illegal character in the entity name.
       occurred at or near line 19, column 24
ERROR: XML parsing error. Please verify that the XML content is well-formed.
ERROR: Error in the LIBNAME statement.
```

Specifying the LIBNAME statement option XMLPROCESS=PERMIT enables the XMLV2 engine to import the XML document:

```
libname myfiles xmlv2 'c:\example\strings.xml' xmlprocess=permit;
proc print data=myfiles.chars;
run;
```

Output 2.2 *PRINT Procedure Output for myfiles.chars*

Obs	status	ampersand	squote	dquote	greater
1	proper escape sequence	&	'	"	>
2	unescaped character in CDATA	Abbott & Costello	Logan's Run	As Benjamin Franklin advised, "Well done is better than well said."	x > y
3	unescaped character in string	Dunn & Bradstreet	Isn't this silly?	Quoth the raven, "Nevermore!"	>

3

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 XMLV2 LIBNAME 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 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 That Contains 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 variables generate their values from the DATETIME(), DATE(), and TIME() functions.

```
data test;
   Var1=datetime();
   format Var1 datetime.;
   Var2=date();
   format Var2 date9.;
   Var3=time();
   format Var3 timeampm.;

proc print data=test;
run;
```

Output 3.1 *PRINT Procedure Output for work.test Containing SAS Datetime, Date, and Time*

The SAS System

Obs	Var1	Var2	Var3
1	10OCT17:13:36:32	10OCT2017	1:36:32 PM

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

```
libname trans xmlv2 'c:\output\test.xml' xmltype=generic; /*1*/

data trans.test; /*2*/
   set work.test;
run;
```

1 The LIBNAME statement assigns the libref `trans` to the physical location of the file that will store the exported XML document and specifies the XMLV2 engine. The physical location includes the complete pathname, filename, and file extension. XMLTYPE= specifies the GENERIC markup type, which is the default.

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

Here is the resulting XML document.

Output 3.2 *XML Document Using GENERIC Markup*

```
<?xml version="1.0" encoding="utf-8" ?>
<TABLE>
   <TEST>
      <Var1>2017-10-10T13:36:32</Var1>
      <Var2>2017-10-10</Var2>
      <Var3>13:36:32</Var3>
   </TEST>
</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 from the `sashelp` library.

First, here is the CONTENTS procedure output for the SAS data set `sashelp.snacks`:

Output 3.3 *CONTENTS Procedure Output for sashelp.snacks*

Alphabetic List of Variables and Attributes					
#	Variable	Type	Len	Format	Label
3	Advertised	Num	8		Advertised (1=yes)
5	Date	Num	8	DATE9.	Date of sale
4	Holiday	Num	8		Holiday (1=yes)
2	Price	Num	8		Retail price of product
6	Product	Char	40		Product name
1	QtySold	Num	8		Quantity sold

The following SAS program exports an XML document from the SAS data set `sashelp.snacks`:

```
filename myxsd 'c:\output\snacks.xsd'; /* 1 */

libname output xmlv2 'c:\output\snacks.xml'
   xmlmeta=schemadata xmlschema=myxsd; /* 2 */

data output.snacks; /* 3 */
   set sashelp.snacks;
run;
```

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

2 The LIBNAME statement assigns the libref `output` to the physical location of the file that will store the exported XML document and specifies the XMLV2 engine. Here are the engine options:

- 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.

3 The DATA step reads the SAS data set `sashelp.snacks` and writes its data content in XML markup to the XML document `snacks.xml`. The DATA step also writes the metadata information to the separate external file `snacks.xsd`.

Part of the resulting XML document is shown in Output 3.4 on page 18. The separate metadata information is shown in Output 3.5 on page 19.

Output 3.4 *XML Document snacks.xml*

```
<?xml version="1.0" encoding="utf-8" ?>
<TABLE xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="snacks.xsd">
   <SNACKS>
      <QtySold>0</QtySold>
      <Price>1.99</Price>
      <Advertised>0</Advertised>
      <Holiday>0</Holiday>
      <Date>2002-01-01</Date>
      <Product>Baked potato chips</Product>
   </SNACKS>
   <SNACKS>
      <QtySold>0</QtySold>
      <Price>1.99</Price>
      <Advertised>0</Advertised>
      <Holiday>0</Holiday>
      <Date>2002-01-02</Date>
      <Product>Baked potato chips</Product>
   </SNACKS>
     .
     .
     .
</TABLE>
```

Output 3.5 *Separate Metadata Information snacks.xsd*

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="TABLE">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="SNACKS"  minOccurs="0" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="SNACKS">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="QtySold" minOccurs="0" type="xs:double" />
            <xs:element name="Price" minOccurs="0" type="xs:double" />
            <xs:element name="Advertised" minOccurs="0" type="xs:double" />
            <xs:element name="Holiday" minOccurs="0" type="xs:double" />
            <xs:element name="Date" minOccurs="0" type="xs:date" />
            <xs:element name="Product" minOccurs="0">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:maxLength value="40" />
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

4

Importing XML Documents Using an XMLMap

Why Use an XMLMap When Importing?

The XMLV2 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, you can use an XMLMap to interpret the XML markup. An XMLMap contains specific XMLMap syntax (which is in itself XML markup). The XMLMap syntax tells the XMLV2 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 12, “Using SAS XML Mapper to Generate and Update an XMLMap,” on page 129.

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 “AUTOMAP= LIBNAME Statement Option” on page 85.

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 version="1.0" encoding="windows-1252" ?>
<LIBRARY> <!-- 1 -->
   <STUDENTS> <!-- 2 -->
      <ID> 0755 </ID>
      <NAME> Brad Martin </NAME>
      <ADDRESS> 1611 Glengreen </ADDRESS>
      <CITY> Huntsville </CITY>
      <STATE> Texas </STATE>
   </STUDENTS>

   <STUDENTS> <!-- 3 -->
      <ID> 1522 </ID>
      <NAME> Zac Harvell </NAME>
      <ADDRESS> 11900 Glenda </ADDRESS>
      <CITY> Houston </CITY>
      <STATE> Texas </STATE>
   </STUDENTS>
.
.  more instances of <STUDENTS>
.
</LIBRARY>
```

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

1. The XMLV2 engine recognizes <LIBRARY> as the root-enclosing element.
2. The engine goes to the second-level instance tag, which is <STUDENTS>, and translates it as the data set name. The engine 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 XMLV2 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 xmlv2 'c:\example\students.xml';

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

Output 4.1 *PRINT Procedure Output for test.students*

The SAS System

Obs	ID	NAME	ADDRESS	CITY	STATE
1	755	Brad Martin	1611 Glengreen	Huntsville	Texas
2	1522	Zac Harvell	11900 Glenda	Houston	Texas

Why Is a Specific Physical Structure Required?

Well-formed XML is determined by structure, not content. Although the XMLV2 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 XMLV2 engine has to account for the possibility of multiple nodes (that is, multiple SAS data sets).

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

```
<?xml version="1.0" encoding="windows-1252" ?>
<CLIMATE> <!-- 1 -->
   <HIGHTEMP> <!-- 2 -->
      <PLACE> Libya </PLACE>
      <DATE> 1922-09-13 </DATE>
      <DEGREE-F> 136 </DEGREE-F>
      <DEGREE-C> 58 </DEGREE-C>
   </HIGHTEMP>
.
.  more instances of <HIGHTEMP>
.
   <LOWTEMP> <!-- 3 -->
      <PLACE> Antarctica </PLACE>
      <DATE> 1983-07-21 </DATE>
      <DEGREE-F> -129 </DEGREE-F>
      <DEGREE-C> -89 </DEGREE-C>
   </LOWTEMP>
.
.  more instances of <LOWTEMP>
.
</CLIMATE>
```

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

1. The XMLV2 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 XMLV2 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 XMLV2 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:

```
libname climate xmlv2 'C:\Example\climate.xml';

proc datasets library=climate;
quit;
```

Output 4.2 *DATASETS Procedure Output for climate Library*

The SAS System

Directory	
Libref	CLIMATE
Engine	XMLV2
Physical Name	C:\Example\climate.xml
XMLType	SAS XML Generic
Version	.

#	Name	Member Type
1	HIGHTEMP	DATA
2	LOWTEMP	DATA

Handling XML Documents That Are Not in the Required Physical Structure

If your XML document is not in the required physical structure, you can tell the XMLV2 engine how to interpret the XML markup to successfully import the document. See "Why Use an XMLMap When Importing?" on page 22.

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 XMLV2 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 version="1.0" encoding="iso-8859-1" ?>
<NHL>
 <CONFERENCE> Eastern
   <DIVISION> Southeast
     <TEAM name="Thrashers"  abbrev="ATL" />
     <TEAM name="Hurricanes" abbrev="CAR" />
     <TEAM name="Panthers"   abbrev="FLA" />
     <TEAM name="Lightning"  abbrev="TB" />
     <TEAM name="Capitals"   abbrev="WSH" />
   </DIVISION>
 </CONFERENCE>

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

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

```
<?xml version="1.0" ?>
<SXLEMAP version="2.1">
  <TABLE name="TEAMS"> <!-- 1 -->
    <TABLE-PATH syntax="XPath">
       /NHL/CONFERENCE/DIVISION/TEAM
    </TABLE-PATH> <!-- 2 -->

    <COLUMN name="NAME"> <!-- 3 -->
      <PATH>
         /NHL/CONFERENCE/DIVISION/TEAM/@name
      </PATH> <!-- 5 -->
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>30</LENGTH>
    </COLUMN>

    <COLUMN name="ABBREV"> <!-- 3 -->
      <PATH>
         /NHL/CONFERENCE/DIVISION/TEAM/@abbrev
      </PATH> <!-- 5 -->
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>3</LENGTH>
    </COLUMN>
```

```
    <COLUMN name="CONFERENCE" retain="YES"> <!--4-->
      <PATH>/NHL/CONFERENCE</PATH> <!--5-->
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>10</LENGTH>
    </COLUMN>

    <COLUMN name="DIVISION" retain="YES"> <!--4-->
      <PATH>
         /NHL/CONFERENCE/DIVISION
      </PATH> <!--5-->
      <TYPE>character</TYPE>
      <DATATYPE>STRING</DATATYPE>
      <LENGTH>10</LENGTH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

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

1. Locate and identify distinct tables of information.

   You want a SAS data set (table) that contains some of the teams of the National Hockey League. Because that is the only information contained in the XML document, you can define a single data set named `teams` in the XMLMap. (Note that other XML documents might contain more than one table of related information. Importing multiple tables is supported by the XMLMap syntax as shown in "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 29.)

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`:

```
filename nhl 'c:\example\nhl.xml'; /* 1 */
filename map 'c:\example\nhl.map'; /* 2 */

libname nhl xmlv2 xmlmap=map; /* 3 */

proc print data=nhl.teams; /* 4 */
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 4.3 *PRINT Procedure Output for nhl.teams*

The SAS System

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

The following code is not necessary for this example, but it creates the data set on disk to use in "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 59.

```
libname myfiles 'c:\myfiles\';
data myfiles.teams;
   set nhl.teams;
```

```
run;
```

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

This example explains how to create and use an XMLMap to import an XML document as two SAS data sets. The example uses the XML document `rss.xml` below. The XML document does not import successfully without an XMLMap because its XML markup is a nonstandard structure. If you attempt to import the XML document without an XMLMap, an error is written to the log. The errors states that the XML data is not in a format supported natively by the engine. The XML content in this example uses a simplified set of RSS (rich site summary) tags.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>My RSS Channel</title>
  <description>This is a simplified example of an RSS feed.</
description>
  <link>http://www.example.com/main.html</link>
  <language>en-us</language>

  <item>
   <title>My news item</title>
   <description>This is a detailed summary of my news item.</
description>
   <link>http://www.example.com/blog/post/1</link>
  </item>

  <item>
   <title>Another news item</title>
   <description>This description is shorter.</description>
   <link>http://www.example.com/blog/post/2</link>
  </item>
</channel>
</rss>
```

The XML document can be successfully imported by creating an XMLMap that defines the XML elements as the columns and other attributes of one or more data sets. The following XMLMap, which is named `rss.map`, defines two SAS data sets. The `channel` data set contains content information. The `item` data set contains individual news stories.

```
<?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="Channel_Title"> <!-- 4 -->
```

```
      <PATH syntax="XPath">/rss/channel/title</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>200</LENGTH>
    </COLUMN>

    <COLUMN name="Channel_URL">
      <PATH syntax="XPath">/rss/channel/link</PATH>
      <DESCRIPTION>Channel link</DESCRIPTION>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>200</LENGTH>
    </COLUMN>

    <COLUMN name="Channel_Description">
      <PATH syntax="XPath">/rss/channel/description</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>1024</LENGTH>
    </COLUMN>

  </TABLE>

  <TABLE description="Individual news stories" name="ITEM"> <!-- 5 -->
    <TABLE-PATH syntax="XPath">/rss/channel/item</TABLE-PATH> <!-- 6 -->

    <COLUMN name="Title"> <!-- 7 -->
      <PATH syntax="XPath">/rss/channel/item/title</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>200</LENGTH>
    </COLUMN>

    <COLUMN name="URL">
      <PATH syntax="XPath">/rss/channel/item/link</PATH>
      <DESCRIPTION>Item link</DESCRIPTION>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>200</LENGTH>
    </COLUMN>

    <COLUMN name="Description">
      <PATH syntax="XPath">/rss/channel/item/description</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>1024</LENGTH>
    </COLUMN>

  </TABLE>

</SXLEMAP>
```

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

2 This TABLE element defines the `channel` data set.

3 This TABLE-PATH element specifies the location path in the XML document to collect variables for the `channel` data set.

4 This COLUMN element defines the `Channel_Title` variable in the `channel` data set. The XPath construction specifies where to find the current tag and where to access data from the named element. The next two COLUMN elements define the `Channel_URL` and `Channel_Description` variables.

5 This TABLE element defines the `item` data set.

6 This TABLE-PATH element specifies the location path in the XML document to collect variables for the `item` data set.

7 This COLUMN element defines the `Title` variable in the `item` data set. The next two COLUMN elements define the `URL` and `Description` variables.

The following SAS statements import the XML document `rss.xml` and specify the XMLMap named `rss.map`. The DATASETS procedure verifies the import results.

```
filename myrss 'C:\Example\rss.xml';
filename mymap 'C:\Example\rss.map';

libname myrss xmlv2 xmlmap=mymap access=readonly;

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

Output 4.4 *DATASETS Procedure Output for myrss Library Showing Two Data Sets*

The SAS System

Directory	
Libref	MYRSS
Engine	XMLV2
Access	READONLY
Physical Name	C:\Example\rss.xml
XMLType	SAS XMLMap
Version	2.1

#	Name	Member Type
1	CHANNEL	DATA
2	ITEM	DATA

Importing Hierarchical Data as Related Data Sets

XML documents often contain hierarchical data in that the data is structured into different levels like a company organization chart. Hierarchical structures are one-to-many relationships. 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 version="1.0" ?>
<PHARMACY>
  <PERSON>
    <NAME>Brad Martin</NAME>
    <STREET>11900 Glenda Court</STREET>
    <CITY>Austin</CITY>
    <PRESCRIPTION>
      <NUMBER>1234</NUMBER>
      <DRUG>Tetracycline</DRUG>
    </PRESCRIPTION>
    <PRESCRIPTION>
      <NUMBER>1245</NUMBER>
      <DRUG>Lomotil</DRUG>
    </PRESCRIPTION>
  </PERSON>
  <PERSON>
    <NAME>Jim Spano</NAME>
    <STREET>1611 Glengreen</STREET>
    <CITY>Austin</CITY>
    <PRESCRIPTION>
      <NUMBER>1268</NUMBER>
      <DRUG>Nexium</DRUG>
    </PRESCRIPTION>
  </PERSON>
</PHARMACY>
```

To import separate data sets, one describing the customers and the other containing prescription information, a relation between each customer and associated prescriptions must be designated. The relationship determines 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.

```
<?xml version="1.0" encoding="windows-1252"?>
<!-- ############################################################ -->
<!-- 2017-10-21T11:57:46 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904500.0.0.20170816190000_v940m5 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- Column (NAME) in table (PRESCRIPTION) has an XPath outside the
     scope of the table path. The contents of this column may not
     correspond to other row values and/or may be missing entirely.
 1 -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="Pharmacy" version="2.1"> <!-- 2 -->

  <NAMESPACES count="0"/>

  <!-- ########################################################## -->
  <TABLE description="PERSON" name="PERSON"> <!-- 3 -->
    <TABLE-PATH syntax="XPath">/PHARMACY/PERSON</TABLE-PATH>

    <COLUMN name="NAME"> <!-- 4 -->
      <PATH syntax="XPath">/PHARMACY/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>11</LENGTH>
    </COLUMN>

    <COLUMN name="STREET"> <!-- 4 -->
      <PATH syntax="XPath">/PHARMACY/PERSON/STREET</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>18</LENGTH>
    </COLUMN>

    <COLUMN name="CITY"> <!-- 4 -->
      <PATH syntax="XPath">/PHARMACY/PERSON/CITY</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>6</LENGTH>
    </COLUMN>

  </TABLE>

  <!-- ########################################################## -->
  <TABLE description="PRESCRIPTION" name="PRESCRIPTION"> <!-- 5 -->
    <TABLE-PATH syntax="XPath">/PHARMACY/PERSON/PRESCRIPTION</TABLE-
PATH>
```

```
    <COLUMN name="NAME" retain="YES"> <!-- 6 -->
      <PATH syntax="XPath">/PHARMACY/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>11</LENGTH>
    </COLUMN>

    <COLUMN name="NUMBER"> <!-- 7 -->
      <PATH syntax="XPath">/PHARMACY/PERSON/PRESCRIPTION/NUMBER</PATH>
      <TYPE>numeric</TYPE>
      <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="DRUG"> <!-- 7 -->
      <PATH syntax="XPath">/PHARMACY/PERSON/PRESCRIPTION/DRUG</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>12</LENGTH>
    </COLUMN>

  </TABLE>

</SXLEMAP>
```

1. If the SAS XML Mapper application issues a warning, a good practice is to print the data set and check for errors. In this case, no error occurs.
2. SXLEMAP is the root-enclosing element for the two SAS data set definitions.
3. First TABLE element defines the `person` data set.
4. COLUMN elements contain the attributes for the `name`, `street`, and `city` variables in the `person` data set.
5. Second TABLE element defines the `prescription` data set.
6. 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.)
7. COLUMN elements contain the attributes for the `number` and `drug` variables in the `prescription` data set.

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

```
filename pharm 'C:\Example\Pharmacy.xml';
filename map 'C:\Example\Pharmacy.map';

libname pharm xmlv2 xmlmap=map;
```

The following code verifies that SAS interprets the XML document `pharmacy.xml` as two SAS data sets: `pharm.person` and `pharm.prescription`.

```
proc datasets library=pharm;
quit;
proc print data=pharm.person;
run;
proc print data=pharm.prescription;
run;
```

Output 4.5 *DATASETS Procedure Output for pharm Library*

The SAS System

Directory	
Libref	PHARM
Engine	XMLV2
Physical Name	C:\Example\Pharmacy.xml
XMLType	SAS XMLMap
Version	2.1

#	Name	Member Type
1	PERSON	DATA
2	PRESCRIPTION	DATA

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

Output 4.6 *PRINT Procedure Output for pharm.person*

The SAS System

Obs	NAME	STREET	CITY
1	Brad Martin	11900 Glenda Court	Austin
2	Jim Spano	1611 Glengreen	Austin

Output 4.7 *PRINT Procedure Output for pharm.prescription*

The SAS System

Obs	NAME	NUMBER	DRUG
1	Brad Martin	1234	Tetracycline
2	Brad Martin	1245	Lomotil
3	Jim Spano	1268	Nexium

Including a Key Field with Generated Numeric Keys

This example imports the XML document `pharmacy.xml`, which contains hierarchical data and is used in the example "Importing Hierarchical Data as Related Data Sets" on page 32. 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 version="1.0" encoding="windows-1252"?>
<!-- ############################################################ -->
<!-- 2017-10-23T11:00:08 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904500.0.0.20170816190000_v940m5 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="PharmacyOrdinal" version="2.1">

  <NAMESPACES count="0"/>

  <!-- ########################################################## -->
  <TABLE description="PERSON" name="PERSON">
    <TABLE-PATH syntax="XPath">/PHARMACY/PERSON</TABLE-PATH> <!-- 1 -->

    <COLUMN class="ORDINAL" name="KEY" retain="YES"> <!-- 2 -->
      <INCREMENT-PATH beginend="BEGIN" 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">
  <TABLE-PATH syntax="XPath">/PHARMACY/PERSON/PRESCRIPTION
  </TABLE-PATH> <!-- 3 -->

  <COLUMN class="ORDINAL" name="KEY" retain="YES"> <!-- 4 -->
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/PHARMACY/PERSON
    </INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
    <FORMAT width="3">Z</FORMAT>
  </COLUMN>

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

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


</TABLE>
```

```
</SXLEMAP>
```

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 XMLV2 engine follows this process to generate the key field values for the `person` data set:
   1. When the XMLV2 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 XMLV2 engine continues reading values into the input buffer until it encounters the </PERSON> end tag. At this time, the engine 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 XMLV2 engine follows this process to generate the key field values for the `prescription` data set:
   1. When the XMLV2 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 engine continues reading values into the input buffer until it encounters the </PRESCRIPTION> end tag. At this time, the engine 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 engine for the `prescription` data set `key` variable is the same as the `person` data set `key` variable. Although the 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 and prints the data sets:

```
filename pharm 'c:\example\pharmacy.xml';
filename map 'c:\example\pharmacyordinal.map';

libname pharm xmlv2 xmlmap=map;
proc print data=pharm.person;
run;
```

```
proc print data=pharm.prescription;
run;
```

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

Output 4.8 *PRINT Procedure Output for pharm.person with a Numeric Key*

The SAS System

Obs	KEY	NAME	STREET	CITY
1	001	Brad Martin	11900 Glenda Court	Austin
2	002	Jim Spano	1611 Glengreen	Austin

Output 4.9 *PRINT Procedure Output for pharm.prescription with a Numeric Key*

The SAS System

Obs	KEY	NUMBER	DRUG
1	001	1234	Tetracycline
2	001	1245	Lomotil
3	002	1268	Nexium

Determining the Observation Boundary to Select the Best Columns for an XMLMap

This example illustrates how to determine the observation boundary so that the import results in the best collection of columns.

The observation boundary defines a collection of rows with a constant set of columns. In an XMLMap, you set the observation boundary with the TABLE-PATH element, which contains a location path.

In the following document, `report.xml`, Publication appears to be a possible element to use as the observation boundary. This boundary would result in an import of the columns `title`, `acquired`, and `topic`. However, because the TOPIC element occurs multiple times within the second and third PUBLICATION container, the import would include multiple `topic` columns. The user would typically want

each observation to contain only one `topic` column, so the TOPIC element is the better choice to use as the observation boundary.

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

Here is `report.map`, which you can use to import the `report.xml` document:

```
<?xml version="1.0" ?>
<SXLEMAP version="2.1">
  <TABLE name="Publication">
    <TABLE-PATH syntax="XPath">
       /Library/Publication/Topic
    </TABLE-PATH> <!--1-->

    <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> <!--2-->
      <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> <!-- 3 -->
        <VALUE>Y</VALUE>
        <VALUE>N</VALUE>
      </ENUM>
     <DEFAULT>N</DEFAULT> <!-- 4 -->
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

The XMLMap tells the engine how to interpret the XML markup:

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 by using the XMLMap syntax FORMAT element. Elements like FORMAT and INFORMAT are useful when data must be converted for use by SAS. The XMLV2 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 that are not contained within the ENUM list are set to MISSING.
4. By default, a missing value is set to MISSING. Here, the DEFAULT element specifies N as the value for a missing value. When the ENUM element is used, a value specified by DEFAULT must be one of the ENUM values.

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

```
filename myreps 'c:\example\report.xml';
filename mymap 'c:\example\report.map';
libname myreps xmlv2 xmlmap=mymap;
proc print data=myreps.publication noobs;
run;
```

Output 4.10 *PRINT Procedure Output for myreps.publication Data Set*

The SAS System

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

Using ISO 8601 SAS Informats and Formats to Import Dates

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

Here is the XML document:

```
<?xml version="1.0" ?>
<Root>
  <ISODATE>
    <BASIC>19450508</BASIC>
    <EXTENDED>1945-05-08</EXTENDED>
  </ISODATE>
</Root>
```

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

```
<?xml version="1.0" encoding="windows-1252"?>
<!-- ############################################################ -->
<!-- 2017-10-23T13:33:48 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904500.0.0.20170816190000_v940m5 -->
<!-- ########################################################## -->
<!-- ###  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"/>

    <!-- ############################################################
-->
    <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> <!-- 1 -->
            <INFORMAT width="8">B8601DA</INFORMAT> <!-- 2 -->
        </COLUMN>

        <COLUMN name="EXTENDED">
            <PATH syntax="XPath">/Root/ISODATE/EXTENDED</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>date</DATATYPE>
            <FORMAT width="10">E8601DA</FORMAT> <!-- 3 -->
            <INFORMAT width="10">E8601DA</INFORMAT> <!-- 4 -->
        </COLUMN>

    </TABLE>

</SXLEMAP>
```

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

1. For the `basic` variable, the FORMAT element specifies the E8601DA SAS format, which writes data values in the extended format *yyyy-mm-dd*.
2. For the `basic` variable, the INFORMAT element specifies the B8601DA SAS informat, which reads date values into a variable in the basic format *yyyymmdd*.

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

3. For the `extended` variable, the FORMAT element specifies the E8601DA SAS format, which writes data values in the extended format *yyyy-mm-dd*.
4. For the `extended` variable, the INFORMAT element specifies the E8601DA SAS informat, which reads date values into a variable in the extended format *yyyy-mm-dd*.

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

```
filename dates 'c:\example\isodate.xml';
filename map 'c:\example\isodate.map';

libname dates xmlv2 xmlmap=map;
```

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

Output 4.11 *PRINT Procedure Output for Imported Data Set dates.isodate*

The SAS System

Obs	BASIC	EXTENDED
1	1945-05-08	1945-05-08

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

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

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

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- 2011-01-11T13:31:41 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 903000.1.0.20101208190000_v930 -->
<!-- ############################################################ -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="ISOtime" version="2.1">
```

```
    <NAMESPACES 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> <!-- 1 -->
            <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> <!-- 2 -->
            <INFORMAT width="8">E8601TM</INFORMAT>
        </COLUMN>

        <COLUMN name="UTC">
            <PATH syntax="XPath">/Root/TIME/UTC</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>time</DATATYPE>
            <FORMAT width="9">E8601TZ</FORMAT> <!-- 3 -->
            <INFORMAT width="9">E8601TZ</INFORMAT>
        </COLUMN>

        <COLUMN name="OFFSET">
            <PATH syntax="XPath">/Root/TIME/OFFSET</PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>time</DATATYPE>
            <FORMAT width="14">E8601TZ</FORMAT> <!-- 4 -->
            <INFORMAT width="14">E8601TZ</INFORMAT>
        </COLUMN>

    </TABLE>

</SXLEMAP>
```

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

1. For the `local` variable, the INFORMAT and FORMAT elements specify the E8601TM SAS informat and format, which reads and writes time values in the extended format *hh:mm:ss.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 *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 *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.

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

3 For the `utc` variable, the INFORMAT and FORMAT elements specify the E8601TZ SAS informat and format, which reads and writes time values in the extended format *hh:mm:ss*+|-*hh:mm*. Because there is a time zone indicator, the value is assumed to be expressed in UTC. No adjustment or conversion is made to the value.

4 For the `offset` variable, the INFORMAT and FORMAT elements specify the E8601TZ SAS informat and format, which reads and writes time values in the extended format *hh:mm:ss*+|-*hh:mm*. A time zone offset is 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. The time zone context is not stored with the value. When the time value is written using the time zone sensitive SAS format, the value is expressed as UTC with a zero offset value and is not adjusted to or from local time.

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

```
filename timzn 'c:\example\Time.xml';
filename map 'c:\example\Time.map';

libname timzn xmlv2 xmlmap=map;

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

Output 4.12 *PRINT Procedure Output for Imported Data Set timzn.time*

The SAS System

Obs	LOCAL	LOCALZONE	UTC	OFFSET
1	09:00:00	09:00:00-05:00	09:00:00Z	09:00:00+00:00

Referencing a Fileref Using the URL Access Method

Using several methods, the XMLV2 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 XMLV2

engine to process the XML document with a single pass of the file. If you do not specify an XMLMap, the engine performs a double pass.

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

```
filename mynhl url 'http://www.a.com/nhl.xml'; /*1*/
filename mymap 'c:\example\nhl.map'; /*2*/

libname mynhl xmlv2 xmlmap=mymap; /*3*/

proc print data=mynhl.teams; /*4*/
run;
```

1. The first FILENAME statement assigns the fileref `mynhl` to the XML document by using the URL access method.
2. The second FILENAME statement assigns the fileref `mymap` to the physical location of the XMLMap `nhl.map`.
3. The LIBNAME statement uses the fileref `mynhl` to reference the XML document, specifies the XMLV2 engine, and uses the fileref `mymap` to reference the XMLMap.
4. PROC PRINT produces output, verifying that the import was successful.

Output 4.13 *PRINT Procedure Output for mynhl.teams*

The SAS System

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

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 XMLV2 engine where in the XML document to locate and access a specific tag for the current variable. In addition, the location path tells the XMLV2 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 `nhlshort.xml` to be imported:

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

Here is the XMLMap `nhl1.map` used to import the XML document, with notations for each XPath form on the PATH element:

```
<?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> <!-- 1 -->
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>3</LENGTH>
         </COLUMN>

         <COLUMN name="FOUNDED">
           <PATH syntax="XPath">
              /NHL/CONFERENCE/DIVISION/TEAM/@founded[@abbrev="ATL"]
            </PATH> <!-- 2 -->
            <TYPE>character</TYPE>
```

```
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

       <COLUMN name="CONFERENCE" retain="YES">
          <PATH syntax="XPath">
             /NHL/CONFERENCE
           </PATH> <!-- 3 -->
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

        <COLUMN name="TEAM">
           <PATH syntax="XPath">
              /NHL/CONFERENCE/DIVISION/TEAM[@founded="1993"]
            </PATH> <!-- 4 -->
            <TYPE>character</TYPE>
            <DATATYPE>STRING</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

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

    </TABLE>
</SXLEMAP>
```

1. The `abbrev` variable uses the attribute form that selects values from a specific attribute. The 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.
2. 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.
3. 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`.
4. 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`.
5. The `team5` variable uses the element form that conditionally selects PCDATA from a named element based on a specific occurrence of the element. The

position function tells the 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:

```
filename nhl 'c:\example\nhlshort.xml';
filename map 'c:\example\nhl1.map';

libname nhl xmlv2 xmlmap=map;

proc print data=nhl.teams noobs;
run;
```

Output 4.14 *PRINT Procedure Output Showing Resulting Variables with Selected Values*

The SAS System

ABBREV	FOUNDED	CONFERENCE	TEAM	TEAM5
ATL	1999	Eastern		
CAR		Eastern		
FLA		Eastern	Panthers	
TB		Eastern		
WSH		Eastern		Capitals

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 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>
        <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>
        <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.

```
<SXLEMAP name="Namespace" version="2.1">

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

    <TABLE description="PERSON" name="PERSON"> <!--3-->
        <TABLE-PATH syntax="XPath">/PEOPLE/PERSON</TABLE-PATH>

        <COLUMN name="NAME"> <!--4-->
            <PATH syntax="XPath">/PEOPLE/PERSON/NAME</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>13</LENGTH>
        </COLUMN>

        <COLUMN name="ADDRESS"> <!--4-->
            <PATH syntax="XPathENR">/PEOPLE/PERSON/{1}ADDRESS</PATH>
<!--5-->
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>16</LENGTH>
        </COLUMN>

        <COLUMN name="PHONE"> <!--4-->
```

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

        <COLUMN name="ADDRESS1"> <!-- 4 -->
            <PATH syntax="XPathENR">/PEOPLE/PERSON/{3}ADDRESS</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>26</LENGTH>
        </COLUMN>

        <COLUMN name="PHONE1"> <!-- 4 -->
            <PATH syntax="XPathENR">/PEOPLE/PERSON/{3}PHONE</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>12</LENGTH>
        </COLUMN>

        <COLUMN name="ADDRESS2"> <!-- 4 -->
            <PATH syntax="XPathENR">/PEOPLE/PERSON/{2}ADDRESS</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>11</LENGTH>
        </COLUMN>

    </TABLE>

</SXLEMAP>
```

1. A NAMESPACES element contains NS elements for defining XML namespaces. The count= attribute specifies that there are three defined XML namespaces.
2. 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.
3. The XMLMap TABLE element contains the data set definition for the PERSON repeating element.
4. XMLMap COLUMN elements contain variable definitions for each nested element within PERSON, which includes NAME, ADDRESS, PHONE, ADDRESS1, PHONE1, and ADDRESS2.
5. 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:

```
filename ns 'c:\example\nssample.xml';
filename nsmap 'c:\example\nssample.map';

libname ns xmlv2 xmlmap=nsmap;

proc print data=ns.person noobs;
run;
```

Output 4.15 *PRINT Procedure Output for ns.person*

The SAS System

NAME	ADDRESS	PHONE	ADDRESS1	PHONE1	ADDRESS2
Joe Smith	1234 Elm Street	999-555-0011	2001 Office Drive, Box 101	999-555-0101	192.168.1.1
Jane Jones	9876 Main Street	999-555-0022	2001 Office Drive, Box 102	999-555-0102	172.16.1.2
Pat Perkinson	1395 Half Way	999-555-0033	2001 Office Drive, Box 103	999-555-0103	10.0.1.3

Using the AUTOMAP= Option to Generate an XMLMap

This example illustrates how to import an XML document using the AUTOMAP= option to automatically generate an XMLMap file. See "AUTOMAP= LIBNAME Statement Option" on page 85.

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). The AUTOMAP= option is supported by the XMLV2 engine only.

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 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`:

```
filename nhl 'c:\example\nhl.xml'; /* 1 */
filename map 'c:\output\nhlgenerate.map'; /* 2 */

libname nhl xmlv2 automap=replace xmlmap=map; /* 3 */

proc print data=nhl.team; /* 4 */
run;
```

1. The first FILENAME statement assigns the file reference `nhl` to the physical location (complete path name, file name, 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 assigned to 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 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. -->
<!-- ############################################################ -->
<SXLEMAP name="AUTO_GEN" version="2.1">

    <NAMESPACES count="0"/>

    <!-- ############################################################ -->
    <TABLE description="NHL" name="NHL">
        <TABLE-PATH syntax="XPath">/NHL</TABLE-PATH>

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

    </TABLE>

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

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

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

        <COLUMN name="TEAM_name">
            <PATH syntax="XPath">/NHL/CONFERENCE/DIVISION/TEAM/@name</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>10</LENGTH>
        </COLUMN>

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

    </TABLE>

</SXLEMAP>
```

Here is the PRINT procedure output.

Output 4.16 *PRINT Procedure Output for nhl.team*

The SAS System

Obs	DIVISION_ORDINAL	TEAM_ORDINAL	TEAM_name	TEAM_abbrev
1	1	1	Thrashers	ATL
2	1	2	Hurricanes	CAR
3	1	3	Panthers	FLA
4	1	4	Lightning	TB
5	1	5	Capitals	WSH
6	2	6	Stars	DAL
7	2	7	Kings	LA
8	2	8	Ducks	ANA
9	2	9	Coyotes	PHX
10	2	10	Sharks	SJ

Avoiding Truncation Errors When Importing in a Multi-Byte Encoding

To avoid truncation in a double-byte character set (DBCS) or a multi-byte character set (MBCS), use the CHARMULTIPLIER= or DERIVECHARMULTIPLIER= option to expand the column (variable) length of a generated XMLMap. The issue is that when an XMLMap is generated or created by SAS XML Mapper, the LENGTH element specifies the number of characters in a column. SAS XML Mapper does not have knowledge of your SAS session encoding. The XMLV2 engine does use the SAS session encoding, and it can use the correct number of bytes to represent a multi-byte character. For example, a character might be represented in the wlatin1 encoding as one byte, but in UTF-8, it is represented as two bytes. The XMLMap assigns a length of 1 to that character. The XMLV2 engine assigns a length of 2 to that character. Because of this difference, when your SAS session encoding uses more than one byte to represent a character, truncation could occur if the XMLMap column length does not accommodate the larger character size.

The example uses `nhlgenerate.map`, which is generated in "Using the AUTOMAP= Option to Generate an XMLMap" on page 53. (The XML content does not include multi-byte characters. The XML and XMLMap are used to show the expansion behavior only.)

```
filename nhl 'c:\example\nhl.xml';
filename map 'c:\output\nhlgenerate.map';

libname nhl xmlv2 xmlmap=map;
proc contents data=nhl.team;
```

```
run;
```

Here is a portion of the PROC CONTENTS output, showing the column lengths without expansion:

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	DIVISION_ORDINAL	Num	8
2	TEAM_ORDINAL	Num	8
4	TEAM_abbrev	Char	3
3	TEAM_name	Char	10

The following example code uses the CHARMULTIPLIER= option to expand column lengths:

```
libname nhl xmlv2 xmlmap=map charmultiplier=2.5;
proc contents data=nhl.team;
run;
```

Here is a portion of the PROC CONTENTS output, showing expanded column lengths for the two character variables:

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	DIVISION_ORDINAL	Num	8
2	TEAM_ORDINAL	Num	8
4	TEAM_abbrev	Char	7
3	TEAM_name	Char	25

5

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 same XMLMap or create a new one. The XMLMap syntax tells the XMLV2 engine how to map a SAS data set into a specific XML document structure.

To export an XML document using an XMLMap, specify the XMLV2 engine in the LIBNAME statement, and use the XMLMAP= option to specify the file. Using an XMLMap for exporting XML data is supported by the XMLV2 engine only, and not the older XML engine.

See the XMLMap syntax on page 112 for the version numbers that are supported for export.

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

This example modifies an existing XMLMap to export a SAS data set to a different XML document structure. For the original XMLMap and data set, see "Using an XMLMap to Import an XML Document as One SAS Data Set" on page 25.

Here is the `nhl.teams` data set to be exported:

Output 5.1 *PRINT Procedure Output of Data Set nhl.teams*

The SAS System

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

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

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

```
<?xml version="1.0" ?>
<SXLEMAP version="2.1"> <!-- 1 -->
  <OUTPUT> <!-- 2 -->
    <HEADING>
      <ATTRIBUTE name="description"
```

```
        value="Teams of the National Hockey League" /> <!-- 3 -->
    </HEADING>
    <TABLEREF name="TEAMS" /> <!-- 4 -->
  </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. See the XMLMap syntax on page 112 for the version numbers that are supported for export.
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 can contain the optional HEADING element.
3. The optional ATTRIBUTE element defines additional XML file attribute information. Each ATTRIBUTE element specifies a name/value pair. To be usable XML, the specification must be a valid XML attribute name and value. Each ATTRIBUTE element is included as an attribute of the root element of the exported document in the form *name=value*. See the result in the exported XML below.
4. The TABLEREF element references the name of the table to be exported. The TABLEREF element must reference the name of a TABLE element in the XMLMap.

The following SAS statements export the SAS data set named `myfiles.teams` to an XML document named `nhlout.xml`, using an XMLMap named `nhlexport.map`:

```
libname myfiles 'c:\myfiles\';
filename out 'c:\output\nhlout.xml';
libname out xmlv2 xmltype=xmlmap xmlmap='c:\example\nhlexport.map';
data out.TEAMS;
   set myfiles.teams;
run;
```

Notice in the code above that `TEAMS` is uppercase to match the casing in the XMLMap. Here is the resulting XML document:

```
<?xml version="1.0" encoding="utf-8" ?>
<!--  SAS XML Libname Engine (SAS92XML)
      SAS XMLMap Generated Output
      Version V.03.03M0P10122017
      Created 2017-10-13T15:54:55
  -->

<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>
```

6

Understanding and Using Tagsets for the XMLV2 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: Procedures 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 you export an XML document, do not attempt to import the XML document generated by that altered tagset. In that case, the 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 XMLV2 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. Submit the following PROC TEMPLATE code in a SAS session:

```
proc template;

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

   define tagset tagsets.custom ;
      notes "SAS XMLV2 engine output event model(interface)";

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


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

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


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


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

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


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

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


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


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

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


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


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


      define event table;
      start:
         unset $col_names;
         unset $col_types;
         unset $col_width;
         eval  $index      1;
```

```
         eval  $index_max  0;
         set   $TABLENAME name / if name;           /* LIBNAME ENGINE
*/
         set   $META XMLMETADATA / if XMLMETADATA ; /* LIBNAME ENGINE
*/
         set   $SCHEMA XMLSCHEMA / if XMLSCHEMA ;   /* LIBNAME ENGINE
*/
         break;
      finish:
         break;
      end;


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


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


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

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


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


      define event table_body;
      start:
```

```
      trigger EmitMeta ;
      break;
   finish:
      trigger EmitMeta ;
      break;
   end;

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

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


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

```
end;


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

   define event EmitMeta; /* 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  ;
         next $col_names;
         eval $index $index+1;
         done;
      put '   -->' CR ;
      break;
   finish:
      put '</' $LIBRARYNAME '>' ;
      break;
   end;


   define event EmitRow; /* 2 */
      ndent;
      put "<STUDENT>" CR ;
      ndent;

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

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


   define event EmitCol; /* 3 */
      unset $value;
      set $value $col_values[$name];
      put '<'  $name '>' ;
      put      $value ;
      put '</' $name '>' CR ;
```

```
            break;
         end;

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

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

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

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

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; /* 1 */
   set sashelp.class (obs=3);
run;

filename xmlout "c:\output\myclass.xml"; /* 2 */

libname xmlout xmlv2 xmltype=generic tagset=tagsets.custom; /* 3 */

data xmlout.class; /* 4 */
   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 path name, file name, and file extension).

3 The LIBNAME statement uses the fileref to reference the XML document and specifies the XMLV2 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 `myclass.xml` document in the format that is defined by the customized tagset.

Here is the resulting XML document:

Output 6.1 *Exported XML Document Using Customized Tagset*

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

PART 2

LIBNAME Statement Syntax

7

Introduction to the XMLV2 LIBNAME Statement

Using the LIBNAME Statement

For the XMLV2 engine, the LIBNAME statement assigns a SAS libref to either a SAS library that stores XML documents or to a specific XML document to import or export.

For examples, see the following usage topics:

- Chapter 2, “Importing XML Documents,” on page 9
- Chapter 3, “Exporting XML Documents,” on page 15
- Chapter 4, “Importing XML Documents Using an XMLMap,” on page 21
- Chapter 5, “Exporting XML Documents Using an XMLMap,” on page 59

See also the XMLV2 LIBNAME Engine Tip Sheet.

Comparing the XMLV2 and XML Engines

Overview of XMLV2 and XML Engine Differences

SAS provides two versions of functionality by implementing two engine names in the LIBNAME statement:

- The **XMLV2** engine supports current functionality and includes enhancements that have been made since SAS 9.2. (If you are running in a SAS 9.2 session, use the alias **XML92**.)
- The **XML** engine is a compatibility engine for legacy programs to support SAS 9.1.3 and earlier functionality.

The differences between the two engines are explained in the following topics.

XML Compliance

The XMLV2 engine is XML compliant, which means the XML markup must be well-formed and in valid construction that is in compliance with the W3C specifications. XML compliance could affect the following situations:

- XML documents that can be imported with the older XML engine might not pass the more strict parsing rules in the XMLV2 engine. For example, XML compliance means that the markup is case sensitive. Opening and closing tags must be written in the same case, such as **<BODY> ...</BODY>** and **<Message>...</Message>**. For the XMLV2 engine, 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 older XML engine might not work with the XMLV2 engine. For example, XML compliance means that the markup and the supported XPath syntax are case sensitive. In addition, the XMLMap markup must follow specific XMLMap rules. Tag names must be uppercase. Element attributes must be lowercase. An example is **<SXLEMAP version="2.1">**.

XMLMap Files

The XMLV2 engine supports XMLMap files starting with XMLMap version 1.2. The older XML engine supports all XMLMap files starting with XMLMap version 1.0.

The documented XMLMap syntax version is 2.1. See Chapter 10, “Introduction to XMLMap Syntax,” on page 107.

The XMLV2 engine supports using XMLMap for exporting and also supports XML namespaces.

LIBNAME Statement Functionality Enhancements for XMLV2

The XMLV2 engine 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.
- Using the GENERIC markup type, you can export a single XML document from multiple SAS data sets.
- Additional options and option values are available for the XMLV2 engine. Some legacy options or option values are not supported for the XMLV2 engine. The following topic covers these differences.

XMLV2 and XML Engine LIBNAME Statement Options

The following table lists the available LIBNAME statement options. The ■ symbol indicates whether the option is available for each engine.

Table 7.1 *LIBNAME Statement Options*

Task	Option	XML	XMLV2
Automatically generate an XMLMap file to import an XML document	“AUTOMAP= LIBNAME Statement Option” on page 85		■
Expand XMLMap column (variable) lengths by a multiplier value	“CHARMULTIPLIER= LIBNAME Statement Option” on page 87		■
Expand XMLMap column (variable) lengths by a default multiplier value that is based on the session encoding	“DERIVECHARMULTIPLIER= LIBNAME Statement Option” on page 87		■
Determine whether SAS formats are used with the GENERIC markup type	“FORMATACTIVE= LIBNAME Statement Option” on page 88	■	■
Determine whether SAS formats are used with the CDISCODM markup type	“FORMATACTIVE= LIBNAME Statement Option” on page 88	■	

Task	Option	XML	XMLV2
Specify the libref to create a format catalog with the CDISCODM markup type	“FORMATLIBRARY= LIBNAME Statement Option” on page 89	■	
Replace existing format entries in the format catalog with the CDISCODM markup type	“FORMATNOREPLACE= LIBNAME Statement Option” on page 90	■	
Indent nested elements in exported XML document	“INDENT= LIBNAME Statement Option” on page 90	■	■
Specify the character set to use for the output file	“ODSCHARSET= LIBNAME Statement Option” on page 91	■	■
Control the generation of a record separator	“ODSRECSEP= LIBNAME Statement Option” on page 92	■	
Specify the translation table to use for the output file	“ODSTRANTAB= LIBNAME Statement Option” on page 93	■	■
Specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element	“PREFIXATTRIBUTES= LIBNAME Statement Option” on page 93		■
Override the default tagset	“TAGSET= LIBNAME Statement Option” on page 94	■	■
Import concatenated XML documents	“XMLCONCATENATE= LIBNAME Statement Option” on page 95	■	
Specify the tag format to contain SAS variable information	“XMLDATAFORM= LIBNAME Statement Option” on page 95	■	■
Control the results of numeric values	“XMLDOUBLE= LIBNAME Statement Option” on page 96	■	■
Override the SAS data set's encoding for the output file	“XMLENCODING= LIBNAME Statement Option” on page 97	■	■
Specify a fileref for the XML document	“XMLFILEREF= LIBNAME Statement Option” on page 98	■	■
Specify an XMLMap	“XMLMAP= LIBNAME Statement Option” on page 99	■	■

Task	Option	XML	XMLV2
Determine whether metadata-related information is included	“XMLMETA= LIBNAME Statement Option” on page 100	■	■
Determine whether to process nonconforming character data	“XMLPROCESS= LIBNAME Statement Option” on page 101	■	■
Specify an external file to contain exported metadata-related information	“XMLSCHEMA= LIBNAME Statement Option” on page 101	■	■
Specify the XML markup type	“XMLTYPE= LIBNAME Statement Option” on page 102	■	■

8

LIBNAME Statement

Dictionary

LIBNAME Statement: XMLV2 and XML Engines

Processes an XML document.

Valid in: Anywhere

Category: Data Access

Syntax

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

Syntax Description

libref
is a valid SAS name that serves as a shortcut name to assign to 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 one of two supported engines that import and export an XML document. The syntax documentation is labeled as *XMLV2 only* or *XML only* or *XMLV2 and XML*.

XMLV2

specifies the XMLV2 engine, which accesses the engine functionality of SAS 9.2 and later.

Alias XML92

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

XML

specifies the XML engine, which accesses the engine functionality in SAS 9.1.3 and earlier.

Note For more information about the two engine names, see "Comparing the XMLV2 and XML Engines" on page 76.

'*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*'

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. Here is an example: `'c:\example\data'`.

Engine XMLV2 only

'*XML-document-path*'

includes the pathname, filename, and file extension. Here is an example: `'c:\example\data\myfile.xml'`.

Engine XMLV2 and 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 to assign a fileref to 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 reads from the XML document fred.xml:

```
filename bedrock 'c:\data\fred.xml';
libname bedrock xmlv2;
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= LIBNAME Statement Option" on page 98.

options

are the XMLV2 or XML engine LIBNAME statement options. See "LIBNAME Statement Options" on page 85.

9

LIBNAME Statement Options

Dictionary

AUTOMAP= LIBNAME Statement Option

Specifies to automatically generate an XMLMap file to import an XML document.

Restrictions: Use this option when importing only.

The AUTOMAP= option is not supported in partitioned data sets (PDS) on z/OS.

Requirement: You must include the "XMLMAP= LIBNAME Statement Option" 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 assigned to the physical location in a FILENAME statement. 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= LIBNAME Statement Option". By default, element names are attached to the resulting SAS variable names.

Engine: XMLV2 only

Tips: The functionality to automatically generate an XMLMap file is also available with SAS XML Mapper. The AUTOMAP= option provides the functionality to create and use the XMLMap with a single LIBNAME statement.

Example: "Using the AUTOMAP= Option to Generate an XMLMap" on page 53

Syntax

AUTOMAP=REPLACE | REUSE

Syntax Description

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.

Details

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.

Beginning with SAS 9.4M7, or if you apply a hot fix to SAS 9.4 or to SAS Viya, the behavior of the AUTOMAP= LIBNAME statement option is changed. When you use the AUTOMAP= option, the XMLV2 engine does not validate the XML file. (In general, the XMLV2 engine does not validate the XML file. However, in previous releases, the AUTOMAP= option causes a parser to validate the XML before the XMLMap is generated. This validation no longer occurs.)

Here are additional interactions for XML entities:

- The AUTOMAP= option is not supported for XML files that contain external parameter entities.
- The AUTOMAP= option is not supported for XML files that contain internal general entities that are referenced in the XML content.
- External general entities are not included in the XML content.

To generate an XMLMap for an XML file that contains any of the above entities, use the SAS XML Mapper. You can also write your own XMLMap.

CHARMULTIPLIER= LIBNAME Statement Option

Expands column (variable) lengths by a multiplier value.

Engine: XMLV2 only

Example: "Avoiding Truncation Errors When Importing in a Multi-Byte Encoding" on page 57

Syntax

CHARMULTIPLIER=*value*

Syntax Description

value

specifies the multiplier value to expand column (variable) lengths. These are column (variable) lengths specified in an XMLMap

Use CHARMULTIPLIER= to avoid truncation when importing XML data that contains double-byte character set (DBCS) or multi-byte character set (MBCS) data. An XMLMap that is generated or created by SAS XML Mapper could specify column lengths that do not accommodate the larger character size.

For the multiplier value, specify a real number greater than or equal to 1.0 and less than or equal to 5.0. The engine multiplies the column lengths that are specified in an XMLMap by this number. For information about DBCS and MBCS issues, see *SAS National Language Support (NLS): Reference Guide*.

DERIVECHARMULTIPLIER= LIBNAME Statement Option

Expands column (variable) lengths by a default multiplier value that is based on the session encoding.

Default: NO

Interaction: If CHARMULTIPLIER= is also specified, then DERIVECHARMULTIPLIER= is ignored.

Engine: XMLV2 only

Syntax

DERIVECHARMULTIPLIER=YES | NO

Syntax Description

YES
expands column (variable) lengths by a default multiplier value that is based on the session encoding. These are column (variable) lengths specified in an XMLMap

NO
does not expand column (variable) lengths.

Details

Use DERIVECHARMULTIPLIER=YES to avoid truncation when importing XML data that contains DBCS or MBCS data. An XMLMap that is generated or created by SAS XML Mapper could specify column lengths that do not accommodate the larger character size.

The engine multiplies the column lengths that are specified in an XMLMap by a default multiplier value that is based on the current session encoding. Use DERIVECHARMULTIPLIER=YES rather than CHARMULTIPLIER= when you are uncertain of the session encoding. For information about DBCS and MBCS issues, see *SAS National Language Support (NLS): Reference Guide*.

FORMATACTIVE= LIBNAME Statement Option

Specifies whether SAS formats are used.

Default:	NO
Restriction:	Use this option for the CDISCODM and GENERIC markup types only.
Interaction:	Behavior differs between the CDISCODM and GENERIC markup types. They are documented separately in the syntax description.

Syntax

FORMATACTIVE=NO | YES

Syntax Description

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 converted to SAS formats, and vice versa.

NO
causes formatting controls to be suppressed for both importing and exporting.

YES
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.

Engine XML only

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= LIBNAME Statement Option" on page 89.

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 144

For the GENERIC markup type:

FORMATACTIVE= specifies whether output values are affected by SAS formats.

NO
writes the actual data value to the XML markup.

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

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

Engine XMLV2 and XML

FORMATLIBRARY= LIBNAME Statement Option

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.

Engine: XML only

Syntax

FORMATLIBRARY=*libref*

Syntax Description

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

FORMATNOREPLACE= LIBNAME Statement Option

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 created by the XML engine when it converts a CDISC ODM CodeList element.

Default: NO

Restrictions: Use this option when importing an XML document only.
Use this option for the CDISCODM markup type only.

Engine: XML only

Syntax

FORMATNOREPLACE=NO | YES

Syntax Description

NO
does not replace formats that have the same name.

YES
does replace formats that have the same name.

INDENT= LIBNAME Statement Option

Specifies the number of columns to indent each nested element in the exported XML document.

Default: 3

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

Engine: XMLV2 and XML

Syntax

INDENT=*integer*

Syntax Description

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

The value can be from 0 (which specifies no indention) through 32. This specification is cosmetic and is ignored by an XML-enabled browser.

ODSCHARSET= LIBNAME Statement Option

Specifies the character set to use for the output file

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.

Engine: XMLV2 and XML

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

Syntax

ODSCHARSET=*character-set*

Syntax Description

character-set
specifies the character set to use for the output file.

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

ODSRECSEP= LIBNAME Statement Option

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

Engine: XML only

Syntax

ODSRECSEP=DEFAULT | NONE | YES

Syntax Description

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= LIBNAME Statement Option

Specifies the translation table to use for the output file.

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.

Engine: XMLV2 and XML

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

Syntax

ODSTRANTAB=*table-name*

Syntax Description

table-name

specifies the translation table to use for the output file.

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

PREFIXATTRIBUTES= LIBNAME Statement Option

Specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element, which defines the SAS variable name.

Default: YES

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

Use this option with the "AUTOMAP= LIBNAME Statement Option" on page 85.

Engine: XMLV2 only

Example: "Using the AUTOMAP= Option to Generate an XMLMap" on page 53

Syntax

PREFIXATTRIBUTES=YES | NO

Syntax Description

YES
specifies that the element name is attached to the resulting SAS variable name.

NO
specifies that the element name is not attached to the resulting SAS variable name.

TAGSET= LIBNAME Statement Option

Specifies the name of a tagset to override the default tagset that is used by the specified markup type.

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.

Engine: XMLV2 and XML

See: Chapter 6, “Understanding and Using Tagsets for the XMLV2 Engine,” on page 63

Example: “Exporting an XML Document Using a Customized Tagset” on page 64

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 engine might not be able to translate the XML markup back to SAS proprietary format.**

Syntax

TAGSET=*tagset-name*

Syntax Description

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

Details

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: Procedures Guide*.

XMLCONCATENATE= LIBNAME Statement Option

Specifies whether the file to be imported contains multiple, concatenated XML documents.

Alias: XMLCONCAT=

Default: NO

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

Interaction: If you specify XMLCONCATENATE=YES, then XMLTYPE=GENERIC is required. Other XML markup types are not supported.

Engine: XML only

Tip: 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 150

Syntax

XMLCONCATENATE=NO | YES

Syntax Description

NO
specifies that the file to be imported does not contain concatenated XML documents.

YES
specifies that the file to be imported contains multiple, concatenated XML documents.

Details

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). For the XMLV2 engine, this option is not necessary and is ignored.

XMLDATAFORM= LIBNAME Statement Option

Specifies whether the tag for the element to contain SAS variable information (name and data) is in open-element or enclosed-attribute format.

Default: ELEMENT

Restrictions: Use this option when exporting an XML document only.
Use this option for the GENERIC markup type only.

Engine: XMLV2 and XML

Syntax

XMLDATAFORM=ELEMENT | ATTRIBUTE

Syntax Description

ELEMENT
specifies that the tag is in open-element format.

ATTRIBUTE
specifies that the tag is in enclosed-attribute format.

Details

Here is an example for the variable PRICE. The value of an observation is `1.98`.

- If XMLDATAFORM=ELEMENT, the generated output is `<PRICE> 1.98 </PRICE>`
- If XMLDATAFORM=ATTRIBUTE, the generated output is `<COLUMN name="PRICE"value="1.98" />`

XMLDOUBLE= LIBNAME Statement Option

Controls the results of importing or exporting numeric values.

Default: DISPLAY

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

Examples: “Exporting Numeric Values” on page 137
“Importing an XML Document with Numeric Values” on page 144

Syntax

XMLDOUBLE=DISPLAY | INTERNAL

Syntax Description

DISPLAY
when exporting, the engine retrieves the stored value for the numeric variable, determines an appropriate display for the value in a readable form, and writes

the display value to the XML document. The display value is affected by the engine and whether a format is assigned.

- The XML engine 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 ignores any assigned format and displays the value using BEST16.

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

Alias FORMAT

Engine XMLV2 and XML

INTERNAL

when exporting, the 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 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.

Alias PRECISION

Engine XMLV2 for export only; XML for import and export

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

XMLENCODING= LIBNAME Statement Option

Overrides the SAS data set's encoding for the output file.

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.

Engine: XMLV2 and XML

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.

See: XMLENCODING= option in *SAS National Language Support (NLS): Reference Guide*.

Syntax

XMLENCODING='*encoding-value*'

Syntax Description

'*encoding-value*'
specifies an encoding value. If the value contains a hyphen, enclose the value in quotation marks.

Details

The XMLENCODING= option overrides the default encoding for export. For the default behavior during export and import, see "Transferring an XML Document across Environments" on page 6.

XMLFILEREF= LIBNAME Statement Option

Specifies an XML document to be exported or imported.

Engine: XMLV2 and XML

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 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 46.

Syntax

XMLFILEREF=*fileref*

Syntax Description

fileref
is the SAS name that is assigned to the physical location of the XML document to be exported or imported.

Details

To assign the fileref, use the FILENAME statement. The engine can access any data referenced by a fileref. For example, the following code writes to the XML document `wilma.xml`:

```
libname source 'c:\example';
filename cartoon 'c:\data\wilma.xml';
libname bedrock xmlv2 xmlfileref=cartoon;
data bedrock.wilma;
   set source.wilma;
run;
```

XMLMAP= LIBNAME Statement Option

Specifies an XML document that contains specific XMLMap syntax.

Restriction: See the XMLMap syntax on page 112 for the version numbers that are supported for import and export by the XMLV2 engine, and for import only by the XML engine.

Interactions: If you specify an XMLMap, you must either 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.

If you specify the AUTOMAP= option, you must specify the XMLMAP= option.

Engine: XMLV2 and XML

See: Chapter 10, “Introduction to XMLMap Syntax,” on page 107

Examples: Chapter 4, “Importing XML Documents Using an XMLMap,” on page 21

Chapter 5, “Exporting XML Documents Using an XMLMap,” on page 59

Syntax

XMLMAP=*fileref* | '*XMLMap-location*'

Syntax Description

fileref

is the SAS name that is assigned to 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-location*'

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

Details

The syntax tells the engine how to interpret the XML markup for importing or exporting. When you import with the AUTOMAP=YES option, specify the fileref or location of the XMLMap to be generated. When you export, or when you import without the AUTOMAP=YES option, specify the fileref or location of the existing XMLMap.

XMLMETA= LIBNAME Statement Option

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.

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.

Engine: XMLV2 and XML

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 146

Syntax

XMLMETA=DATA | SCHEMADATA | SCHEMA

Syntax Description

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.

Details

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.

XMLPROCESS= LIBNAME Statement Option

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

Default: CONFORM

Engine: XMLV2 and XML

Example: "Importing an XML Document with Non-Escaped Character Data" on page 11

Syntax

XMLPROCESS=CONFORM | PERMIT

Syntax Description

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 `&`. 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 `'` and the double quotation mark character (") can be represented as `"`.

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.

XMLSCHEMA= LIBNAME Statement Option

Specifies an external file to contain metadata-related information.

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.

Engine: XMLV2 and XML

Example: "Exporting an XML Document with Separate Metadata" on page 17

Syntax

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

Syntax Description

fileref
is the SAS name that is assigned to 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.

XMLTYPE= LIBNAME Statement Option

Specifies the XML markup type.

Default: GENERIC

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

Syntax

XMLTYPE=GENERIC | CDISCODM | MSACCESS | ORACLE | XMLMAP

Syntax Description

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.

Engine XMLV2 and XML

See "Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type" on page 22

Examples "Exporting an XML Document That Contains SAS Dates, Times, and Datetimes" on page 15

"Exporting Numeric Values" on page 137

"Importing an XML Document Using the GENERIC Markup Type" on page 9

CDISCODM

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.

Engine XML only

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 142

"Exporting an XML Document in CDISC ODM Markup" on page 144

MSACCESS

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.

Engine XML only

Example "Importing an XML Document Created by Microsoft Access" on page 146

ORACLE

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.

Engine XML only

Example "Exporting an XML Document for Use by Oracle" on page 135

XMLMAP

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 engine how to map the SAS data into a specific XML document structure. To specify the XMLMap in the LIBNAME statement, use the "PREFIXATTRIBUTES= LIBNAME Statement Option" on page 93.

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

Engine XMLV2 only

Example "Using an XMLMap to Export an XML Document with a Hierarchical Structure" on page 59

PART 3

XMLMap File Syntax

10

Introduction to XMLMap Syntax

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 111.
- 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 113.
- If you use an XMLMap for exporting, you must include the exporting elements. See "Elements for Exporting" on page 114.
- The table elements define the SAS data set. See "Elements for Tables" on page 115.
- The column elements define the variables for the SAS data set. See "Elements for Columns" on page 119.
- The syntax documentation is labeled with a superscript such as XMLV2 only or XML only or XMLV2 and XML to indicate which engines are supported. See also "Comparing the XMLMap Syntax" on page 108.

CAUTION

The XMLMap markup, as XML itself, is case sensitive. The tag names must be uppercase, and the element attributes must be lowercase. Here is an 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 each engine.

Table 10.1 *XMLMap Syntax*

Syntax	Description	Import	Export	XML	XMLV2
SXLEMAP on page 111	Primary (root) enclosing element	■	■	■	■
NAMESPACES on page 113	Contains elements for defining XML namespaces	■	■		■
NS on page 113	Defines an XML namespace by referencing a unique URI	■	■		■
OUTPUT on page 114	Contains elements for exporting a SAS data set as an XML document		■		■
HEADING on page 114	Contains ATTRIBUTE elements for exporting		■		■
ATTRIBUTE on page 115	Contains file attribute information for exporting		■		■
TABLEREF on page 115	Specifies the name of the table for exporting		■		■
TABLE on page 115	Contains a data set definition	■	■	■	■
TABLE-PATH on page 116	Specifies a location path for variables	■	■	■	■
TABLE-END-PATH on page 117	Specifies a location path to stop processing	■	■	■	■
TABLE-DESCRIPTION on page 119	Specifies a SAS data set description	■	■	■	■
COLUMN name= on page 119	Specifies the variable name	■	■	■	■

Syntax	Description	Import	Export	XML	XMLV2
COLUMN retain= on page 120	Determines the contents of the input buffer	■	■	■	■
COLUMN class= on page 120	Determines the type of variable	■	■		■
TYPE on page 120	Specifies the SAS data type for the variable	■	■	■	■
DATATYPE on page 121	Specifies the type of data being read	■	■	■	■
DEFAULT on page 122	Specifies a default value for a missing value	■	■	■	■
ENUM on page 122	Contains a list of valid values for the variable	■	■	■	■
FORMAT on page 122	Specifies a SAS format for the variable	■	■	■	■
INFORMAT on page 123	Specifies a SAS informat for the variable	■	■	■	■
DESCRIPTION on page 123	Specifies a description for the variable	■	■	■	■
LENGTH on page 123	Determines the maximum field storage length for a character variable	■	■	■	■
PATH on page 123	Specifies a location path for the current variable	■	■	■	■
INCREMENT-PATH on page 126	Specifies a location path for incrementing the accumulated value for a counter variable	■	■	■	■
RESET-PATH on page 126	Specifies a location path for resetting the accumulated value for a counter variable to zero	■	■	■	■
DECREMENT-PATH on page 127	Specifies a location path to decrement the accumulated value for the counter variable by 1	■	■	■	■

11

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 For import, the XMLV2 engine supports XMLMap syntax versions 1.2, 1.9, and 2.1. For export, the XMLV2 engine supports version 2.1, and the XML92 engine (alias for XMLV2) supports version 1.9.

For import, the XML engine supports XMLMap syntax versions 1.0, 1.1, and 1.2. The XML engine does not support using an XMLMap for export.

Tip To update an XMLMap to version 2.1, load the existing XMLMap into SAS 9.3 or later XML Mapper, and then save the XMLMap. For information about SAS XML Mapper, see Chapter 12, “Using SAS XML Mapper to Generate and Update an XMLMap,” on page 129.

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 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

NAMESPACES count="*number*"
NS id="*number*" <prefix="*name*">

Elements

NAMESPACES count="*number*" XMLV2 only

is an optional element that contains one or more NS elements for defining XML namespaces. Here is an example: `<NAMESPACES 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.

Requirement 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 50

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. Here is an 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.

In the variable definition, the identification number must be included in the location path preceding the element that is being defined. See "PATH syntax="*type*" XMLV2 and XML" on page 123.

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 50

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

OUTPUT
HEADING
ATTRIBUTE name="*name*" value="*value*"
TABLEREF name="*name*"

Elements

OUTPUT XMLV2 only
is an optional element that contains zero 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 59

HEADING XMLV2 only
is an optional element that contains zero or more ATTRIBUTE elements.

ATTRIBUTE name="*attribute-name*" value="*attribute-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. Here is an example: `<NHL description="Teams of the National Hockey League">`.

name="*attribute-name*"
specifies a name for a file attribute. In the example code above, it is `name="description"`.

value="*attribute-value*"
specifies a value for the attribute. In the example code above, it is `value="Teams of the National Hockey League"`.

TABLEREF name="*table-name*" XMLV2 only
specifies the name of the table in the XMLMap to be exported.

name="*table-name*"
is a valid SAS name that is unique in the XMLMap definition.

Requirements If you specify the OUTPUT element, you must specify one (and no more than one) TABLEREF element.

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

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*" XMLV2 and XML
is an element that contains a data set definition. Here is an 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*" XMLV2 and XML
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, in "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 29, consider this TABLE-PATH element in `rss.map` for the `item` data set:

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

The TABLE-PATH causes the following to occur

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. Here is an 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. Here is an example: `syntax="XPathENR"`.

CAUTION

Specifying the table location path, which is the observation boundary, can be complicated 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, errors could result.

Requirements The TABLE-PATH element 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. Here is an 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" XMLV2 and XML
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, in "Using an XMLMap to Import an XML Document as Multiple SAS Data Sets" on page 29, here is the TABLE-PATH location path in `rss.map` for the `channel` data set:

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

The XML document `rss.xml` has only one <CHANNEL> start tag and one </CHANNEL> end tag, so processing could conclude after the </CHANNEL> end tag. The engine does not store new data in the input buffer after it encounters the first <ITEM> start tag because the remaining elements no longer qualify. However, the engine continues to process the entire XML document. To improve efficiency for a long XML document, you could add a TABLE-END-PATH element. For example, if you add the following specification for the `channel` data set, the XML engine stops processing when the first <ITEM> start tag is encountered:

```
<TABLE-END-PATH syntax="XPath" beginend="BEGIN"> /rss/channel/item </TABLE-END-PATH>
```

With both the TABLE-PATH and TABLE-END-PATH specifications, the XML engine processes only the highlighted data in the `rss.xml` document for the `channel` data set rather than the entire XML document.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>My RSS Channel</title>
  <description>This is a simplified example of an RSS feed.</description>
  <link>http://www.example.com/main.html</link>
```

```
  <language>en-us</language>
  <item>
   <title>My news item</title>
   <description>This is a detailed summary of my news item.</description>
   <link>http://www.example.com/blog/post/1</link>
  </item>
  <item>
   <title>Another news item</title>
   <description>This description is shorter.</description>
   <link>http://www.example.com/blog/post/2</link>
  </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. Here is an 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. Here is an 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

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

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. Here is an 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 29

TABLE-DESCRIPTION XMLV2 and XML
is an optional element that specifies a description for the SAS data set, which can be up to 256 characters. Here is an 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"
DECREMENT-PATH syntax="*type*" beginend="BEGIN | END"

Elements

COLUMN name="*name*" retain="NO | YES" class= "ORDINAL"
is an element that contains a variable definition. Here is an example: `<COLUMN name="Title">`.

name="*name*" XMLV2 and XML
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" XMLV2 and XML
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 32

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 36

Requirement At least one COLUMN element is required.

Interaction COLUMN can contain one or more of the following elements that describe the variable attributes: DATATYPE, DEFAULT, ENUM, FORMAT, INFORMAT, DESCRIPTION, LENGTH, TYPE, PATH, INCREMENT-PATH, DECREMENT-PATH, and RESET-PATH.

TYPE XMLV2 and XML
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

```
<DATATYPE> float </DATATYPE>
<TYPE> numeric </TYPE>
```

.

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

To control data type conversion in input, use the INFORMAT element. Here is an example: `<INFORMAT> datetime </INFORMAT>`.

DATATYPE XMLV2 and XML

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 123.

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 123.

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 123.

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 XMLV2 and XML

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.

ENUM XMLV2 and XML

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.

```
<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>
```

FORMAT width="*w*" ndec="*d*" XMLV2 and XML

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:

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

INFORMAT width="*w*" ndec="*d*" XMLV2 and XML

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:

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

width="*w*"

is an optional attribute that specifies an informat width, which for most informats is the number of columns in the input data.

ndec="*d*"

is an optional attribute that specifies a decimal scaling factor for numeric informats. SAS divides the input data by 10 to the power of this value.

DESCRIPTION XMLV2 and XML

is an optional element that specifies a description for the variable, which can be up to 256 characters. The following example shows that the description is assigned as the variable label.

```
<DESCRIPTION> Story link </DESCRIPTION>
```

LENGTH XMLV2 and XML

is the maximum field storage length from the XML data for a character variable. The value refers to the number of characters 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. Here is an 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.

Tips 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.

When importing a table with multi-byte character set (MBCS) data, you might need to expand the number of bytes specified in the LENGTH element. Use the CHARMULTIPLIER= or DERIVECHARMULTIPLIER= LIBNAME statement options to ensure that the SAS data set column accommodates the larger character size.

PATH syntax="*type*" XMLV2 and XML

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. Here is an 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 48.

- The following location path tells the XML engine to scan the XML markup until it finds the CONFERENCE element. The XML engine retrieves the value between the <CONFERENCE> start tag and the </CONFERENCE> end tag.

```
<PATH> /NHL/CONFERENCE </PATH>
```

- The following location path tells the XML engine to scan the XML markup until it finds the TEAM element where the value of the founded= attribute is 1993. The XML engine retrieves the value between the <TEAM> start tag and the </TEAM> end tag.

```
<PATH> /NHL/CONFERENCE/DIVISION/TEAM[@founded="1993"] </PATH>
```

- The following location path uses the position function to tell the XML engine to scan the XML markup until it finds the fifth occurrence of the TEAM element. The XML engine retrieves the value between the <TEAM> start tag and the </TEAM> end tag.

```
<PATH> /NHL/CONFERENCE/DIVISION/TEAM[position()=5] </PATH>
```

You can use the following shorter version for the position function:

```
<PATH> /NHL/CONFERENCE/DIVISION/TEAM[5] </PATH>
```

attribute-form

selects values from an attribute. The following attribute forms enable you to select from a specific attribute or conditionally select from a specific attribute based on the value of another attribute:

```
<PATH> /LEVEL/ITEM/@attr </PATH>
<PATH> /LEVEL/ITEM/@attr[@attr2="value"] </PATH>
```

The following examples illustrate the attribute forms. For more information about the examples, see "Specifying a Location Path on the PATH Element" on page 48.

- 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. Here is an example: `<PATH syntax="XPathENR">/Table/Hurricane/{1}Month</PATH>`. See "Including Namespace Elements in an XMLMap" on page 50.

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 48

INCREMENT-PATH syntax="*type*" beginend="BEGIN | END" XMLV2 and XML

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. Here is an 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. Here is an 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. Here is an 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 36

RESET-PATH syntax="*type*" beginend="BEGIN | END" XMLV2 and XML

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. Here is an 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. Here is an 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 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. Here is an example: `<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 syntax="*type*" beginend=""BEGIN | END" XMLV2 and XML
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.

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. Here is an 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. Here is an 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 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. Here is an 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.

12

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 XMLV2 and XML engines. Based on Java, SAS XML Mapper is a stand-alone application that removes the tedium of creating and modifying an XMLMap.

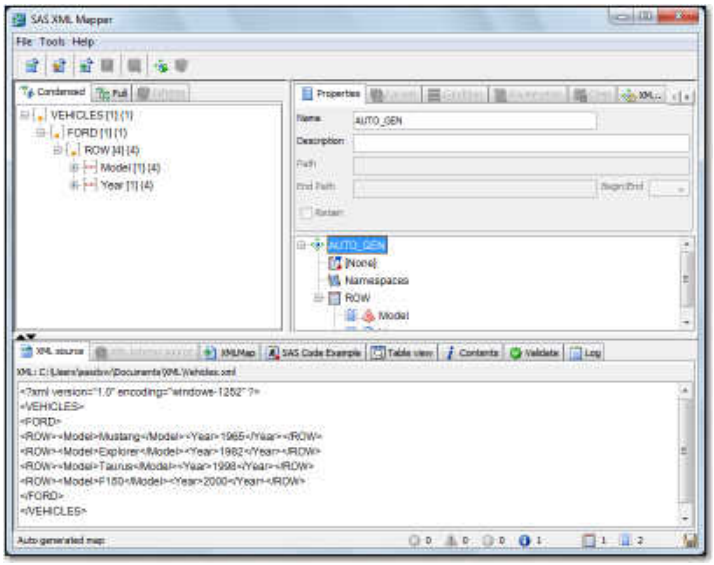
SAS XML Mapper provides a graphical interface that you can use to generate the appropriate XML elements. SAS XML Mapper analyzes the structure of an XML document or an XML schema and generates basic XML syntax for the XMLMap.

See these helpful videos:

- How to Automatically Generate XMLMap Files
- How to Generate Custom XMLMap Files

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.

Figure 12.1 SAS XML Mapper Application



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 at http://support.sas.com/demosdownloads/setupcat.jsp?cat=Base+SAS+Software. For SAS Viya releases, SAS XML Mapper is not included in the installation media. You can download it from the SAS website to install it.

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**.

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

Usage Not Supported for the XMLV2 Engine

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. XMLTYPE=ORACLE is not supported by the XMLV2 engine

The following output shows the SAS data set `sashelp.class` to be exported to Oracle.

Output A4.1 *SAS Data Set sashelp.class*

The SAS System

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

The following SAS program exports an XML document from the SAS data set `sashelp.class`:

```
libname trans xml 'path-name/output/myxmlfile.xml'
xmltype=oracle; /*1*/

data trans.class; /*2*/
   set myfiles.class;
run;
```

1 The second LIBNAME statement assigns the libref `trans` to the physical location of the file (complete path name, file name, 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.

2 The DATA step reads the SAS data set `sashelp.class` and writes its content in Oracle XML markup to the specified XML document, `myxmlfile.xml`.

Here is the resulting XML document.

Output A4.2 *XML Document myxmlfile.xml to Be Used by Oracle*

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

Exporting 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.

This example is appropriate for the XML engine only. The XMLV2 engine ignores any assigned format and displays the value using BEST16.

```
libname format xml 'c:\output\format.xml'; /* 1 */

libname prec xml 'c:\output\precision.xml' xmldouble=internal; /* 2 */

data npi; /* 3 */
   do n=1 to 10;
      n_pi = n*3.141592653589793;
      output;
   end;
format n_pi 14.2;
run;

data format.dbltest; /* 4 */
```

```
   set npi;
run;

data prec.rawtest; /* 5 */
   set npi;
run;

title 'Drops the Precision'; /* 6 */
proc print data=format.dbltest;
   format n_pi 14.10;
run;

title 'Keeps the Precision'; /* 7 */
proc print data=prec.rawtest;
   format n_pi 14.10;
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=INTERNAL option 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 generates the XML document `format.xml`, which contains numeric values controlled by the SAS format. See Output A1.3 on page 139.
5. The DATA step generates the XML document `precision.xml`, which contains the stored numeric values. See Output A1.5 on page 141.
6. For the PRINT procedure output, a format is specified to show the precision loss. In the output, the decimals after the second digit are zeros. See Output A1.4 on page 140.
7. For the PRINT procedure output, a format is specified to show the retained precision. See Output A1.6 on page 142.

Output A4.3 *XML Document format.xml*

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

Output A4.4 *PRINT Procedure Output for format.dbltest*

Drops the Precision

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

Output A4.5 *XML Document precision.xml*


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

Output A4.6 *PRINT Procedure Output for prec.rawtest*

Keeps the Precision

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

Importing a CDISC ODM Document

This example imports the XML document that is shown in Appendix 2, “Example CDISC ODM Document,” on page 153. 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. Note that XMLTYPE=CDISCODM is not supported by the XMLV2 engine.

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

```
filename odm 'c:\example\ae.xml'; /* 1 */
libname odm xml xmltype=cdiscodm /* 2 */
   formatactive=yes /* 3 */
   formatnoreplace=no /* 4 */
   formatlibrary="work"; /* 5 */
proc contents data=odm.ae varnum; /* 6 */
run;

data ae;  /* 7 */
   set odm.ae;
run;
```

1 The FILENAME statement assigns the fileref `odm` to the physical location of the XML document (complete path name, file name, 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 FORMATLIBRARY="work" specifies to create the format catalog in the temporary Work library. The Work library is also the default if you omit the FORMATLIBRARY= 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.

7 The DATA step imports the document as a SAS data set `ae` in the Work library.

Output A4.7 *CONTENTS Procedure Output for odm.ae*

The SAS System

The CONTENTS Procedure

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

Variables in Creation Order

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

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 142 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. XMLTYPE=CDISCODM is not supported by the XMLV2 engine.

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

```
filename output 'c:\myoutput.xml'; /* 1 */
libname output xml xmltype=cdiscodm formatactive=yes; /* 2 */

data output.ae;  /* 3 */
   set work.ae;
run;
```

1. The FILENAME statement assigns the fileref `output` to the physical location of the external file (complete path name, file name, 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.
3. The DATA step creates an XML document, `myoutput.xml`.

The output is the same as the XML document that is shown in Appendix 2, "Example CDISC ODM Document," on page 153.

Importing an XML Document with Numeric Values

This example imports the XML document Precision.XML, which was exported in "Exporting Numeric Values" on page 137. 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:

```
libname default xml 'c:\output\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 A4.8 *PRINT Procedure Output for default.rawtest*

Default Method

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. (This example uses the XML engine. When importing, XMLDOUBLE=INTERNAL is not supported by the XMLV2 engine.)

```
libname new xml 'c:\output\precision.xml' xmldouble=internal;

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

The result of the import is SAS data set `new.rawtest`.

Output A4.9 *PRINT Procedure Output for new.rawtest*

Precision Method

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

Importing an XML Document 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. XMLTYPE=MSACCESS is not supported by the XMLV2 engine.

```
<?xml version="1.0" encoding="windows-1252" ?>
<root      xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:od="urn:schemas-microsoft-com:officedata">
   <xs:schema>
      <xs:element name="dataroot">
         <xs:complexType>
            <xs:sequence>
               <xs:element ref="SUPPLIERS"  minOccurs="0"
maxOccurs="unbounded" />
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="SUPPLIERS">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="HOMEPAGE" minOccurs="0"
                           od:jetType="text" od:sqlSType="nvarchar">
```

```
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="94" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="FAX" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="15" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="PHONE" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="15" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="COUNTRY" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="11" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="POSTALCODE" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="8" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="REGION" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="8" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="CITY" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
   <xs:simpleType>
      <xs:restriction base="xs:string">
         <xs:maxLength value="13" />
      </xs:restriction>
   </xs:simpleType>
</xs:element>
<xs:element name="ADDRESS" minOccurs="0"
            od:jetType="text" od:sqlSType="nvarchar">
```

```
                        <xs:simpleType>
                           <xs:restriction base="xs:string">
                              <xs:maxLength value="45" />
                           </xs:restriction>
                        </xs:simpleType>
                     </xs:element>
                     <xs:element name="CONTACTTITLE" minOccurs="0"
                                 od:jetType="text" od:sqlSType="nvarchar">
                        <xs:simpleType>
                           <xs:restriction base="xs:string">
                              <xs:maxLength value="28" />
                           </xs:restriction>
                        </xs:simpleType>
                     </xs:element>
                     <xs:element name="CONTACTNAME" minOccurs="0"
                                 od:jetType="text" od:sqlSType="nvarchar">
                        <xs:simpleType>
                           <xs:restriction base="xs:string">
                              <xs:maxLength value="26" />
                           </xs:restriction>
                        </xs:simpleType>
                     </xs:element>
                     <xs:element name="COMPANYNAME" minOccurs="0"
                                 od:jetType="text" od:sqlSType="nvarchar">
                        <xs:simpleType>
                           <xs:restriction base="xs:string">
                              <xs:maxLength value="38" />
                           </xs:restriction>
                        </xs:simpleType>
                     </xs:element>
                     <xs:element name="SUPPLIERID" minOccurs="0"
                                 od:jetType="double" od:sqlSType="double"
type="xs:double" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:schema>
         <dataroot>
            <SUPPLIERS>
               <HOMEPAGE/>
               <FAX/>
               <PHONE>(272) 444-2222</PHONE>
               <COUNTRY>UK</COUNTRY>
               <POSTALCODE>EC1 4SD</POSTALCODE>
               <REGION/>
               <CITY>London</CITY>
               <ADDRESS>49 Franklin St.</ADDRESS>
               <CONTACTTITLE>Purchasing Manager</CONTACTTITLE>
               <CONTACTNAME>Charlotte Smith</CONTACTNAME>
               <COMPANYNAME>Exotic Flowers</COMPANYNAME>
               <SUPPLIERID>1</SUPPLIERID>
            </SUPPLIERS>
            <SUPPLIERS>
               <HOMEPAGE>#MYCAJUN.HTM#</HOMEPAGE>
               <FAX/>
               <PHONE>(512) 284-3677</PHONE>
```

```
            <COUNTRY>USA</COUNTRY>
            <POSTALCODE>70117</POSTALCODE>
            <REGION>LA</REGION>
            <CITY>New Orleans</CITY>
            <ADDRESS>P.O. Box 78934</ADDRESS>
            <CONTACTTITLE>Order Administrator</CONTACTTITLE>
            <CONTACTNAME>Shelley Martin</CONTACTNAME>
            <COMPANYNAME>New Orleans Cajun Foods</COMPANYNAME>
            <SUPPLIERID>2</SUPPLIERID>
         </SUPPLIERS>
      .
      .
      .
      </dataroot>
   </root>
```

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

```
libname access xml '/u/myid/XML/suppliers.xml'
   xmltype=msaccess xmlmeta=schemadata; /* 1 */

proc print data=access.suppliers (obs=2); /* 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 A4.10 *PRINT Procedure Output for access.suppliers*

The SAS System

Obs	CONTACTNAME	COMPANYNAME
1	Charlotte Smith	Exotic Flowers
2	Shelley Martin	New Orleans Cajun Foods

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;
```

Output A4.11 *CONTENTS Procedure Output for access.suppliers*

The SAS System

The CONTENTS Procedure

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

Alphabetic List of Variables and Attributes

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

Importing Concatenated XML Documents

For a file that is a concatenation of multiple XML documents, you can use the XMLV2 engine without the XMLCONCATENATE= option, as long as the elements pass XML markup parsing rules such as case sensitivity.

To use the older XML engine to import the file, simply specify the LIBNAME statement option XMLCONCATENATE=YES.

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

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

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

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

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

```
libname concat xml 'c:\example\concatstudents.xml';
proc datasets library=concat;
run;
quit;
```

Example Code A4.1 *SAS Log Output*

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

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 'c:\example\concatstudents.xml' xmlconcatenate=yes;

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

Output A4.12 *PRINT Procedure Output for concat.students*

The SAS System

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

Appendix 2

Example CDISC ODM Document

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 142 and in “Exporting an XML Document in CDISC ODM Markup” on page 144.

```
<?xml version="1.0" encoding="windows-1252" ?>
- <!--       Clinical Data Interchange Standards Consortium (CDISC)
      Operational Data Model (ODM) for clinical data interchange

      You can learn more about CDISC standards efforts at
      http://www.cdisc.org/standards/index.html

  -->
- <ODM xmlns="http://www.cdisc.org/ns/odm/v1.2"
   xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.cdisc.org/ns/odm/v1.2 ODM1-2-0.xsd"
   ODMVersion="1.2"
   FileOID="000-00-0000"
   FileType="Snapshot"
   Description="Adverse events from the CTChicago file"
   AsOfDateTime="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.AESEV" OrderNumber="15" Mandatory="No" />
  <ItemRef ItemOID="ID.AEREL" OrderNumber="16" Mandatory="No" />
  <ItemRef ItemOID="ID.AEOUT" OrderNumber="17" Mandatory="No" />
  <ItemRef ItemOID="ID.AEACTTRT" OrderNumber="18" Mandatory="No" />
  <ItemRef ItemOID="ID.AECONTRT" OrderNumber="19" Mandatory="No" />
  </ItemGroupDef>
```

```
- <!--                Column attributes as defined in the table

  -->
- <ItemDef OID="ID.TAREA" SASFieldName="TAREA" Name="Therapeutic Area" DataType="text" Length="4">
  <CodeListRef CodeListOID="CL.$TAREAF" />
  </ItemDef>
  <ItemDef OID="ID.PNO" SASFieldName="PNO" Name="Protocol Number" DataType="text" Length="15" />
- <ItemDef OID="ID.SCTRY" SASFieldName="SCTRY" Name="Country" DataType="text" Length="4">
  <CodeListRef CodeListOID="CL.$SCTRYF" />
  </ItemDef>
- <ItemDef OID="ID.F_STATUS" SASFieldName="F_STATUS" Name="Record status, 5 levels, internal use"
DataType="text" Length="1">
  <CodeListRef CodeListOID="CL.$F_STATU" />
  </ItemDef>
  <ItemDef OID="ID.LINE_NO" SASFieldName="LINE_NO" Name="Line Number" DataType="integer"
Length="2" />
  <ItemDef OID="ID.AETERM" SASFieldName="AETERM" Name="Conmed Indication" DataType="text"
Length="100" />
  <ItemDef OID="ID.AESTMON" SASFieldName="AESTMON" Name="Start Month - Enter Two Digits 01-12"
DataType="integer" Length="2" />
  <ItemDef OID="ID.AESTDAY" SASFieldName="AESTDAY" Name="Start Day - Enter Two Digits 01-31"
DataType="integer" Length="2" />
  <ItemDef OID="ID.AESTYR" SASFieldName="AESTYR" Name="Start Year - Enter Four Digit Year"
DataType="integer" Length="4" />
  <ItemDef OID="ID.AESTDT" SASFieldName="AESTDT" Name="Derived Start Date" DataType="date" />
  <ItemDef OID="ID.AEENMON" SASFieldName="AEENMON" Name="Stop Month - Enter Two Digits 01-12"
DataType="integer" Length="2" />
  <ItemDef OID="ID.AEENDAY" SASFieldName="AEENDAY" Name="Stop Day - Enter Two Digits 01-31"
DataType="integer" Length="2" />
  <ItemDef OID="ID.AEENYR" SASFieldName="AEENYR" Name="Stop Year - Enter Four Digit Year"
DataType="integer" Length="4" />
  <ItemDef OID="ID.AEENDT" SASFieldName="AEENDT" Name="Derived Stop Date" DataType="date" />
- <ItemDef OID="ID.AESEV" SASFieldName="AESEV" Name="Severity" DataType="text" Length="1">
  <CodeListRef CodeListOID="CL.$AESEV" />
  </ItemDef>
- <ItemDef OID="ID.AEREL" SASFieldName="AEREL" Name="Relationship to study drug" DataType="text"
Length="1">
  <CodeListRef CodeListOID="CL.$AEREL" />
  </ItemDef>
- <ItemDef OID="ID.AEOUT" SASFieldName="AEOUT" Name="Outcome" DataType="text" Length="1">
  <CodeListRef CodeListOID="CL.$AEOUT" />
  </ItemDef>
- <ItemDef OID="ID.AEACTTRT" SASFieldName="AEACTTRT" Name="Actions taken re study drug"
DataType="text" Length="1">
  <CodeListRef CodeListOID="CL.$AEACTTR" />
  </ItemDef>
- <ItemDef OID="ID.AECONTRT" SASFieldName="AECONTRT" Name="Actions taken, other" DataType="text"
Length="1">
  <CodeListRef CodeListOID="CL.$AECONTR" />
  </ItemDef>
- <!--                Translation to ODM markup for any PROC FORMAT style
               user defined or SAS internal formatting specifications
               applied to columns in the table

  -->
- <CodeList OID="CL.$TAREAF" SASFormatName="$TAREAF" Name="$TAREAF" DataType="text">
- <CodeListItem CodedValue="ONC">
```

```
- <Decode>
  <TranslatedText xml:lang="en">Oncology</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
- <CodeList OID="CL.$SCTRYF" SASFormatName="$SCTRYF" Name="$SCTRYF" DataType="text">
- <CodeListItem CodedValue="USA">
- <Decode>
  <TranslatedText xml:lang="en">United States</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
- <CodeList OID="CL.$F_STATU" SASFormatName="$F_STATU" Name="$F_STATU" DataType="text">
- <CodeListItem CodedValue="S">
- <Decode>
  <TranslatedText xml:lang="en">Source verified, not queried</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="V">
- <Decode>
  <TranslatedText xml:lang="en">Source verified, queried</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
- <CodeList OID="CL.$AESEV" SASFormatName="$AESEV" Name="$AESEV" DataType="text">
- <CodeListItem CodedValue="1">
- <Decode>
  <TranslatedText xml:lang="en">Mild</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="2">
- <Decode>
  <TranslatedText xml:lang="en">Moderate</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="3">
- <Decode>
  <TranslatedText xml:lang="en">Severe</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="4">
- <Decode>
  <TranslatedText xml:lang="en">Life Threatening</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
- <CodeList OID="CL.$AEREL" SASFormatName="$AEREL" Name="$AEREL" DataType="text">
- <CodeListItem CodedValue="0">
- <Decode>
  <TranslatedText xml:lang="en">None</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="1">
- <Decode>
  <TranslatedText xml:lang="en">Unlikely</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="2">
- <Decode>
  <TranslatedText xml:lang="en">Possible</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="3">
- <Decode>
  <TranslatedText xml:lang="en">Probable</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
```

```
- <CodeList OID="CL.$AEOUT" SASFormatName="$AEOUT" Name="$AEOUT" DataType="text">
- <CodeListItem CodedValue="1">
- <Decode>
  <TranslatedText xml:lang="en">Resolved, no residual effects</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="2">
- <Decode>
  <TranslatedText xml:lang="en">Continuing</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="3">
- <Decode>
  <TranslatedText xml:lang="en">Resolved, residual effects</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="4">
- <Decode>
  <TranslatedText xml:lang="en">Death</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
- <CodeList OID="CL.$AEACTTR" SASFormatName="$AEACTTR" Name="$AEACTTR" DataType="text">
- <CodeListItem CodedValue="0">
- <Decode>
  <TranslatedText xml:lang="en">None</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="1">
- <Decode>
  <TranslatedText xml:lang="en">Discontinued permanently</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="2">
- <Decode>
  <TranslatedText xml:lang="en">Reduced</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="3">
- <Decode>
  <TranslatedText xml:lang="en">Interrupted</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
- <CodeList OID="CL.$AECONTR" SASFormatName="$AECONTR" Name="$AECONTR" DataType="text">
- <CodeListItem CodedValue="0">
- <Decode>
  <TranslatedText xml:lang="en">None</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="1">
- <Decode>
  <TranslatedText xml:lang="en">Medication required</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="2">
- <Decode>
  <TranslatedText xml:lang="en">Hospitalization required or prolonged</TranslatedText>
  </Decode>
  </CodeListItem>
- <CodeListItem CodedValue="3">
- <Decode>
  <TranslatedText xml:lang="en">Other</TranslatedText>
  </Decode>
  </CodeListItem>
  </CodeList>
  </MetaDataVersion>
  </Study>
```

```
- <!--          Administrative metadata

  -->
  <AdminData />
- <!--          Clinical Data   : AE
                           Adverse Events
                           Some adverse events from this trial

  -->
- <ClinicalData StudyOID="STUDY.StudyOID" MetaDataVersionOID="v1.1.0">
- <SubjectData SubjectKey="001">
- <StudyEventData StudyEventOID="SE.VISIT1" StudyEventRepeatKey="1">
- <FormData FormOID="FORM.AE" FormRepeatKey="1">
- <ItemGroupData ItemGroupOID="IG.AE" ItemGroupRepeatKey="1">
  <ItemData ItemOID="ID.TAREA" Value="ONC" />
  <ItemData ItemOID="ID.PNO" Value="143-02" />
  <ItemData ItemOID="ID.SCTRY" Value="USA" />
  <ItemData ItemOID="ID.F_STATUS" Value="V" />
  <ItemData ItemOID="ID.LINE_NO" Value="1" />
  <ItemData ItemOID="ID.AETERM" Value="HEADACHE" />
  <ItemData ItemOID="ID.AESTMON" Value="06" />
  <ItemData ItemOID="ID.AESTDAY" Value="10" />
  <ItemData ItemOID="ID.AESTYR" Value="1999" />
  <ItemData ItemOID="ID.AESTDT" Value="1999-06-10" />
  <ItemData ItemOID="ID.AEENMON" Value="06" />
  <ItemData ItemOID="ID.AEENDAY" Value="14" />
  <ItemData ItemOID="ID.AEENYR" Value="1999" />
  <ItemData ItemOID="ID.AEENDT" Value="1999-06-14" />
  <ItemData ItemOID="ID.AESEV" Value="1" />
  <ItemData ItemOID="ID.AEREL" Value="0" />
  <ItemData ItemOID="ID.AEOUT" Value="1" />
  <ItemData ItemOID="ID.AEACTTRT" Value="0" />
  <ItemData ItemOID="ID.AECONTRT" Value="1" />
  </ItemGroupData>
- <ItemGroupData ItemGroupOID="IG.AE" ItemGroupRepeatKey="2">
  <ItemData ItemOID="ID.TAREA" Value="ONC" />
  <ItemData ItemOID="ID.PNO" Value="143-02" />
  <ItemData ItemOID="ID.SCTRY" Value="USA" />
  <ItemData ItemOID="ID.F_STATUS" Value="V" />
  <ItemData ItemOID="ID.LINE_NO" Value="2" />
  <ItemData ItemOID="ID.AETERM" Value="CONGESTION" />
  <ItemData ItemOID="ID.AESTMON" Value="06" />
  <ItemData ItemOID="ID.AESTDAY" Value="11" />
  <ItemData ItemOID="ID.AESTYR" Value="1999" />
  <ItemData ItemOID="ID.AESTDT" Value="1999-06-11" />
  <ItemData ItemOID="ID.AEENMON" Value="" />
  <ItemData ItemOID="ID.AEENDAY" Value="" />
  <ItemData ItemOID="ID.AEENYR" Value="" />
  <ItemData ItemOID="ID.AEENDT" Value="" />
  <ItemData ItemOID="ID.AESEV" Value="1" />
  <ItemData ItemOID="ID.AEREL" Value="0" />
  <ItemData ItemOID="ID.AEOUT" Value="2" />
  <ItemData ItemOID="ID.AEACTTRT" Value="0" />
  <ItemData ItemOID="ID.AECONTRT" Value="1" />
  </ItemGroupData>
  </FormData>
  </StudyEventData>
  </SubjectData>
  </ClinicalData>
  </ODM>
```