Contents

What's New in the SAS 9.4 XMLV2 Engine

PART 1 Usage 1

Chapter 1 / Getting Started with the XMLV2 Engine .............................................................. 3
What Does the XMLV2 Engine Do? ................................................................. 3
Understanding How the XMLV2 Engine Works ............................................. 4
SAS Processing Supported by the XMLV2 Engine ....................................... 6
Transferring an XML Document across Environments ................................ 6
Frequently Asked Questions ........................................................................... 7
Accessibility Features of the XMLV2 Engine ................................................ 8

Chapter 2 / Importing XML Documents ................................................................. 9
Understanding How to Import an XML Document .................................... 9
Importing an XML Document Using the GENERIC Markup Type .............. 9
Importing an XML Document with Non-Escaped Character Data .............. 11

Chapter 3 / Exporting XML Documents .............................................................. 15
Understanding How to Export an XML Document ................................... 15
Exporting an XML Document That Contains SAS Dates, Times, and Datetimes ................................................. 15
Exporting an XML Document with Separate Metadata ......................... 16

Chapter 4 / Importing XML Documents Using an XMLMap ..................... 19
Why Use an XMLMap When Importing? ....................................................... 19
Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type .......................................................... 20
Using an XMLMap to Import an XML Document as One SAS Data Set .... 23
Using an XMLMap to Import an XML Document as Multiple SAS Data Sets ................................................................. 26
Importing Hierarchical Data as Related Data Sets .................................. 29
Including a Key Field with Generated Numeric Keys ................................ 33
Determining the Observation Boundary to Avoid Concatenated Data ........ 36
Using ISO 8601 SAS Informats and Formats to Import Dates ................. 39
Using ISO 8601 SAS Informats and Formats to Import Time Values with a Time Zone ................................................. 41
Referencing a Fileref Using the URL Access Method ......................... 43
Specifying a Location Path on the PATH Element ..................................... 44
Including Namespace Elements in an XMLMap ...................................... 47
Using the AUTOMAP= Option to Generate an XMLMap .................... 50
Avoiding Truncation Errors When Importing in a Multi-Byte Encoding .... 54

Chapter 5 / Exporting XML Documents Using an XMLMap ..................... 57
Why Use an XMLMap when Exporting? ....................................................... 57
Using an XMLMap to Export an XML Document with a Hierarchical Structure ................................................................. 57
<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Understanding and Using Tagsets for the XMLV2 Engine</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Is a Tagset?</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Creating Customized Tagsets</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Exporting an XML Document Using a Customized Tagset</td>
<td>62</td>
<td></td>
</tr>
</tbody>
</table>

PART 2 LIBNAME Statement Syntax 71

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th>Introduction to the XMLV2 LIBNAME Statement</th>
<th>73</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the LIBNAME Statement</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>Comparing the XMLV2 and XML Engines</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8</th>
<th>LIBNAME Statement for the XMLV2 Engine</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>79</td>
<td></td>
</tr>
</tbody>
</table>

PART 3 XMLMap File Syntax 95

<table>
<thead>
<tr>
<th>Chapter 9</th>
<th>Introduction to XMLMap Syntax</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using XMLMap Syntax</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>Comparing the XMLMap Syntax</td>
<td>98</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 10</th>
<th>XMLMap Syntax Version 2.1</th>
<th>101</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 11</th>
<th>Using SAS XML Mapper to Generate and Update an XMLMap</th>
<th>119</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Is SAS XML Mapper?</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>Using the Windows</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Using the Menu Bar</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Using the Toolbar</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>How Do I Get SAS XML Mapper?</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>To Start SAS XML Mapper</td>
<td>121</td>
<td></td>
</tr>
</tbody>
</table>

PART 4 Appendixes 123

<table>
<thead>
<tr>
<th>Appendix 1</th>
<th>Usage Not Supported for the XMLV2 Engine</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exporting an XML Document for Use by Oracle</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>Exporting Numeric Values</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>Exporting an XML Document in CDISC ODM Markup</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>Importing an XML Document with Numeric Values</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>Importing an XML Document Created by Microsoft Access</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Importing Concatenated XML Documents</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>Importing a CDISC ODM Document</td>
<td>140</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appendix 2</th>
<th>Example CDISC ODM Document</th>
<th>143</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example CDISC ODM Document</td>
<td>143</td>
<td></td>
</tr>
</tbody>
</table>
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

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=REPLACE|REUSE on page 82.

New XMLV2 LIBNAME Statement Options

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

- **CHARMULTIPLIER=** option XMLV2 Only expands column (variable) lengths by a multiplier value. These column (variable) lengths are specified in an XMLMap.

- **DERIVECHARMULTIPLIER=** option XMLV2 Only 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=** option XMLV2 Only 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 with a superscript such 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 125. For more information, see “Comparing the XMLV2 and XML Engines” on page 73.

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

Chapter 1
Getting Started with the XMLV2 Engine .............................................. 3

Chapter 2
Importing XML Documents ................................................................. 9

Chapter 3
Exporting XML Documents ............................................................... 15

Chapter 4
Importing XML Documents Using an XMLMap .................................. 19

Chapter 5
Exporting XML Documents Using an XMLMap .................................. 57

Chapter 6
Understanding and Using Tagsets for the XMLV2 Engine ................... 61
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 PRINT procedure results in the following output:

```
#  Name       Member Type
 1  STUDENTS  DATA
```
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:

```sas
libname myfiles 'C:\Example\';
libname myxml xmlv2 'C:\Output\Singers.xml';
```

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

```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>
```

Understanding How the XMLV2 Engine Works 5
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 is UTF-8 encoding. If the document's encoding is not compatible with UTF-8, 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
<?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 19.
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 73.

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.
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 20.

```xml
<?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>
</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;
```
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 &amp; and &apos;. For example, to allow attribute values to contain both single and double quotation marks, the apostrophe or single-quotation character (') can be represented as &apos; and the double-quotation character (") as &quot;.

To import an XML document that contains non-escaped characters, you can specify the LIBNAME statement option XMLPROCESS=PERMIT. 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.)
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
<?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>&lt;![CDATA[Abbott & Costello]]&gt;</ampersand>
    <squote>&lt;![CDATA[Logan's Run]]&gt;</squote>
    <dquote>&lt;![CDATA[As Benjamin Franklin advised, "Well done is better than well said."]]&gt;</dquote>
    <greater>&lt;![CDATA[ x > y ]]&gt;</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> &gt;</greater>
  </CHARS>
</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.

```sas
libname myfiles xmlv2 'C:\Example\strings.xml';
```

**Example Code 2.1  SAS Log Output**

```
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:

```sas
libname myfiles xmlv2 'C:\Example\strings.xml' xmlprocess=permit;
proc print data=myfiles.chars;
run;
```
### Output 2.2  PRINT Procedure Output for MyFiles.Cchars

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

```sas
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 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
<?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

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Advertised</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Advertised (1=yes)</td>
</tr>
<tr>
<td>5</td>
<td>Date</td>
<td>Num</td>
<td>8</td>
<td>DATE9.</td>
<td>Date of sale</td>
</tr>
<tr>
<td>4</td>
<td>Holiday</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Holiday (1=yes)</td>
</tr>
<tr>
<td>2</td>
<td>Price</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Retail price of product</td>
</tr>
<tr>
<td>6</td>
<td>Product</td>
<td>Char</td>
<td>40</td>
<td></td>
<td>Product name</td>
</tr>
<tr>
<td>1</td>
<td>QtySold</td>
<td>Num</td>
<td>8</td>
<td></td>
<td>Quantity sold</td>
</tr>
</tbody>
</table>

The following SAS program exports an XML document from the SAS data set
Sashelp.Snacks:

```sas
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 18.
Output 3.4  XML Document Snacks.XML

```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
<?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? ............................. 19

Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type ............................. 20
  What Is the Required Physical Structure? ............................. 20
  Why Is a Specific Physical Structure Required? ......................... 21
  Handling XML Documents That Are Not in the Required Physical Structure ............................. 23

Using an XMLMap to Import an XML Document as One SAS Data Set ............................. 23
Using an XMLMap to Import an XML Document as Multiple SAS Data Sets ............................. 26
Importing Hierarchical Data as Related Data Sets ............................. 29
Including a Key Field with Generated Numeric Keys ............................. 33
Determining the Observation Boundary to Avoid Concatenated Data ............................. 36
Using ISO 8601 SAS Informats and Formats to Import Dates ............................. 39
Using ISO 8601 SAS Informats and Formats to Import Time Values with a Time Zone ............................. 41
Referencing a Fileref Using the URL Access Method ............................. 43
Specifying a Location Path on the PATH Element ............................. 44
Including Namespace Elements in an XMLMap ............................. 47
Using the AUTOMAP= Option to Generate an XMLMap ............................. 50
Avoiding Truncation Errors When Importing in a Multi-Byte Encoding ............................. 54

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
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 11, “Using SAS XML Mapper to Generate and Update an XMLMap,” on page 119.

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

The AUTOMAP= option in the LIBNAME statement provides another alternative to creating an XMLMap. AUTOMAP= specifies to automatically generate an XMLMap file to import an XML document. See the AUTOMAP= option on page 82.

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

What Is the Required Physical Structure?

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

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

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

```xml
<?xml version="1.0" encoding="windows-1252" ?>
<LIBRARY> <!--
1
-->
  <STUDENTS> <!--
2
-->
    <ID> 0755 </ID>
    <NAME> Brad Martin </NAME>
    <ADDRESS> 1611 Glengreen </ADDRESS>
    <CITY> Huntsville </CITY>
    <STATE> Texas </STATE>
  </STUDENTS>
<!--
3
-->
<STUDENTS> <!--
4
-->
</LIBRARY>
```
Understanding the Required Physical Structure for an XML Document to Be Imported
Using the GENERIC Markup Type 21

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

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

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
2
-->
  <HIGHTEMP> <!--
1-->
  
  <LOWTEMP> <!--
2-->
</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;
```
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 19.

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
<?xml version="1.0" encoding="iso-8859-1" ?>
<NHL>
  <CONFERENCE> Eastern
    <DIVISION> Southeast
      <TEAM name="Thrashers" abbrev="ATL" />
      <TEAM name="Hurricanes" abbrev="CAR" />
      <TEAM name="Panthers" abbrev="FLA" />
      <TEAM name="Lightning" abbrev="TB" />
      <TEAM name="Capitals" abbrev="WSH" />
    </DIVISION>
  </CONFERENCE>
</NHL>
```
To successfully import the XML document, an XMLMap is needed. After familiarizing yourself with the data to be imported, you can code the XMLMap syntax so that the data is successfully imported. Here is the XMLMap used to import the XML document, with notations for the data investigation:

```xml
<?xml version="1.0" ?>
<SXLEMAP version="2.1">
<TABLE name="TEAMS"> <!-- 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>
```
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 26.)

2. Identify the SAS data set observation boundary, which translates into a collection of rows with a constant set of columns.
   
   In the XML document, information about individual teams occurs in a <TEAM> tag located with <CONFERENCE> and <DIVISION> enclosures. You want a new observation generated each time a TEAM element is read.

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

4. Add foreign keys or required external context.
   
   You want to include information about the league orientation for the teams. Also, you want to extract CONFERENCE and DIVISION data.

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

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

The following SAS statements import the XML document Nhl.xml:

```sas
filename nhl 'C:\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 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 57.

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. RSS.xml 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 XML LIBNAME engine. The XML content in this example uses a simplified set of RSS (rich site summary) tags.

```xml
<?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
<?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>
    </COLUMN>
  </TABLE>
  <TABLE name="ITEM"> <!-- 2 -->
    <TABLE-PATH syntax="XPath">/rss/channel/item</TABLE-PATH> <!-- 3 -->
    <COLUMN name="Item_Title"> <!-- 4 -->
      <PATH syntax="XPath">/rss/channel/item/title</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>200</LENGTH>
    </COLUMN>
    <COLUMN name="Item_Description">
      <PATH syntax="XPath">/rss/channel/item/description</PATH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```
SXLEMAP is the root-enclosing element for SAS data set definitions.

This TABLE element defines the Channel data set.

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

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.

This TABLE element defines the Item data set.

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

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

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

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

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

```xml
<?xml version="1.0" ?>
<PHARMACY>
  <PERSON>
    <NAME>Brad Martin</NAME>
    <STREET>11900 Glenda Court</STREET>
    <CITY>Austin</CITY>
  </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.
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.

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

First TABLE element defines the Person data set.

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

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

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

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

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

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

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

Output 4.5  DATASETS Procedure Output for Pharm Library

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

Output 4.6  PRINT Procedure Output for Pharm.Person

<table>
<thead>
<tr>
<th>Obs</th>
<th>NAME</th>
<th>STREET</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Brad Martin</td>
<td>11900 Gienda Court</td>
<td>Austin</td>
</tr>
<tr>
<td>2</td>
<td>Jim Spano</td>
<td>1611 Glengreen</td>
<td>Austin</td>
</tr>
</tbody>
</table>
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 29. This example continues with the XMLMap by adding a key field with generated numeric key values to provide a relationship between the two data sets. (A key field holds unique data to identify that record from the other records. For example, account number, product code, and customer name are typical key fields.)

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

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

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

```xml
<?xml version="1.0" encoding="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>
  <COLUMN class="ORDINAL" name="KEY" retain="YES">
    <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>
  <COLUMN class="ORDINAL" name="KEY" retain="YES">
    <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>
</TABLE>
</SXLEMAP>
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.

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.

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.

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 XMLV2 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 XMLV2 engine for the Prescription data set Key variable is the same as the Person data set Key variable. Although the XMLV2 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.
The result is three variables and three observations.

The following SAS statements import the XML document:

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

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

**Output 4.8** *PRINT Procedure Output for Pharm.Person*

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

**Output 4.9** *PRINT Procedure Output for Pharm.Prescription*

```
<table>
<thead>
<tr>
<th>Obs</th>
<th>KEY</th>
<th>NUMBER</th>
<th>DRUG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>001</td>
<td>1234</td>
<td>Tetracycline</td>
</tr>
<tr>
<td>2</td>
<td>001</td>
<td>1245</td>
<td>Lomotil</td>
</tr>
<tr>
<td>3</td>
<td>002</td>
<td>1258</td>
<td>Nexium</td>
</tr>
</tbody>
</table>
```

**Determining the Observation Boundary to Avoid Concatenated Data**

This example illustrates how to determine the observation boundary so that an import results in separate observations and not concatenated data.

The observation boundary translates into a collection of rows with a constant set of columns. Using an XMLMap, you determine the observation boundary with the TABLE-PATH element by specifying a location path. The end tag for the location path determines when data is written to the SAS data set as an observation. Identifying the observation boundary can be tricky due to sequences of start-tag and end-tag pairing. If you do not identify the appropriate observation boundary, the result could be a concatenated data string instead of separate observations. This example illustrates pairing situations that can cause unwanted results.
For the following XML document, an XMLMap is necessary to import the file successfully. Without an XMLMap, the XMLV2 engine would import a data set named Ford with variables Row0, Model0, Year0, Row1, Model1, Year1, and so on.

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

Looking at the above XML document, there are three sequences of element start tags and end tags: VEHICLES, FORD, and ROW. If you specify the following table location path and column locations paths, the XMLV2 engine processes the XML document as follows:

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

1. The XMLV2 engine reads the XML markup until it encounters the <FORD> start tag, because FORD is the last element specified in the table location path.

2. The XMLV2 engine clears the input buffer and scans subsequent elements for variables based on the column location paths. As a value for each variable is encountered, it is read into the input buffer. For example, after reading the first ROW element, the input buffer contains the values Mustang and 1965.

3. The XMLV2 engine continues reading values into the input buffer until it encounters the </FORD> end tag. At this time, the engine writes the completed input buffer to the SAS data set as an observation.

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

To get separate observations, you must change the table location path so that the XMLV2 engine writes separate observations to the SAS data set. Here are the correct location paths and the process that the engine would follow:

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

1. The XMLV2 engine reads the XML markup until it encounters the <ROW> start tag, because ROW is the last element specified in the table location path.
The XMLV2 engine clears the input buffer and scans subsequent elements for variables based on the column location paths. As a value for each variable is encountered, it is read into the input buffer.

The XMLV2 engine continues reading values into the input buffer until it encounters the </ROW> end tag. At this time, the engine writes the completed input buffer to the SAS data set as an observation. That is, one observation is written to the SAS data set that contains the values Mustang and 1965.

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

The result is four observations.

Here is the complete XMLMap syntax:

```xml
<?xml version="1.0" ?>
<SXLEMAP version="2.1" name="path" description="XMLMap for path">
  <TABLE name="FORD">
    <TABLE-PATH syntax="XPath"> /VEHICLES/FORD/ROW </TABLE-PATH>
    <COLUMN name="Model">
      <DATATYPE> string </DATATYPE>
      <LENGTH> 20 </LENGTH>
      <TYPE> character </TYPE>
      <PATH syntax="XPath"> /VEHICLES/FORD/ROW/Model </PATH>
    </COLUMN>
    <COLUMN name="Year">
      <DATATYPE> string </DATATYPE>
      <LENGTH> 4 </LENGTH>
      <TYPE> character </TYPE>
      <PATH syntax="XPath"> /VEHICLES/FORD/ROW/Year </PATH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```

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

```sas
filename path 'C:\Example\path.xml';
filename map 'C:\Example\path.map';
libname path xmlv2 xmlmap=map;
proc print data=path.ford noobs;
run;
```
Using ISO 8601 SAS Informs and Formats to Import Dates

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

Here is the XML document:

```xml
<?xml version="1.0" ?>
<Root>
  <ISODATE>
    <BASIC>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
<?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 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904500.0.0.20170816190000_v940m5 -->
<!-- >>> Validation report >>> -->
<!-- >>> Validation report >>> -->
<!-- >>> Validation report >>> -->
<!-- >>> Validation report >>> -->
<SXLEMAP description="Reading a Basic and Extended format ISO date field" name="ISOdate" version="2.1">
  <NAMESPACE count="0"/>
</SXLEMAP>
```
The following explains the XMLMap syntax that imports the date values:

1. For the Basic variable, the FORMAT element specifies the E8601DA SAS format, which writes data values in the extended format `yyyy-mm-dd`.

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

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

3. For the Extended variable, the FORMAT element specifies the E8601DA SAS format, which writes data values in the extended format `yyyy-mm-dd`.

4. For the Extended variable, the INFORMAT element specifies the E8601DA SAS informat, which reads date values into a variable in the basic format `yyyy-mm-dd`.

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

```sas
filename dates 'C:\Example\isodate.xml';
filename map 'C:\Example\isodate.map';
libname dates xmlv2 xmlmap=map;
proc print data=dates.isodate;
run;
```
This example illustrates importing an XML document that contains time values in various forms. The XMLMap uses the FORMAT and INFORMAT elements to specify the appropriate SAS formats and SAS informats in order to represent the times appropriately.

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

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

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- ############################################################ -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 903000.1.0.20101208190000_v930 -->
<!-- ###  Validation report                                   ### -->
<!-- ############################################################ -->
<!-- # Validation report # -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="ISOtime" version="2.1">
    <NAMESPACE count="0"/>
</SXLEMAP>

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

    <COLUMN name="LOCAL">
```
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 \texttt{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 \textit{Offset} variable, the \texttt{INFORMAT} and \texttt{FORMAT} elements specify the E8601TZ SAS informat and format, which reads and writes time values in the extended format \texttt{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 \texttt{PRINT} procedure output:

\begin{verbatim}
filename timzn 'C:\Example\Time.xml';
filename map 'C:\Example\Time.map';
libname timzn xmlv2 xmlmap=map;
proc print data=timzn.time;
run;
\end{verbatim}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{Obs} & \textbf{LOCAL} & \textbf{LOCALZONE} & \textbf{UTC} & \textbf{OFFSET} \\
\hline
1 & 09:00:00 & 09:00:00-05:00 & 09:00:00Z & 09:00:00+00:00 \\
\hline
\end{tabular}
\caption{PRINT Procedure Output for Imported Data Set Timzn.Time}
\end{table}

\section*{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:

\begin{verbatim}
filename mynhl url 'http://www.a.com/NHL.xml'; /*1*/
filename mymap 'C:\Example\NHL.map'; /*2*/
libname mynhl xmlv2 xmlmap=mymap; /*3*/
\end{verbatim}
The first `FILENAME` statement assigns the fileref MyNhl to the XML document by using the URL access method.

The second `FILENAME` statement assigns the fileref MyMap to the physical location of the XMLMap Nhl.map.

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.

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

Output 4.13  PRINT Procedure Output for MyNhl.TEAMS

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

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
<?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
<?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>
  </TABLE>
</SXLEMAP>
```

Specifying a Location Path on the PATH Element 45
The Abbrev variable uses the attribute form that selects values from a specific attribute. The engine scans the XML markup until it finds the TEAM element. The engine retrieves the value from the abbrev= attribute, which results in each team abbreviation.

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

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

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

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

The following SAS statements import the XML document NhlShort.xml and specify the XMLMap named Nhl1.map. The PRINT procedure shows the resulting variables with selected values:

```sas
filename nhl 'C:\Example\NhlShort.xml';
filename map 'C:\Example\Nhl1.map';
libname nhl xmlv2 xmlmap=map;
proc print data=nhl.teams noobs;
run;
```
Including Namespace Elements in an XMLMap

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

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PEOPLE xmlns:HOME="http://sample.url.org/home"
xmlns:IP="http://sample.url.org/ip"
xmlns:WORK="http://sample.url.org/work">
  <PERSON>
    <NAME>Joe Smith</NAME>
    <HOME:ADDRESS>1234 Elm Street</HOME:ADDRESS>
    <HOME:PHONE>999-555-0011</HOME:PHONE>
    <WORK:ADDRESS>2001 Office Drive, Box 101</WORK:ADDRESS>
    <WORK:PHONE>999-555-0101</WORK:PHONE>
    <IP:ADDRESS>192.168.1.1</IP:ADDRESS>
  </PERSON>
  <PERSON>
    <NAME>Jane Jones</NAME>
    <HOME:ADDRESS>9876 Main Street</HOME:ADDRESS>
    <HOME:PHONE>999-555-0022</HOME:PHONE>
    <WORK:ADDRESS>2001 Office Drive, Box 102</WORK:ADDRESS>
    <WORK:PHONE>999-555-0102</WORK:PHONE>
    <IP:ADDRESS>172.16.1.2</IP:ADDRESS>
  </PERSON>
  <PERSON>
    <NAME>Pat Perkinson</NAME>
    <HOME:ADDRESS>1395 Half Way</HOME:ADDRESS>
  </PERSON>
</PEOPLE>
```
Chapter 4 / Importing XML Documents Using an XMLMap

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

```xml
<SXLEMAP name="Namespace" version="2.1">
  <NAMESPACES count="3">
    <NS id="1" prefix="HOME">http://sample.url.org/home</NS>
    <NS id="2" prefix="IP">http://sample.url.org/ip</NS>
    <NS id="3" prefix="WORK">http://sample.url.org/work</NS>
  </NAMESPACES>
  <TABLE description="PERSON" name="PERSON">
    <TABLE-PATH>/PEOPLE/PERSON</TABLE-PATH>
    <COLUMN name="NAME">
      <PATH>/PEOPLE/PERSON/NAME</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>13</LENGTH>
    </COLUMN>
    <COLUMN name="ADDRESS">
      <PATH>/PEOPLE/PERSON/ADDRESS</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>16</LENGTH>
    </COLUMN>
    <COLUMN name="PHONE">
      <PATH>/PEOPLE/PERSON/PHONE</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>12</LENGTH>
    </COLUMN>
    <COLUMN name="ADDRESS1">
      <PATH>/PEOPLE/PERSON/ADDRESS1</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>26</LENGTH>
    </COLUMN>
    <COLUMN name="PHONE1">
      <PATH>/PEOPLE/PERSON/PHONE1</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>12</LENGTH>
    </COLUMN>
    <COLUMN name="ADDRESS2">
      <PATH>/PEOPLE/PERSON/ADDRESS2</PATH>
      <TYPE>character</TYPE>
      <DATATYPE>string</DATATYPE>
      <LENGTH>26</LENGTH>
    </COLUMN>
  </TABLE>
</SXLEMAP>
```
Including Namespace Elements in an XMLMap

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>ADDRESS</th>
<th>PHONE</th>
<th>ADDRESS1</th>
<th>PHONE1</th>
<th>ADDRESS2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe Smith</td>
<td>1234 Elm Street</td>
<td>999-555-0011</td>
<td>2001 Office Drive, Box 101</td>
<td>999-555-0101</td>
<td>152.18:1.1</td>
</tr>
<tr>
<td>Jane Jones</td>
<td>5676 Main Street</td>
<td>999-555-0022</td>
<td>2001 Office Drive, Box 102</td>
<td>999-555-0102</td>
<td>172.16:1.2</td>
</tr>
<tr>
<td>Pat Parkinson</td>
<td>1395 Half Way</td>
<td>999-555-0033</td>
<td>2001 Office Drive, Box 103</td>
<td>999-555-0103</td>
<td>10.0.1.3</td>
</tr>
</tbody>
</table>
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. 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
<?xml version="1.0" encoding="iso-8859-1" ?>
<NHL>
  <CONFERENCE> Eastern
    <DIVISION> Southeast
      <TEAM name="Thrashers" abbrev="ATL" />
      <TEAM name="Hurricanes" abbrev="CAR" />
      <TEAM name="Panthers" abbrev="FLA" />
      <TEAM name="Lightning" abbrev="TB" />
      <TEAM name="Capitals" abbrev="WSH" />
    </DIVISION>
  </CONFERENCE>
  <CONFERENCE> Western
    <DIVISION> Pacific
      <TEAM name="Stars" abbrev="DAL" />
      <TEAM name="Kings" abbrev="LA" />
      <TEAM name="Ducks" abbrev="ANA" />
      <TEAM name="Coyotes" abbrev="PHX" />
      <TEAM name="Sharks" abbrev="SJ" />
    </DIVISION>
  </CONFERENCE>
</NHL>
```

The following SAS statements import the XML document Nhl.xml:

```sas
filename nhl 'C:\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 pathname, filename, and file extension) of the XML document named Nhl.xml to be imported.

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

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

Here is the generated XMLMap NhlGenerate.map:

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

<!-- ############################################################ -->
<!-- 2012-09-21T10:30:48 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 904000.4.0.20120905190000_v940 -->
<!-- ############################################################ -->
<!-- ### Validation report ### -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<XMLMAP 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>
</TABLE>
</XMLMAP>
```
<table>
<thead>
<tr>
<th>CONFERENCE_ORDINAL</th>
<th>CONFERENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NHL</td>
</tr>
<tr>
<td>2</td>
<td>NHL</td>
</tr>
<tr>
<td>3</td>
<td>NHL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIVISION_ORDINAL</th>
<th>DIVISION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Western</td>
</tr>
<tr>
<td>2</td>
<td>Eastern</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TEAM_ORDINAL</th>
<th>TEAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Team1</td>
</tr>
<tr>
<td>2</td>
<td>Team2</td>
</tr>
<tr>
<td>3</td>
<td>Team3</td>
</tr>
</tbody>
</table>
Here is the PRINT procedure output for Nhl.Team

Output 4.16  PRINT Procedure Output for Nhl.Team

The SAS System

<table>
<thead>
<tr>
<th>Obs</th>
<th>DIVISION_ORDINAL</th>
<th>TEAM_ORDINAL</th>
<th>TEAM_name</th>
<th>TEAM_abbrev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Thrashers</td>
<td>ATL</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Hurricanes</td>
<td>CAR</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>Panthers</td>
<td>FLA</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>Lightning</td>
<td>TB</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5</td>
<td>Capitals</td>
<td>WSH</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>6</td>
<td>Stars</td>
<td>DAL</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>7</td>
<td>Kings</td>
<td>LA</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>8</td>
<td>Ducks</td>
<td>ANA</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>9</td>
<td>Coyotes</td>
<td>PHX</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>10</td>
<td>Sharks</td>
<td>SJ</td>
</tr>
</tbody>
</table>
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 50. (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

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DIVISION_ORDINAL</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>TEAM_ORDINAL</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>TEAM_abbrev</td>
<td>Char</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>TEAM_name</td>
<td>Char</td>
<td>10</td>
</tr>
</tbody>
</table>
```

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:

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DIVISION_ORDINAL</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>TEAM_ORDINAL</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>TEAM_abbrev</td>
<td>Char</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>TEAM_name</td>
<td>Char</td>
<td>25</td>
</tr>
</tbody>
</table>
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 101 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 23.

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

```xml
<?xml version="1.0" encoding="windows-1252" ?>
<TABLE>
  <TEAMS>
    <NAME>Thrashers</NAME>
    <ABBREV>ATL</ABBREV>
    <CONFERENCE>Eastern</CONFERENCE>
    <DIVISION>Southeast</DIVISION>
  </TEAMS>
  <TEAMS>
    <NAME>Hurricanes</NAME>
    <ABBREV>CAR</ABBREV>
    <CONFERENCE>Eastern</CONFERENCE>
    <DIVISION>Southeast</DIVISION>
  </TEAMS>
  ...
</TABLE>
```

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

```xml
<?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 101 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
<?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>
```
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:

```sas
proc template;
/* +------------------------------------------------+*/
|                                                |
/* +------------------------------------------------+*/
define tagset tagsets.custom ;
notes "SAS XMLV2 engine output event model(interface)";
indent = 3;
map = '<>&"''';
mapsub = '/&lt;/&gt;/&quot;/&apos;/';
/* +------------------------------------------------+*/
|                                                |
/* +------------------------------------------------+*/
define event XMLversion;
  put  '<?xml version="1.0"';
  putq ' encoding=' ENCODING;
  put  ' ?>' CR;
  break;
end;

/* +------------------------------------------------+*/
|                                                |
/* +------------------------------------------------+*/
define event XMLcomment;
  put  '<!-- ' CR;
  break;
end;
```

62 Chapter 6 / Understanding and Using Tagsets for the XMLV2 Engine
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 ;
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");
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 ;
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*.

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

The EmitCol event creates generic-looking XML for each processed variable.
Export XML Document Using Customized Tagset

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

```
data work.class; /*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 pathname, filename, 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:
Exporting an XML Document Using a Customized Tagset

Output 6.1 Exported XML Document Using Customized Tagset

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

LIBNAME Statement Syntax

Chapter 7
Introduction to the XMLV2 LIBNAME Statement .......................... 73

Chapter 8
LIBNAME Statement for the XMLV2 Engine ............................... 79
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 19
- Chapter 5, “Exporting XML Documents Using an XMLMap,” on page 57

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 9, “Introduction to XMLMap Syntax,” on page 97.

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.

### Table 7.1 LIBNAME Statement Options

<table>
<thead>
<tr>
<th>Task</th>
<th>Option</th>
<th>XML</th>
<th>XMLV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically generate an XMLMap file to import an XML document</td>
<td>AUTOMAP= on page 82</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>Expand XMLMap column (variable) lengths by a multiplier value</td>
<td>CHARMULTIPLIER= on page 83</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>Expand XMLMap column (variable) lengths by a default multiplier value that is based on the session encoding</td>
<td>DERIVECHARMULTIPLIER= on page 83</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>Determine whether SAS formats are used with the GENERIC markup type</td>
<td>FORMATACTIVE= on page 83</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Determine whether SAS formats are used with the CDISCODM markup type</td>
<td>FORMATACTIVE= on page 83</td>
<td>■</td>
<td></td>
</tr>
<tr>
<td>Specify the libref to create a format catalog with the CDISCODM markup type</td>
<td>FORMATLIBRARY= on page 84</td>
<td>■</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Option</td>
<td>XML</td>
<td>XMLV2</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>Replace existing format entries in the format catalog with the CDISCODM markup type</td>
<td>FORMATNOREPLACE = on page 85</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Indent nested elements in exported XML document</td>
<td>INDENT= on page 85</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Specify the character set to use for the output file</td>
<td>ODSCHARSET= on page 85</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Control the generation of a record separator</td>
<td>ODSRECSEP= on page 85</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Specify the translation table to use for the output file</td>
<td>ODMSTRANTAB= on page 86</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element</td>
<td>PREFIXATTRIBUTES= on page 86</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Override the default tagset</td>
<td>TAGSET= on page 87</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Import concatenated XML documents</td>
<td>XMLCONCATENATE= on page 87</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Specify the tag format to contain SAS variable information</td>
<td>XMLDATAFORM= on page 87</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Control the results of numeric values</td>
<td>XMLDOUBLE= on page 88</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Override the SAS data set's encoding for the output file</td>
<td>XMLENCODING= on page 89</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Option</td>
<td>XML</td>
<td>XMLV2</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------------------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>Specify a fileref for the XML document</td>
<td>XMLFILEREF= on page 89</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Specify an XMLMap</td>
<td>XMLMAP= on page 89</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Determine whether metadata-related information is included</td>
<td>XMLMETA= on page 90</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Determine whether to process nonconforming character data</td>
<td>XMLPROCESS= on page 91</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>Specify an external file to contain exported metadata-related</td>
<td>XMLSCHEMA= on page 92</td>
<td>☑</td>
<td>☑</td>
</tr>
<tr>
<td>information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify the XML markup type</td>
<td>XMLTYPE= on page 92</td>
<td>☑</td>
<td>☑</td>
</tr>
</tbody>
</table>
LIBNAME Statement for the XMLV2 Engine

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>;

Summary of Optional Arguments

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

CHARMULTIPLIER=value XMLV2 only
expands column (variable) lengths by a multiplier value. These are column (variable) lengths specified in an XMLMap.

DERIVECHARMULTIPLIER=YES | NO XMLV2 only
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.

FORMATACTIVE=NO | YES XMLV2 and XML with differences
determines whether SAS formats are used.

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

**FORMATNOREPLACE=** NO | YES  
XML only
specifies whether to replace existing format entries in the format catalog search path in cases where an existing format entry has the same name as a format that is being created by the XML engine when it converts a CDISC ODM CodeList element.

**INDENT=** integer  
XMLV2 and XML
specifies the number of columns to indent each nested element in the exported XML document.

**ODSCHARSET=** character-set  
XMLV2 and XML
specifies the character set to use for the output file.

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

**ODSTRANTAB=** table-name  
XMLV2 and XML
specifies the translation table to use for the output file.

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

**TAGSET=** tagset-name  
XMLV2 and XML
specifies the name of a tagset to override the default tagset that is used by the markup type that is specified with XMLTYPE=.

**XMLCONCATENATE=** NO | YES  
XMLV2 and XML
specifies whether the file to be imported contains multiple, concatenated XML documents.

**XMLDATAFORM=** ELEMENT | ATTRIBUTE  
XMLV2 and XML
specifies whether the tag for the element to contain SAS variable information (name and data) is in open element or enclosed attribute format.

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

**XMLENCODING='** encoding-value **'**  
XMLV2 and XML
overrides the SAS data set's encoding for the output file.

**XMLFILEREF=** fileref  
XMLV2 and XML
is the SAS name that is assigned to the physical location of the XML document to be exported or imported.

**XMLMAP=** fileref | 'XMLMap-location'  
XMLV2 and XML
specifies an XML document that contains specific XMLMap syntax.

**XMLMETA=** DATA | SCHEMADATA | SCHEMA  
XMLV2 and XML
specifies whether to include metadata-related information in the exported markup, or specifies whether to import metadata-related information that is included in the input XML document.

**XMLPROCESS=** CONFORM | PERMIT  
XMLV2 and XML
determines how the engine processes character data that does not conform to W3C specifications.

**XMLSCHEMA=** fileref | 'external-file'  
XMLV2 and XML
specifies an external file to contain metadata-related information.

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

**libref**

is a valid SAS name that serves as a shortcut name to 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 with a superscript such 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 73.

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

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

**'SAS-library' XMLV2 only**

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

'C:\My Documents\XML'.

**'XML-document-path' XMLV2 and XML**

includes the pathname, filename, and file extension. Here is an example:

'C:\My Documents\XML\myfile.xml'.

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

**Interactions** You can use the FILENAME statement 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:

```sas
filename bedrock 'C:\XMLdata\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=** option on page 89.
Optional Arguments

**AUTOMAP=REPLACE | REUSE**  
XMLV2 only

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

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.

**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.

**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= 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 for the DISK or TEMP device type only. The AUTOMAP= option does not support accessing an XMLMap file by using access methods such as FTP, SFTP, URL, or WebDAV.

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

<table>
<thead>
<tr>
<th>Tips</th>
</tr>
</thead>
<tbody>
<tr>
<td>The functionality to automatically generate an XMLMap file is also available with SAS XML Mapper. The \texttt{AUTOMAP=} option provides the functionality to create and use the XMLMap with a single \texttt{LIBNAME} statement.</td>
</tr>
<tr>
<td>You can assign the generated XMLMap file to a temporary location, which is deleted at the end of the SAS session. To do this, assign a file reference to a physical location and specify the \texttt{TEMP} device type.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Using the \texttt{AUTOMAP=} Option to Generate an XMLMap” on page 50</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\texttt{CHARMULTIPLIER=} \texttt{value} \texttt{XMLV2 only}</th>
</tr>
</thead>
<tbody>
<tr>
<td>expands column (variable) lengths by a multiplier value. These are column (variable) lengths specified in an XMLMap. Use \texttt{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.</td>
</tr>
<tr>
<td>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 \textit{SAS National Language Support (NLS): Reference Guide}.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Avoiding Truncation Errors When Importing in a Multi-Byte Encoding” on page 54</td>
</tr>
</tbody>
</table>

| \texttt{DERIVECHARMULTIPLIER=} \texttt{YES | NO} \texttt{XMLV2 only} |
|---------------------------------------------------------------|
| 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. Use \texttt{DERIVECHARMULTIPLIER=} 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 \texttt{DERIVECHARMULTIPLIER=} \texttt{YES} rather than \texttt{CHARMULTIPLIER=} when you are uncertain of the session encoding. For information about DBCS and MBCS issues, see \textit{SAS National Language Support (NLS): Reference Guide}. |

<table>
<thead>
<tr>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>NO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>If \texttt{CHARMULTIPLIER=} is also specified, then \texttt{DERIVECHARMULTIPLIER=} is ignored.</td>
</tr>
</tbody>
</table>

| \texttt{FORMATACTIVE=} \texttt{NO | YES} \texttt{XMLV2 and XML with differences} |
|---------------------------------------------|
| determines whether SAS formats are used. Behavior differs between the two supported markup types as follows. |
For the CDISCODM markup type: XML only

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.

Tips
By default, the format catalog is created in the Work library. If you want to store the catalog in a permanent library, use the FORMATLIBRARY= option on page 84.

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 132

For the GENERIC markup type: XMLV2 and XML

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.

Default
NO

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

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

Restrictions
Use this option when importing an XML document only.
Use this option only for the CDISCODM markup type with FORMATACTIVE=YES.

**FORMATNOREPLACE=NO | YES**  
XML only  
specifies whether to replace existing format entries in the format catalog search path in cases where an existing format entry has the same name as a format that is being created by the XML engine when it converts a CDISC ODM CodeList element.  

NO  
do not replace formats that have the same name.  

YES  
replaces formats that have the same name.  

Default  
NO  

Restrictions  
Use this option when importing an XML document only.  

Use this option for the CDISCODM markup type only.

**INDENT=integer**  
XMLV2 and XML  
specifies the number of columns to indent each nested element in the exported XML document. The value can be from 0 (which specifies no indention) through 32. This specification is cosmetic and is ignored by an XML-enabled browser.  

Default  
3  

Restriction  
Use this option when exporting an XML document only.

**ODSCHARSET=character-set**  
XMLV2 and XML  
specifies the character set to use for the output file. A character set includes letters, logograms, digits, punctuation, symbols, and control characters that are used for display and printing. An example of a character set is ISO-8859-1.  

Restriction  
Use this option when exporting an XML document only.

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

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

See  

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

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

The use of a record separator varies by operating environment.  

Tip  
If you do not transfer XML documents across environments, use the default behavior.
NONE
specifies to not generate a record separator.

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

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

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

YES
specifies to generate a record separator.

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

Restriction
Use this option when exporting an XML document only.

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

ODSTRANTAB=table-name XMLV2 and XML
specifies the translation table to use for the output file. The translation table (encoding method) is a set of rules that are used to map characters in a character set to numeric values. An example of a translation table is one that converts characters from EBCDIC to ASCII-ISO. The table-name can be any translation table that SAS provides or any user-defined translation table. The value must be the name of a SAS catalog entry in either the Sasuser.Profile catalog or the Sashelp.Host catalog.

Restriction
Use this option when exporting an XML document only.

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

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

See

PREFIXATTRIBUTES=YES | NO XMLV2 only
specifies whether the element name is concatenated to the attribute name when generating each XMLMap COLUMN element, which defines the SAS variable name. If you specify PREFIXATTRIBUTES=NO, the element name is not attached to the resulting SAS variable name.

Default
YES
## XMLV2 and XML Engines

### TAGSET=tagset-name

Specifies the name of a tagset to override the default tagset that is used by the markup type that is specified with XMLTYPE=. To change the tags that are produced, you can create a customized tagset and specify it with the TAGSET= option. For information about creating customized tagsets, see the TEMPLATE procedure in the *SAS Output Delivery System: Procedures Guide*.

- **Restriction**: Use this option when importing an XML document only.
- **Requirement**: Use this option with caution. If you are unfamiliar with XML markup, do not use this option.

### XMLCONCATENATE=NO | YES

Specifies whether the file to be imported contains multiple, concatenated XML documents. Importing multiple, concatenated XML documents can be useful (for example, if an application is producing a complete document per query or response as in a web form). For the XMLV2 engine, this option is not necessary and is ignored.

- **Alias**: XMLCONCAT=
- **Default**: NO
- **Interaction**: If you specify XMLCONCATENATE=YES, then XMLTYPE=GENERIC is required. Other XML markup types are not supported.

### XMLDATAFORM=ELEMENT | ATTRIBUTE

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

Default  
Restrictions  
Use this option when exporting an XML document only.

Use this option for the GENERIC markup type only.

**XMLDOUBLE=DISPLAY | INTERNAL**

controls the results of importing or exporting numeric values.

**DISPLAY**  
XMLV2 and XML

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**

**INTERNAL**  
XML import, XMLV2 and XML export

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**

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

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

Examples  
“Exporting Numeric Values” on page 127

“Importing an XML Document with Numeric Values” on page 133
**XMLENCODING='encoding-value'**  XMLV2 and XML

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

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

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

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

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

**See**

**XMLFILEREF=fileref**  XMLV2 and XML

is the SAS name that is assigned to the physical location of the XML document to be exported or imported. 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:\XMLdata\wilma.xml';
libname bedrock xmlv2 xilinxref=cartoon;
data bedrock.wilma;
  set source.wilma;
run;
```

**Tip**
When using the URL access method to reference a fileref that is assigned to an XML document, you should also specify an XMLMap. Specifying an XMLMap causes the 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 43.

**XMLMAP=fileref | 'XMLMap-location'**  XMLV2 and XML

specifies an XML document that contains specific XMLMap syntax. 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.

**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.

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

```
libname test xmlv2 'C:\Example\Nhl.xml' xmlmap='C:\Example\Nhl.map';
proc print data=test.teams;
run;
```

Restriction

See the XMLMap syntax on page 101 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.

See

Chapter 9, “Introduction to XMLMap Syntax,” on page 97

Examples

Chapter 4, “Importing XML Documents Using an XMLMap,” on page 19

Chapter 5, “Exporting XML Documents Using an XMLMap,” on page 57

XMLMETA=DATA | SCHEMADATA | SCHEMA

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

DATA

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

SCHEMADATA

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

<table>
<thead>
<tr>
<th>Default</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>Use this option for the GENERIC and MSACCESS markup types only.</td>
</tr>
<tr>
<td>Interaction</td>
<td>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.</td>
</tr>
<tr>
<td>Tip</td>
<td>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.</td>
</tr>
<tr>
<td>Examples</td>
<td>“Exporting an XML Document with Separate Metadata” on page 16</td>
</tr>
<tr>
<td></td>
<td>“Importing an XML Document Created by Microsoft Access” on page 134</td>
</tr>
</tbody>
</table>

XMLPROCESS=CONFORM | PERMIT  XMLV2 and XML

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

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

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

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

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

<table>
<thead>
<tr>
<th>Default</th>
<th>CONFORM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>“Importing an XML Document with Non-Escaped Character Data” on page 11</td>
</tr>
</tbody>
</table>
XMLSCHEMA=fileref | 'external-file' XMLV2 and XML
specifies an external file to contain metadata-related information.

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.

Restrictions Use this option when exporting an XML document only.

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

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

Example “Exporting an XML Document with Separate Metadata” on page 16

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

GENERIC XMLV2 and XML
is a simple, well-formed XML markup type. The XML document consists of a root (enclosing) element and repeating element instances. GENERIC determines a variable's attributes from the data content.

Requirement When importing, the GENERIC markup type requires a specific physical structure.

See “Understanding the Required Physical Structure for an XML Document to Be Imported Using the GENERIC Markup Type” on page 20

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

“Exporting Numeric Values” on page 127

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

CDISCODM XML only
is the XML markup type for the markup standards that are defined in the Operational Data Model (ODM) that was created by the Clinical Data Interchange Standards Consortium (CDISC). The XML engine supports the ODM 1.2 schema specification. ODM supports the electronic acquisition, exchange, and archiving of clinical trials data and metadata for medical and biopharmaceutical product development.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tip</strong></td>
<td>Use the FORMATACTIVE=, FORMATNOREPLACE=, and FORMATLIBRARY= options to specify how display data are read and stored in the target environment.</td>
</tr>
<tr>
<td><strong>Examples</strong></td>
<td>“Importing a CDISC ODM Document” on page 140</td>
</tr>
<tr>
<td></td>
<td>“Exporting an XML Document in CDISC ODM Markup” on page 132</td>
</tr>
<tr>
<td><strong>MSACCESS</strong></td>
<td><strong>XML only</strong>&lt;br&gt;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.</td>
</tr>
<tr>
<td></td>
<td>Example “Importing an XML Document Created by Microsoft Access” on page 134</td>
</tr>
<tr>
<td><strong>ORACLE</strong></td>
<td><strong>XML only</strong>&lt;br&gt;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.</td>
</tr>
<tr>
<td></td>
<td>Example “Exporting an XML Document for Use by Oracle” on page 125</td>
</tr>
<tr>
<td><strong>XMLMAP</strong></td>
<td><strong>XMLV2 only</strong>&lt;br&gt;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 XMLMAP= option on page 89.</td>
</tr>
<tr>
<td></td>
<td>Restriction Exporting an XML document that is controlled by an XMLMap is limited to a single SAS data set.</td>
</tr>
<tr>
<td></td>
<td>Example “Using an XMLMap to Export an XML Document with a Hierarchical Structure” on page 57</td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>GENERIC</td>
</tr>
<tr>
<td><strong>Tip</strong></td>
<td>You can control the markup by specifying options such as INDENT=, XMLDATAFORM=, XMLMETA= (when applicable), and TAGSET=.</td>
</tr>
</tbody>
</table>
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 101.
- 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 102.
- If you use an XMLMap for exporting, you must include the exporting elements. See “Elements for Exporting” on page 104.
- The table elements define the SAS data set. See “Elements for Tables” on page 105.
- The column elements define the variables for the SAS data set. See “Elements for Columns” 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 9.1  XMLMap Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
<th>Import</th>
<th>Export</th>
<th>XML</th>
<th>XMLV2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXLEMAP on page 101</td>
<td>Primary (root) enclosing element</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>NAMESPACES on page 102</td>
<td>Contains elements for defining XML namespaces</td>
<td>■</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>NS on page 103</td>
<td>Defines an XML namespace by referencing a unique URI</td>
<td>■</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>OUTPUT on page 104</td>
<td>Contains elements for exporting a SAS data set as an XML document</td>
<td>■</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>HEADING on page 104</td>
<td>Contains ATTRIBUTE elements for exporting</td>
<td>■</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>ATTRIBUTE on page 104</td>
<td>Contains file attribute information for exporting</td>
<td>■</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>TABLEREF on page 104</td>
<td>Specifies the name of the table for exporting</td>
<td>■</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>TABLE on page 105</td>
<td>Contains a data set definition</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE-PATH on page 105</td>
<td>Specifies a location path for variables</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE-END-PATH on page 107</td>
<td>Specifies a location path to stop processing</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>TABLE-DESCRIPTION on page 108</td>
<td>Specifies a SAS data set description</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>COLUMN name= on page 109</td>
<td>Specifies the variable name</td>
<td>■</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Syntax</td>
<td>Description</td>
<td>Import</td>
<td>Export</td>
<td>XML</td>
<td>XMLV2</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------</td>
<td>--------</td>
<td>--------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>COLUMN retain= on page 109</td>
<td>Determines the contents of the input buffer</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>COLUMN class= on page 109</td>
<td>Determines the type of variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>TYPE on page 110</td>
<td>Specifies the SAS data type for the variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>DATATYPE on page 110</td>
<td>Specifies the type of data being read</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>DEFAULT on page 111</td>
<td>Specifies a default value for a missing value</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>ENUM on page 111</td>
<td>Contains a list of valid values for the variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>FORMAT on page 112</td>
<td>Specifies a SAS format for the variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>INFORMAT on page 112</td>
<td>Specifies a SAS informat for the variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>DESCRIPTION on page 113</td>
<td>Specifies a description for the variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>LENGTH on page 113</td>
<td>Determines the maximum field storage length for a character variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>PATH on page 113</td>
<td>Specifies a location path for the current variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>INCREMENT-PATH on page 115</td>
<td>Specifies a location path for incrementing the accumulated value for a counter variable</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>RESET-PATH on page 116</td>
<td>Specifies a location path for resetting the accumulated value for a counter variable to zero</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>DECREMENT-PATH on page 117</td>
<td>Specifies a location path to decrement the accumulated value for the counter variable by 1</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>
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 11, “Using SAS XML Mapper to Generate and Update an XMLMap,” on page 119.

name="XMLMap"

is an optional attribute that specifies the filename of the XMLMap.

description="description"

is an optional attribute that specifies a description of the XMLMap. The description can be up to 256 characters.

Details

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

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

---

**Elements for XML Namespaces**

Define XML namespaces.

**Syntax**

```xml
NAMESPACE count="number"
NS id="number" <prefix="name">
```

**Elements**

```xml
NAMESPACE count="number" XMLV2 only
```

is an optional element that contains one or more NS elements for defining XML namespaces. Here is an example: `<NAMESPACE count="2">`.
XMLMap namespace elements enable you to import an XML document with like-named elements that are qualified with XML namespaces. In addition, XMLMap namespace elements maintain XML namespaces from the imported XML document to export an XML document from the SAS data set.

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

=count="number"

specifies the number of defined XML namespaces.

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 47

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 113.

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 47
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

```xml
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 57

**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.
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 26, 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.
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.)

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

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, the result could be concatenated data instead of separate observations, or an unexpected set of columns. See "Determining the Observation Boundary to Avoid Concatenated Data" on page 36.

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 26, here is the TABLE-PATH location path in RSS.map for the Channel data set:

```xml
<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:

```xml
<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
<?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

**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 26

**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 29

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 33

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

<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 112.

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 112.

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 112.

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

Requirement
The DATATYPE element is required.

DEFAULT
is an optional element that specifies a default value for a missing value for the variable. Use the DEFAULT element to assign a nonmissing value to missing data. For example, `<DEFAULT> single </DEFAULT>` assigns the value `single` when a missing value occurs.

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

ENUM
is an optional element that contains a list of valid values for the variable. The ENUM element can contain one or more VALUE elements to list the values. By using ENUM, values in the XML document are verified against the list of values. If a value is not valid, it is either set to MISSING (by default) or set to the value specified by the DEFAULT element. Note that a value specified for DEFAULT must be one of the ENUM values in order to be valid.

```xml
<COLUMN name="filing_status">
  
  
  
  <DEFAULT> single </DEFAULT>
  
  
  
  <ENUM>
```

Elements for Columns
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:
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:

```xml
<PATH> /LEVEL/ITEM </PATH>
<PATH> /LEVEL/ITEM[@attr="value"] </PATH>
<PATH> /LEVEL/ITEM[position()=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 44.

- 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.

  ```xml
  <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.

  ```xml
  <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.

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

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

  ```xml
  ```

**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:

```xml
<PATH> /LEVEL/ITEM/@attr </PATH>
<PATH> /LEVEL/ITEM[@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 44.

- 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.

  ```xml
  <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.

\[\text{<PATH> /NHL/CONFERENCE/DIVISION/TEAM/@founded[@abbrev="ATL"] </PATH>}\]

**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: \[\text{<PATH syntax="XPathENR">/Table/Hurricane/{1}Month</PATH>}\]. See “Including Namespace Elements in an XMLMap” on page 47.

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 44

**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.
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:

```xml
<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 33

**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.
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.

The interface consists of windows, a menu bar, and a toolbar. Using SAS XML Mapper, you can display an XML document or an XML schema, create and modify an XMLMap, and generate example SAS programs.
Using the Windows

The XML window and the XMLMap window are the two primary windows. The XML window, which is on the left, displays an XML document in a tree structure. The XMLMap window, which is on the right, displays an XMLMap in a tree structure. The map tree displays three layers: the top level is the map itself, the second tier includes tables, and the leaf nodes are columns. The detail area at the top displays information about the currently selected item, such as attributes for the table or column. The information is subdivided into tabs.

There are several source windows on the bottom of the interface, such as the XML source window, the XMLMap source window, the SAS Code Example window, and so on.

Using the Menu Bar

The menu bar provides menus to request functionality. For example, select the File menu, and then Open XML to display a browser so that you can select an XML document to open.
Using the Toolbar

The toolbar contains icons for shortcuts to several items on the menu bar. For example, the first icon from the left is the **Open an XML file** icon. Select it to display a browser so that you can select an XML document to open.

How Do I Get SAS XML Mapper?

SAS XML Mapper can be installed from the installation media for Windows and UNIX platforms, or it can be downloaded from the SAS website 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**.

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

To Start SAS XML Mapper

To start SAS XML Mapper:

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

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

Appendix 1
*Usage Not Supported for the XMLV2 Engine* .......................... 125

Appendix 2
*Example CDISC ODM Document* ........................................ 143
Appendix 1

Usage Not Supported for the XMLV2 Engine

Exporting an XML Document for Use by Oracle ........................................ 125
Exporting Numeric Values ........................................................................ 127
Exporting an XML Document in CDISC ODM Markup ....................... 132
Importing an XML Document with Numeric Values ................................. 133
Importing an XML Document Created by Microsoft Access .................. 134
Importing Concatenated XML Documents ............................................. 138
Importing a CDISC ODM Document .................................................... 140

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 MyFiles.Class to be exported to Oracle.
The following SAS program exports an XML document from the SAS data set MyFiles.Class:

libname myfiles 'SAS-library'; /*1*/
libname trans xml 'XML-document' xmltype=oracle; /*2*/

data trans.class; /*3*/
  set myfiles.class;
run;

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

2 The second LIBNAME statement assigns the libref Trans to the physical location of the file (complete pathname, filename, and file extension) that will store the exported XML document and specifies the XML engine. The engine option XMLTYPE=ORACLE produces tags that are equivalent to the Oracle 8i XML implementation.
The DATA step reads the SAS data set MyFiles.Class and writes its content in Oracle XML markup to the specified XML document.

Here is the resulting XML document.

Output A3.2  XML Document Exported from MyFiles.Class to Be Used by Oracle

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

Exporting 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.

```sas
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;
```

1. Libname to access the format.xml library.
2. Libname to access the precision.xml library with xmldouble=internal.
3. Data step to create the npi dataset with the n_pi variable.

---

3. The DATA step reads the SAS data set MyFiles.Class and writes its content in Oracle XML markup to the specified XML document.

Here is the resulting XML document.

Output A3.2  XML Document Exported from MyFiles.Class to Be Used by Oracle

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

Exporting 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.

```sas
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 f14.2;
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 f14.10;
run;

title 'Keeps the Precision'; /*7*/
proc print data=prec.rawtest;
   format n_pi f14.10;
run;

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

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

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

4 The DATA step creates the data set Format.DblTest from Work.NPI.

5 The DATA step creates the data set Prec.RawTest from Work.NPI.

6 From the data set Format.DblTest, the PRINT procedure generates the XML
document Format.XML, which contains numeric values controlled by the SAS
format. See Output A1.3 on page 129.

7 For the PRINT procedure output, a format was specified to show the precision
loss. In the output, the decimals after the second digit are zeros. See Output
A1.4 on page 130.

8 From the data set Prec.RawTest, the PRINT procedure generates the XML
document Precision.XML, which contains the stored numeric values. See Output
A1.5 on page 131.

9 For the PRINT procedure output, a format was specified to show the retained
precision. See Output A1.6 on page 132.
<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 A3.4  PRINT Procedure Output for Format.DblTest

<table>
<thead>
<tr>
<th>Obs</th>
<th>N_Pi</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.1400000000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6.2800000000</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9.4200000000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12.5700000000</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>15.7100000000</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>18.8500000000</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>21.9900000000</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>25.1300000000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>28.2700000000</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>31.4200000000</td>
<td>10</td>
</tr>
</tbody>
</table>
### Output A3.5  XML Document Precision.XML

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<TABLE>
  <RAWTEST>
    <n rawvalue="QRAAAAAAAA=">1</n>
    <n_pi rawvalue="QTJD9qiIWjA=">3.14</n_pi>
  </RAWTEST>
  <RAWTEST>
    <n rawvalue="QSAAAAAAA=">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="Qh9qe11VDF4=">31.42</n_pi>
  </RAWTEST>
</TABLE>
```

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 140 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:

```sas
filename output 'C:\myoutput.xml'; /*1*/
libname output xml xmltype=CDISCODM formatactive=yes; /*2*/
data output.AE2;
set odm.AE;
run;
```

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

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

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

The output is the same as the XML document that is shown in Appendix 2, “Example CDISC ODM Document,” on page 143.

**Importing an XML Document with Numeric Values**

This example imports the XML document Precision.XML, which was exported in “Exporting Numeric Values” on page 127. This example illustrates how you can change the behavior for importing numeric values.

The first SAS program imports the XML document using the default behavior, which retrieves parsed character data (PCDATA) from the element:

```sas
libname default xml 'C:\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 A3.7**  PRINT Procedure Output for Default.RawTest

<table>
<thead>
<tr>
<th>Obs</th>
<th>N_Pi</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.140000000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6.280000000</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9.420000000</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12.570000000</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>15.710000000</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>18.850000000</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>21.990000000</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>25.130000000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>28.270000000</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>31.420000000</td>
<td>10</td>
</tr>
</tbody>
</table>

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.)
The result of the import is SAS data set New.RawTest

Output A3.8  PRINT Procedure Output for New.RawTest

<table>
<thead>
<tr>
<th>Obs</th>
<th>N_PI</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.1415926536</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6.2831853072</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9.4247779608</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>12.663706144</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>15.7079632679</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>18.8495559215</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>21.9911487517</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>25.1327412287</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>28.2743338823</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>31.4159265359</td>
<td>10</td>
</tr>
</tbody>
</table>

Importing an XML Document Created by Microsoft Access

This example imports the following XML document, which was created from a Microsoft Access database. Because the XML document contains an embedded XML schema, you must specify the MSACCESS markup type rather than the default GENERIC type. MSACCESS obtains a variable’s attributes from the embedded schema. XMLTYPE=MSACCESS is not supported by the XMLV2 engine.

```xml
<?xml version="1.0" encoding="windows-1252" ?>
<root xmlns:xs="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:od="urn:schemas-microsoft-com:officedata">
  <xs:schema>
    <xs:element name="dataroot">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="SUPPLIERS" minOccurs="0" maxOccurs="unbounded" />
<xs: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="8" />
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:maxLength value="13" />
  </xs:restriction>
</xs:simpleType>
</xs:element>

<xs:element name="ADDRESS" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="45" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="CONTACTTITLE" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="28" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="CONTACTNAME" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="26" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="COMPANYNAME" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="38" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="SUPPLIERID" minOccurs="0">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:maxLength value="" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<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>
  </SUPPLIERS>
</dataRoot>
The following SAS program interprets the XML document as a SAS data set:

```sas
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 A3.9**  PRINT Procedure Output for Access.Suppliers

<table>
<thead>
<tr>
<th>Obs</th>
<th>CONTACTNAME</th>
<th>COMPANYNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Charlotte Smith</td>
<td>Exotic Flowers</td>
</tr>
<tr>
<td>2</td>
<td>Shelley Martin</td>
<td>New Orleans Cajun Foods</td>
</tr>
</tbody>
</table>
length, which are obtained from the embedded XML schema. Without the embedded XML schema, the results for the attributes would be default values.

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

**Output A3.10** CONTENTS Procedure Output for Access.Suppliers

---

## 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;

Example Code A3.1 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 A3.11 PRINT Procedure Output for Concat.Students

The SAS System

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

Importing a CDISC ODM Document

This example imports the XML document that is shown in Appendix 2, "Example
CDISC ODM Document," on page 143. 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*/
   Formattive=YES /*3*/
The FILENAME statement assigns the fileref Odm to the physical location of the XML document (complete pathname, filename, and file extension).

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.

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

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

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.

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.
### The SAS System

**The CONTENTS Procedure**

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>CONTENTS</th>
<th>Observations</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>33</td>
</tr>
<tr>
<td>Engine</td>
<td>SQL</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td></td>
<td>Observation Length</td>
<td>0</td>
</tr>
<tr>
<td>Last Modified</td>
<td></td>
<td>Coding Observations</td>
<td>0</td>
</tr>
<tr>
<td>Projection</td>
<td>Compressed</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Data Set Type</td>
<td>Sorted</td>
<td>NO</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Variables in Creation Order

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Internal</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>__STUDYID</td>
<td>Char</td>
<td>100</td>
<td>_STUDYID</td>
<td></td>
<td>_STUDYID</td>
</tr>
<tr>
<td>2</td>
<td>__METADATAVERSIONID</td>
<td>Char</td>
<td>100</td>
<td>_METADATAVERSIONID</td>
<td></td>
<td>_METADATAVERSIONID</td>
</tr>
<tr>
<td>3</td>
<td>__SUBJECTKEY</td>
<td>Char</td>
<td>100</td>
<td>_SUBJECTKEY</td>
<td></td>
<td>_SUBJECTKEY</td>
</tr>
<tr>
<td>4</td>
<td>__STUDYEVENTID</td>
<td>Char</td>
<td>100</td>
<td>_STUDYEVENTID</td>
<td></td>
<td>_STUDYEVENTID</td>
</tr>
<tr>
<td>5</td>
<td>__STUDYEVENTREPEATKEY</td>
<td>Char</td>
<td>100</td>
<td>_STUDYEVENTREPEATKEY</td>
<td></td>
<td>_STUDYEVENTREPEATKEY</td>
</tr>
<tr>
<td>6</td>
<td>__FORAUD</td>
<td>Char</td>
<td>100</td>
<td>_FORAUD</td>
<td></td>
<td>_FORAUD</td>
</tr>
<tr>
<td>7</td>
<td>__FORREPEATKEY</td>
<td>Char</td>
<td>100</td>
<td>_FORREPEATKEY</td>
<td></td>
<td>_FORREPEATKEY</td>
</tr>
<tr>
<td>8</td>
<td>__TECHNOLOGYID</td>
<td>Char</td>
<td>100</td>
<td>_TECHNOLOGYID</td>
<td></td>
<td>_TECHNOLOGYID</td>
</tr>
<tr>
<td>9</td>
<td>__TECHNOLOGYREPEATKEY</td>
<td>Char</td>
<td>100</td>
<td>_TECHNOLOGYREPEATKEY</td>
<td></td>
<td>_TECHNOLOGYREPEATKEY</td>
</tr>
<tr>
<td>10</td>
<td>TABAREA</td>
<td>Char</td>
<td>4</td>
<td>TABAREA</td>
<td></td>
<td>TABAREA</td>
</tr>
<tr>
<td>11</td>
<td>PID</td>
<td>Char</td>
<td>13</td>
<td>Protocol Number</td>
<td></td>
<td>Protocol Number</td>
</tr>
<tr>
<td>12</td>
<td>CTRY</td>
<td>Char</td>
<td>4</td>
<td>CTRY</td>
<td></td>
<td>Country</td>
</tr>
<tr>
<td>13</td>
<td>F_STATUS</td>
<td>Char</td>
<td>1</td>
<td>SF_STATUS</td>
<td></td>
<td>Record status, 0=missing, 1=use</td>
</tr>
<tr>
<td>14</td>
<td>LINE_NO</td>
<td>Num</td>
<td>8</td>
<td>0</td>
<td>Line Number</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>ABSTEM</td>
<td>Char</td>
<td>100</td>
<td>Connected Indication</td>
<td></td>
<td>Connected Indication</td>
</tr>
<tr>
<td>16</td>
<td>ABSCN</td>
<td>Num</td>
<td>2</td>
<td>1</td>
<td>Start Month - Enter Two Digits 01-12</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>ABSCY</td>
<td>Num</td>
<td>8</td>
<td>5</td>
<td>Start Day - Enter Two Digits 01-31</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>ABSTEY</td>
<td>Num</td>
<td>8</td>
<td>4</td>
<td>Start Year - Enter Four Digits Year</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ABSET</td>
<td>Num</td>
<td>8</td>
<td>DATE</td>
<td></td>
<td>Defined Start Case</td>
</tr>
<tr>
<td>20</td>
<td>ABENVN</td>
<td>Num</td>
<td>0</td>
<td>2</td>
<td>Stop Month - Enter Two Digits 01-12</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>ABENAV</td>
<td>Num</td>
<td>0</td>
<td>2</td>
<td>Stop Day - Enter Two Digits 01-31</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>ABENVY</td>
<td>Num</td>
<td>0</td>
<td>4</td>
<td>Stop Year - Enter Four Digits Year</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ABSETT</td>
<td>Num</td>
<td>0</td>
<td>DATE</td>
<td></td>
<td>Defined Start Date</td>
</tr>
<tr>
<td>24</td>
<td>ABERR</td>
<td>Char</td>
<td>1</td>
<td>$ABERR</td>
<td></td>
<td>Severity</td>
</tr>
<tr>
<td>25</td>
<td>ABREL</td>
<td>Char</td>
<td>1</td>
<td>$ABREL</td>
<td></td>
<td>Relationship to study drug</td>
</tr>
<tr>
<td>26</td>
<td>ABOUT</td>
<td>Char</td>
<td>1</td>
<td>$ABOUT</td>
<td></td>
<td>Outcome</td>
</tr>
<tr>
<td>27</td>
<td>ABACTTRT</td>
<td>Char</td>
<td>1</td>
<td>$ABACTTRT</td>
<td></td>
<td>Actions taken by study drug</td>
</tr>
<tr>
<td>28</td>
<td>ABCONTRT</td>
<td>Char</td>
<td>1</td>
<td>$ABCONTRT</td>
<td></td>
<td>Actions taken, other</td>
</tr>
</tbody>
</table>
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 140 and in “Exporting an XML Document in CDISC ODM Markup” on page 132.
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

<?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#"
xsi:schemaLocation="http://www.cdisc.org/ns/odm/v1.2 ODM1-2-0.xsd"
ODMVersion="1.2"
FileOID="000-00-0000"
FileType="Snapshot"
Description="Adverse events from the CTCChicago file"
AsOfDateTime="2009-03-31T14:01:41"
CreationDateTime="2009-03-31T14:01:41">
- <Study OID="STUDY.StudyOID">
- <!--             GlobalVariables is a REQUIRED section in ODM markup
-->
- <GlobalVariables>
  <StudyName>CDISC Connect-A-Thon Test Study III</StudyName>
  <StudyDescription>This file contains test data from a previous CDISC Connect-A-Thon.</StudyDescription>
  <ProtocolName>CDISC-Protocol-00-000</ProtocolName>
</GlobalVariables>
-[BasicDefinitions />
- <!--             Internal ODM markup required metadata
-->
- <MetaDataVersion OID="v1.1.0" Name="Version 1.1.0">
  <Protocol>
    <StudyEventRef StudyEventOID="SE.VISIT1" OrderNumber="1" Mandatory="Yes" />
  </Protocol>
  <StudyEventDef OID="SE.VISIT1" Name="Study Event Definition" Repeating="Yes" Type="Common">
    <FormRef FormOID="FORM.AE" OrderNumber="1" Mandatory="No" />
  </StudyEventDef>
- <FormDef OID="FORM.AE" Name="Form Definition" Repeating="Yes">
  <ItemGroupRef ItemGroupOID="IG.AE" Mandatory="No" />
</FormDef>
- <!--                Columns defined in the table
-->
- <ItemGroupDef OID="IG.AE" Repeating="Yes" SASDatasetName="AB" 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.AEENDT" 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.AECONTRT" OrderNumber="15" 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">
Example CDISC ODM Document 145
</CodeListItem>
</CodeList>

- <CodeList OID="CL.$TAREAF" SASFormatName="$TAREAF" Name=$TAREAF" DataType="text">
  <CodeListItem CodedValue="ONC">
Example CDISC ODM Document 145
</CodeListItem>
</CodeList>
- <CodeList OID="CL.$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>
Adverse Events

Some adverse events from this trial

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.AECTRRT" 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.AECTRRT" Value="0" />
  <ItemData ItemOID="ID.AECONTRT" Value="1" />
</ItemGroupData>
</FormData>
</StudyEventData>
</SubjectData>
</ClinicalData>
</ODM>