Paper AD07-2011

SAS® and Code Lists from Genericode

Larry Hoyle, IPSR, University of Kansas, Lawrence, KS

Abstract

Genericode is an OASIS committee specification, expressed as an XML schema, for representing code lists. In SAS, code lists can be handled through foreign key relationships among tables, or through Format definitions. A basic genericode code list corresponds to a table where each row corresponds to some entity and each column corresponds to some way of representing that entity. Days of the week, for example, might be represented by columns in multiple languages in a variety of abbreviations and also with different arrangements of numbers. Columns or combinations of columns, with unique values may be designated as keys. This paper describes importing a genericode XML file into SAS and defining all possible formats.

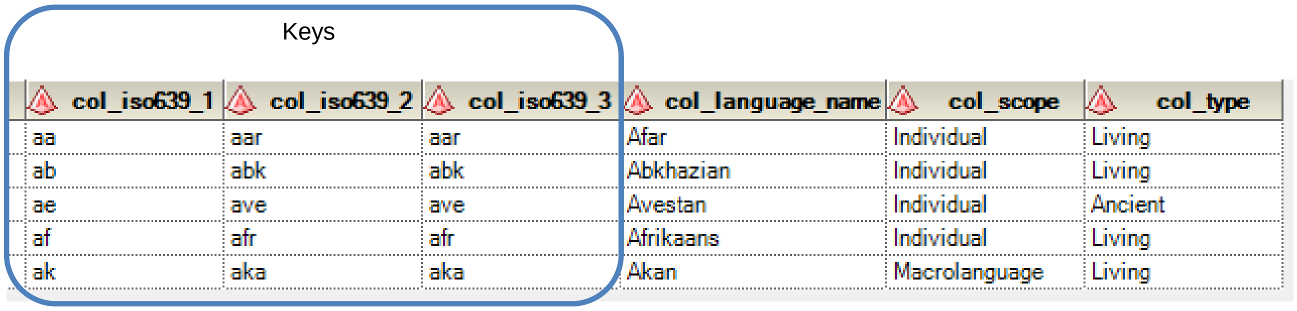
Introduction

Code lists are important in many computing contexts. Original data values may be coded for display purposes, e.g. “Male” instead of “1”. Alternative responses to a question on a survey may be presented as a selection list to control the range of responses, ensuring comparability across repeated administration or across different surveys. Keywords may also be restricted to a controlled vocabulary to make searching more efficient and accurate. See, for example, the use of controlled vocabularies in DDI (http://www.ddialliance.org/controlled-vocabularies).

A single value in a data file may have multiple possible alternative associated codes. The code “1” for gender above might have labels in multiple languages. Codes used for reports might have short forms or long forms depending on display context.

Genericode is an OAIS standard for representing code lists which recognizes the possibility of multiple possible encodings. A genericode simple code list can be conceptualized as a table, with each row corresponding to some entity and each column corresponding to a particular coding of those entities. Columns with no null values and unique entries can be denoted as keys. An analogue would be foreign key tables in a relational model.

The table in figure 1 is a representation of the code list from the example “B.3.Multiple Key Example” in the genericode specification document (http://docs.oasis-open.org/codelist/cs-genericode-1.0/doc/oasis-code-list-representation-genericode.pdf), which shows three of the ISO639 language codes for five languages.



col_iso639_1	col_iso639_2	col_iso639_3	col_language_name	col_scope	col_type
aa	aar	aar	Afar	Individual	Living
ab	abk	abk	Abkhazian	Individual	Living
ae	ave	ave	Avestan	Individual	Ancient
af	afr	afr	Afrikaans	Individual	Living
ak	aka	aka	Akan	Macrolanguage	Living

Figure 1 Genericode Example B.3 – Multiple Keys. The first 3 columns are keys.

Genericode code lists are expressed as XML documents. The schema can be found at: http://docs.oasis-open.org/codelist/cs-genericode-1.0/xsd/genericode.xsd. This paper will describe an example of importing a genericode code list into SAS and creating formats from that list. The imported data could also be used to set up foreign key constraints, but that step will not be described here.

SAS Formats and Informats

The SAS format facility allows for the association between sets of ranges of either character or numeric values and character values. There are some fundamental differences between SAS and genericode. Genericode associates multiple labels with unique key values and not ranges. Genericode also allows for multi-valued keys. A SAS format is applied to a single value.

The multilabel option in SAS allows each value-range-set to be associated with more than one label, with the first label being defined considered as the primary label and the rest as secondary labels. Some procedures, such as PROC MEANS can use the whole collection of labels. Other procedures use just the primary label.

An example from the SAS PROC FORMAT documentation is

```
value one (multilabel)
  1='ONE'
  1='UNO'
  1='UN';
```

Here the format "one." labels the value 1 with labels from three languages. The majority of SAS procedures would not be able to use the Spanish or French versions of the labels.

In order to be able to select specific versions of the labels, e.g. the French labels, a separate format can be defined for each set. In the following example English, French, and German versions are defined for a gender variable. A "long" version of the format is also defined.

```
Proc format;
 value GendE
   1='male' 2='female';
 value GendF
   1='mâle' 2='femelle';
 value GendG
   1='männlich' 2='weiblich';
 value GendL
   1='Y chromosome present'
   2='Y chromosome absent';
```

SAS has great flexibility in the way it allows formats to be associated with variables. A default format may be linked to a variable in the dataset in a DATA step, or with PROC SQL or PROC DATASETS. Formats may also be dynamically associated with variables in other procedures. Formats may also be used in functions like PUT and INPUT.

User defined formats are not stored in the SAS dataset. This allows reuse of formats (which may be stored in a format library) across multiple datasets, but also creates an opportunity for formats to be separated from datasets as when a dataset is distributed without the accompanying format definitions.

SAS PROC FORMAT offers a convenient mechanism for exporting or importing formats through the CNTLIN / CNTLOUT file format. A dataset with the proper structure can be read by PROC FORMAT which will use it to define formats. Figure 6 shows an example of a CNTLIN dataset from this project. The code below shows the result of an SQL "describe table" query on an CNTLIN / CNTLOUT dataset.

```
create table WORK.MYFMT( bufsize=16384 )
  (
   FMTNAME char(32) label='Format name',
   START char(36) label='Starting value for format',
   END char(16) label='Ending value for format',
   LABEL char(4) label='Format value label',
   MIN num label='Minimum length',
   MAX num label='Maximum length',
   DEFAULT num label='Default length',
   LENGTH num label='Format length',
   FUZZ num label='Fuzz value',
   PREFIX char(2) label='Prefix characters',
   MULT num label='Multiplier',
   FILL char(1) label='Fill character',
   NOEDIT num label='Is picture string noedit?',
   TYPE char(1) label='Type of format',
   SEXCL char(1) label='Start exclusion',
   EEXCL char(1) label='End exclusion',
   HLO char(11) label='Additional information',
   DECSEP char(1) label='Decimal separator',
   DIG3SEP char(1) label='Three-digit separator',
   DATATYPE char(8) label='Date/time/datetime?',
   LANGUAGE char(8) label='Language for date strings'
  );
```

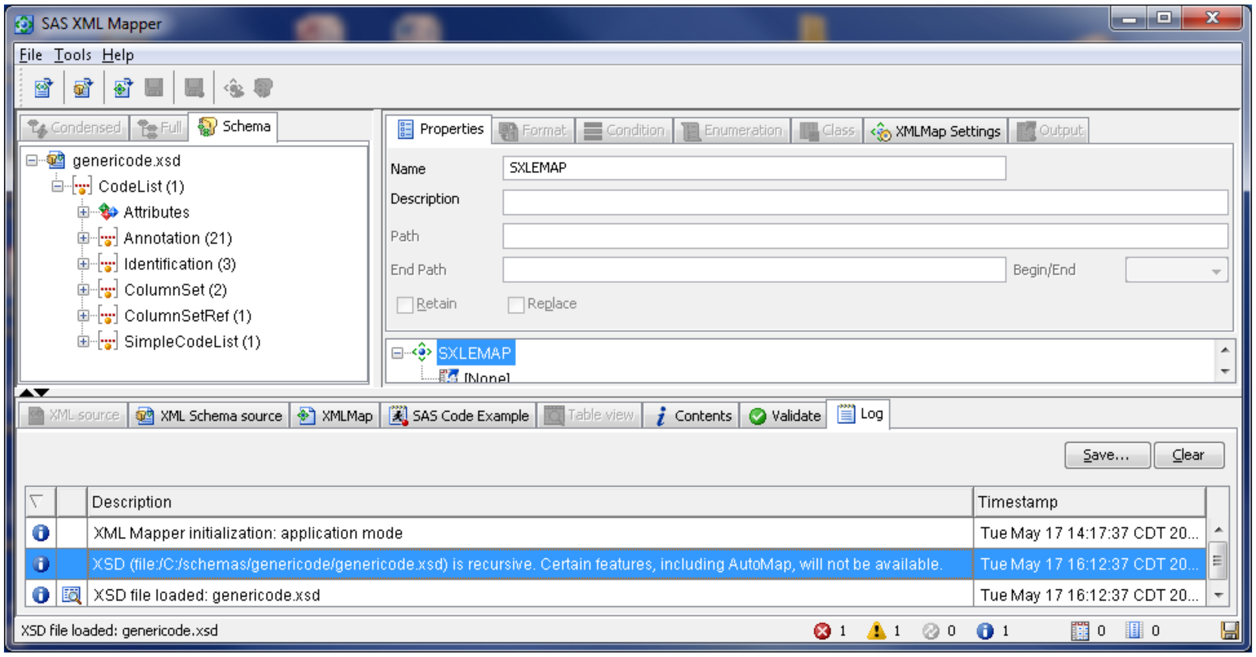
Importing Genericode

The first step in building a process to import the XML code list into SAS is to create an XMLMap file. This is most easily accomplished by using the XML Mapper application. The ideal approach would be to use the automap facility on the XML schema itself, but because the genericode schema has recursive elements this will not work. Opening the schema in XML Mapper produces this message in the log: "XSD (file:/C:/schemas/genericode/genericode.xsd) is recursive. Certain features, including AutoMap, will not be available." (see Figure 2)

Figure 2 shows the top level elements in the genericode standard. For our example the *ColumnSet* and *SimpleCodeList* elements will contain the data we need.

When the schema will not work for automap, it is possible to use an exemplar XML file which contains all of the structure that might appear in the set of XML files you might read with the XML Map file to be produced. This file need not contain actual data, it just needs all of the structure that might be encountered in any file to be read with the XML Map.

For this paper that file is the Example B.3 – Multiple Keys file mentioned above. The XML Map file produced will read any genericode file that is similar in structure – having no optional columns, with all columns containing character data. Using a sample XML file with more features would make generating a more comprehensive XML Map file possible. The associated code for joining the initial tables would have to be more complex, since when genericode columns are optional the association between values and columns in the *Value* element then has to be by reference instead of through position. When all columns are required, the references are optional so they will not be used here.



Name: SXLEMAP

Description	Timestamp
XML Mapper initialization: application mode	Tue May 17 14:17:37 CDT 20...
XSD (file:/C:/schemas/genericode/genericode.xsd) is recursive. Certain features, including AutoMap, will not be available.	Tue May 17 16:12:37 CDT 20...
XSD file loaded: genericode.xsd	Tue May 17 16:12:37 CDT 20...

```
XSD file loaded: genericode.xsd     1   1   0   1   0   0
```

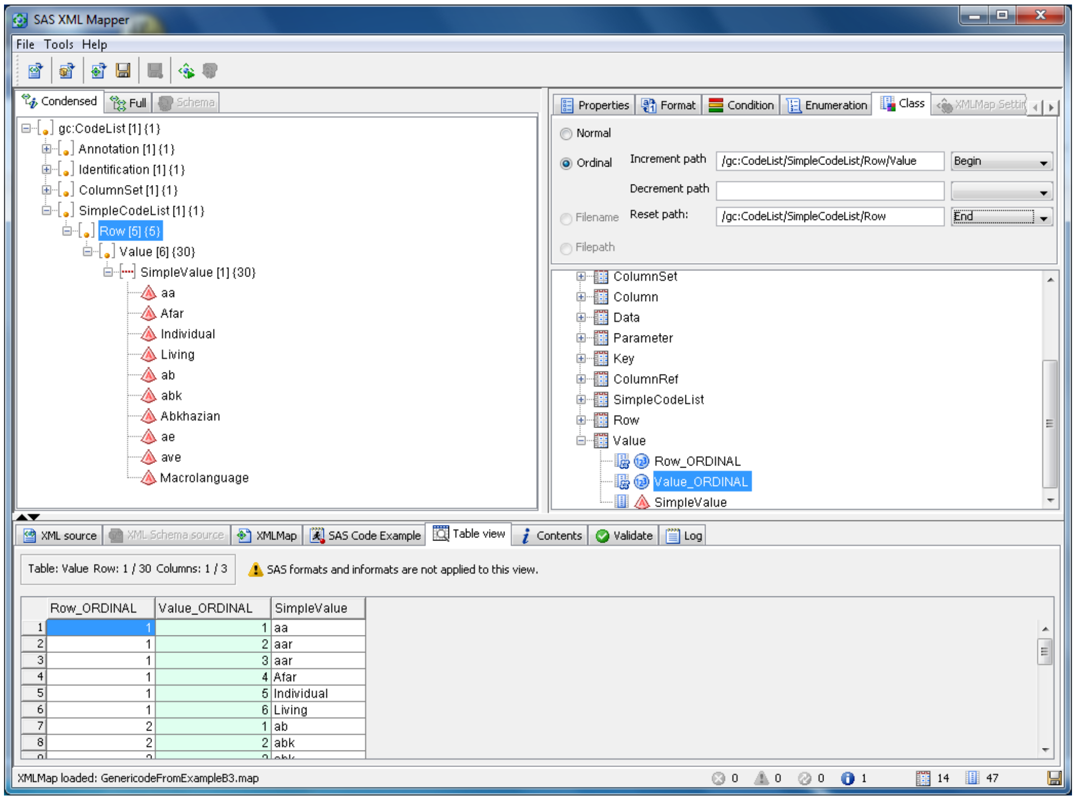
Figure 2 Opening the Genericode schema in XML Mapper

The XML Map for this project has two changes from the default automap. The first, seen in figure 3, sets the end of the element */gc:CodeList/SimpleCodeList/Row* as the reset setting for the *value_ORDINAL* variable in the *Value* table. This causes this ordinal variable to be sequentially numbered within each row, more what we would expect from column numbers – e.g. the second column is then always numbered as 2. This also allows easier checking that every row has exactly the same number of columns.

Similarly, Figure 4 shows setting */gc:CodeList/ColumnSet/Key* as the reset value for *ColumnRef*. This allows checking for multi-column keys, and excluding them from the creation of formats with code like:

```
select max(ColumnRef_ORDINAL) into :maxColumnRef_ORDINAL
 from work.keytable;

%IF &maxColumnRef_ORDINAL NE 1 %THEN %DO;
%PUT WARNING: multi-column keys cannot be made into Formats.
```



Normal [✓ Ordinal]

Increment path: /gc:CodeList/SimpleCodeList/Row/Value — Begin

Decrement path:

Reset path: /gc:CodeList/SimpleCodeList/Row — End

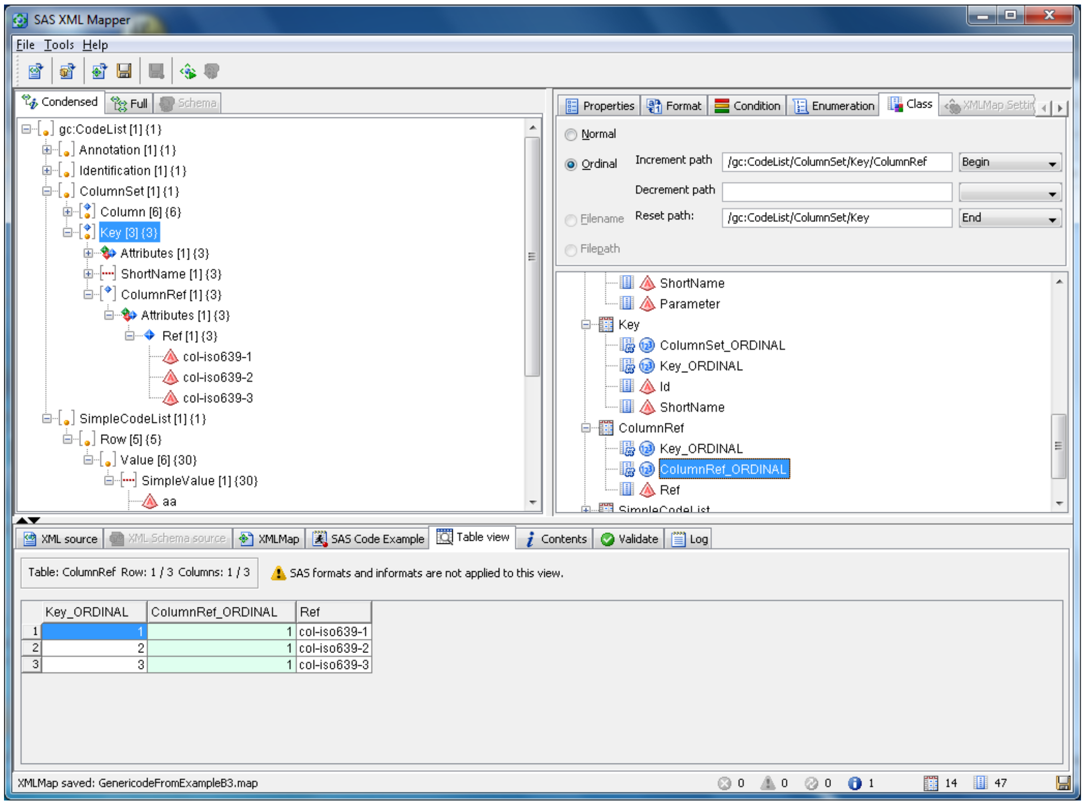
Table: Value Row: 1 / 30 Columns: 1 / 3

SAS formats and informats are not applied to this view.

	Row_ORDINAL	Value_ORDINAL	SimpleValue
1	1	1	aa
2	1	2	aar
3	1	3	aar
4	1	4	Afar
5	1	5	Individual
6	1	6	Living
7	2	1	ab
8	2	2	abk

XMLMap loaded: GenericodeFromExampleB3.map 0 0 0 1 14 47

Figure 3 Using reset to have *Value_ORDINAL* correspond to column number



Normal [✓ Ordinal]

Increment path: /gc:CodeList/ColumnSet/Key/ColumnRef — Begin

Decrement path:

Reset path: /gc:CodeList/ColumnSet/Key — End

Table: ColumnRef Row: 1 / 3 Columns: 1 / 3

SAS formats and informats are not applied to this view.

	Key_ORDINAL	ColumnRef_ORDINAL	Ref
1	1	1	col-iso639-1
2	2	1	col-iso639-2
3	3	1	col-iso639-3

XMLMap saved: GenericodeFromExampleB3.map 0 0 0 1 14 47

Figure 4 Number *ColumnRef_ORDINAL* within *Key_ORDINAL*

XML Mapper makes an XML Map and sample SAS code to create the set of tables to be extracted from an XML data file. The XML Map for files like the Example B.3 – Multiple Keys file can be seen in Appendix 2. Sample code to use that XML Map can be seen in Appendix 3. Extracting data in useful form, though, will often require additional processing.

Figure 5 shows the Enterprise Guide process flow for reading the genericode XML file, producing a CNTLIN file defining all possible SAS formats from that file, and selecting and generating one format from that file. The code nodes are arranged across the top of the diagram and links show the datasets used by each successive node. Code for each of the nodes appears in the appendices beginning with Appendix 3. The program node labeled "Import Data" contains sample code generated by XML Mapper. The process flow diagram has been arranged to show which of the imported datasets are used by the second step "checks and joins". Code for the second step can be seen in Appendix 4. This step produces several intermediate tables and one, containing everything needed to generate a CNTLIN file, used by the third step "Make Formats". The "Make Formats" code appears in Appendix 5. The CNTLIN file made in step 3 is used in step 4 to define a format and test it. Step 4 code is listed in Appendix 6

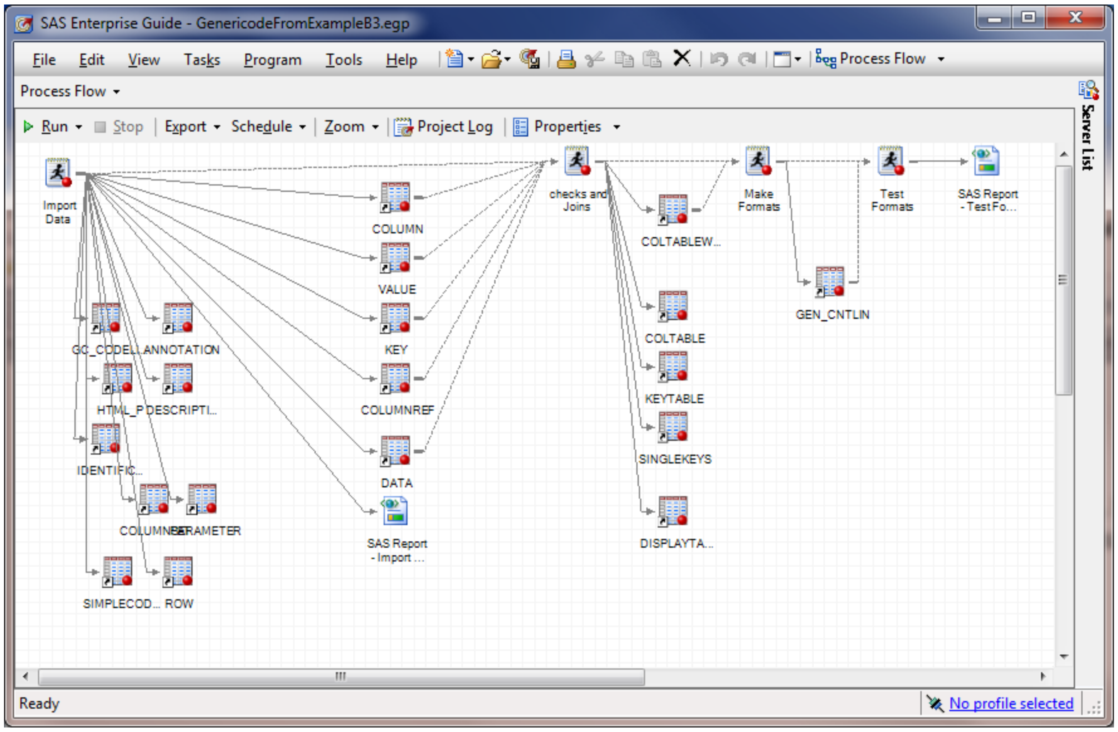
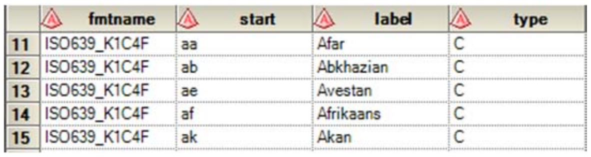


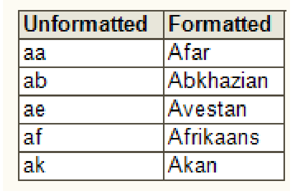
Figure 5 SAS Enterprise Guide Project

Figure 6 shows the required columns from the CNTLIN file necessary to create the format *ISO639_K1C4F*. Figure 7 shows the result of formatting column 1 (the first key) with that format. Appendix 6 contains the code used to apply PROC FORMAT to the rows of the CNTLIN table that defines that format.



	fmtname	start	label	type
11	ISO639_K1C4F	aa	Afar	C
12	ISO639_K1C4F	ab	Abkhazian	C
13	ISO639_K1C4F	ae	Avestan	C
14	ISO639_K1C4F	af	Afrikaans	C
15	ISO639_K1C4F	ak	Akan	C

Figure 6 CNTLIN data for format ISO639_K1C4F



Unformatted	Formatted
aa	Afar
ab	Abkhazian
ae	Avestan
af	Afrikaans
ak	Akan

Figure 7 Results applying format ISO639_K1C4F

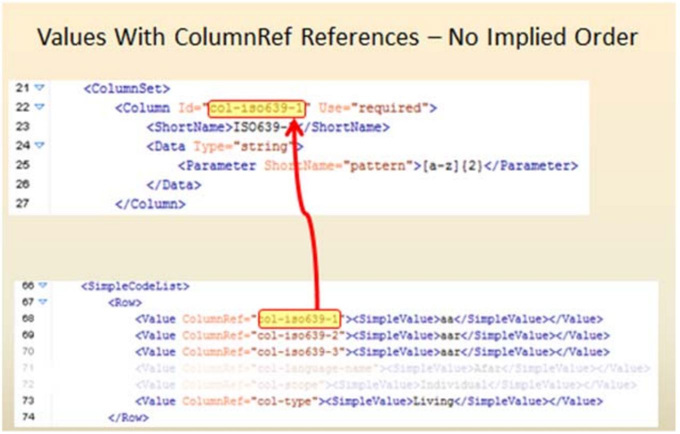
Other Issues

Reading all possible variants even a relatively simple document type like genericode can require complex logic. Schema definitions can define multiple alternative structures. Genericode, for example, allows for a *ColumnRef* attribute for a *Value* element. The documentation states:

> *"Note that the ColumnRef attribute of a Value is optional. If it is not provided, it is assumed that the column is the one which follows the column associated with the previous value in the row. If the first Value in a Row does not have a ColumnRef, it is assumed to be associated with the first column in the column set. It is an error if a row contains more than one value for the same column, or if it does not contain a value for a required column.".*

Our example relied on all values being present and in the proper order. Figure 8 shows a value for col-iso639-1 that is linked to its column by reference. A more complex XML Map could create tables containing the ColumnRef attributes linked to their associated values. The SAS code processing those tables could use that chain of references to place the data values in their proper cells of the desired ultimate table.

A *Value* element can contain either a *SimpleValue* element or a *ComplexValue* element but not both. The latter can contain XML from other namespaces. The example file (B.3) used for our XMLMap, contained only *SimpleValue* elements and had no *ColumnRef* attributes. Obviously, a program which would read any possible *Row* element would be quite a bit more complex than the one in this paper.



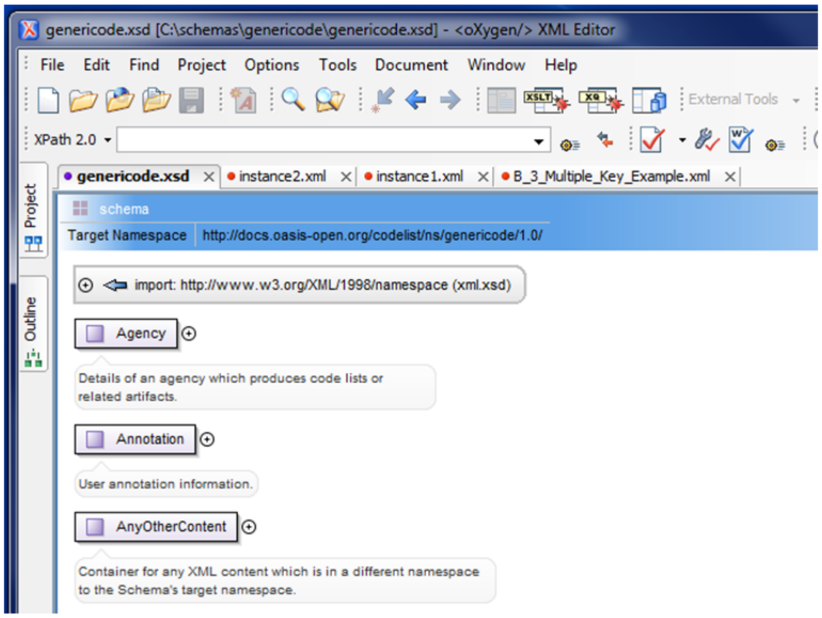
```
    <ColumnSet>
        <Column Id="col-iso639-1" Use="required">
            <ShortName>ISO639-1</ShortName>
            <Data Type="string">
                <Parameter ShortName="pattern">[a-z]{2}</Parameter>
            </Data>
        </Column>

    <SimpleCodeList>
        <Row>
            <Value ColumnRef="col-iso639-1"><SimpleValue>aa</SimpleValue></Value>
            <Value ColumnRef="col-iso639-2"><SimpleValue>aar</SimpleValue></Value>
            <Value ColumnRef="col-iso639-3"><SimpleValue>aar</SimpleValue></Value>
            <Value ColumnRef="col-language-name"><SimpleValue>Afar</SimpleValue></Value>
            <Value ColumnRef="col-scope"><SimpleValue>Individual</SimpleValue></Value>
            <Value ColumnRef="col-type"><SimpleValue>Living</SimpleValue></Value>
        </Row>
```

Figure 8 – A value linked to a column by reference

XML Mapper's inability to automap schemas with recursive elements adds a further complication. Example XML instances having all the possibilities needed may not exist. One approach to building such an example XML file might be to use an XML editor having a "Generate Sample XML Files" feature. Figure 9 shows a screenshot from and XML Editor where two sample files have been created from the schema –instance1.xml and instance2.xml. Options were chosen for the number of files, the depth of recursion and so on. It might then be possible to then cut and paste desired features into a combined XML file to quickly create a validated exemplar for XML Mapper to automap.

It might not be possible, though, to create a single exemplar file. One such situation might involve a schema that defines an exclusive required choice between two elements. A valid instance might have either of the elements, but not both. An alternative in this case would be to skip using automap and manually define an XMLMap from the schema using XML Mapper.



Target Namespace http://docs.oasis-open.org/codelist/ns/genericode/1.0/

import: http://www.w3.org/XML/1998/namespace (xml.xsd)

Agency

Details of an agency which produces code lists or related artifacts.

Annotation

User annotation information.

AnyOtherContent

Container for any XML content which is in a different namespace to the Schema's target namespace.

Figure 9 - An XML editor generates sample XML from a Schema

Suggestions for SAS

This project raises several areas in which SAS (as of 9.2) could be improved.

- The ability of XML editors to generate a set of sample XML instances from a schema suggests that the automap facility in XML Mapper could be modified to generate a mapping for recursive schemas by allowing the specification of maximum depth of recursion.
- It would be useful to have some facility in SAS datasets to indicate multiple alternative associations between variables and formats. This might allow for, for example, an easy change from the English format for all variables in a dataset to the formats for another language.
- In a broader way SAS could be improved by the ability to attach XML metadata to a dataset. In the research community there is a growing awareness of the need to document the whole data lifecycle. Data without the associated "who, what, when, where, why and how" of their creation are not ultimately interpretable.
- With a place to store structured metadata as in the previous suggestion, a tool like Enterprise Guide could be developed into a lifecycle management tool. Documenting process flow and data dependencies is a good start, but much more could be added.

References

Coates, Anthony. *genericode - The Generic Format for Code Lists* http://www.genericode.org/

DDI - Data Documentation Initiative. *Controlled Vocabularies http://www.ddialliance.org/controlled-vocabularies*

OASIS Home Page http://www.oasis-open.org

Hoyle, Larry, Joachim Wackerow with Oliver Hopt. DDI 3: Extracting Metadata From The Data Analysis Workflow DDI Working Paper Series -- Use Cases, No. 4
http://www.ddialliance.org/sites/default/files/ExtractingMetadataFromTheDataAnalysisWorkflow.pdf

Contact Information

Your comments and questions are valued and encouraged. Contact the author at:

Larry Hoyle

Institute for Policy & Social Research, University of Kansas

1541 Lilac Lane, Blake 607

Lawrence, KS 66045-3129

LarryHoyle@ku.edu

http://www.ipsr.ku.edu

Appendix 1   Genericode Example B.3 – Multiple Keys

```
<?xml version="1.0" encoding="UTF-8"?>
<gc:CodeList xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:gc="http://docs.oasis-open.org/codelist/ns/genericode/1.0/"
  xmlns:html="http://www.w3.org/1999/xhtml/">
  <Annotation>
    <Description>
      <html:p>Example of ISO639-1 language codes with ISO639-2 and ISO639-3 alternatives.</html:p>
      <html:p>See "http://www.sil.org/iso639-3/codes.asp?order=639_1&amp;letter=a".</html:p>
    </Description>
  </Annotation>
  <Identification>
    <ShortName>iso639-1</ShortName>
    <Version>1.0</Version>
    <CanonicalUri>http://www.example.com/languages/iso639-1</CanonicalUri>
    <CanonicalVersionUri>http://www.example.com/languages/iso639-1/1.0</CanonicalVersionUri>
    <Agency>
      <!-- TODO -->
    </Agency>
  </Identification>
  <ColumnSet>
    <Column Id="col-iso639-1" Use="required">
      <ShortName>ISO639-1</ShortName>
      <Data Type="string">
        <Parameter ShortName="pattern">[a-z]{2}</Parameter>
      </Data>
    </Column>
    <Column Id="col-iso639-2" Use="required">
      <ShortName>ISO639-2</ShortName>
      <Data Type="string">
        <Parameter ShortName="pattern">[a-z]{3}</Parameter>
      </Data>
    </Column>
    <Column Id="col-iso639-3" Use="required">
      <ShortName>ISO639-3</ShortName>
      <Data Type="string">
        <Parameter ShortName="pattern">[a-z]{3}</Parameter>
      </Data>
    </Column>
    <Column Id="col-language-name" Use="required">
      <ShortName>LanguageName</ShortName>
      <LongName>Language Name</LongName>
      <Data Type="string"/>
    </Column>
    <Column Id="col-scope" Use="required">
      <ShortName>Scope</ShortName>
      <Data Type="string"/>
    </Column>
    <Column Id="col-type" Use="required">
      <ShortName>Type</ShortName>
      <Data Type="string"/>
    </Column>
    <Key Id="key-iso639-1">
      <ShortName>Key-ISO639-1</ShortName>
      <ColumnRef Ref="col-iso639-1"/>
    </Key>
    <Key Id="key-iso639-2">
      <ShortName>Key-ISO639-2</ShortName>
      <ColumnRef Ref="col-iso639-2"/>
    </Key>
    <Key Id="key-iso639-3">
      <ShortName>Key-ISO639-3</ShortName>
      <ColumnRef Ref="col-iso639-3"/>
    </Key>
  </ColumnSet>
  <SimpleCodeList>
    <Row>
      <Value><SimpleValue>aa</SimpleValue></Value>
      <Value><SimpleValue>aar</SimpleValue></Value>
      <Value><SimpleValue>aar</SimpleValue></Value>
      <Value><SimpleValue>Afar</SimpleValue></Value>
      <Value><SimpleValue>Individual</SimpleValue></Value>
      <Value><SimpleValue>Living</SimpleValue></Value>
```

```
        </Row>
        <Row>
            <Value><SimpleValue>ab</SimpleValue></Value>
            <Value><SimpleValue>abk</SimpleValue></Value>
            <Value><SimpleValue>abk</SimpleValue></Value>
            <Value><SimpleValue>Abkhazian</SimpleValue></Value>
            <Value><SimpleValue>Individual</SimpleValue></Value>
            <Value><SimpleValue>Living</SimpleValue></Value>
        </Row>
        <Row>
            <Value><SimpleValue>ae</SimpleValue></Value>
            <Value><SimpleValue>ave</SimpleValue></Value>
            <Value><SimpleValue>ave</SimpleValue></Value>
            <Value><SimpleValue>Avestan</SimpleValue></Value>
            <Value><SimpleValue>Individual</SimpleValue></Value>
            <Value><SimpleValue>Ancient</SimpleValue></Value>
        </Row>
        <Row>
            <Value><SimpleValue>af</SimpleValue></Value>
            <Value><SimpleValue>afr</SimpleValue></Value>
            <Value><SimpleValue>afr</SimpleValue></Value>
            <Value><SimpleValue>Afrikaans</SimpleValue></Value>
            <Value><SimpleValue>Individual</SimpleValue></Value>
            <Value><SimpleValue>Living</SimpleValue></Value>
        </Row>
        <Row>
            <Value><SimpleValue>ak</SimpleValue></Value>
            <Value><SimpleValue>aka</SimpleValue></Value>
            <Value><SimpleValue>aka</SimpleValue></Value>
            <Value><SimpleValue>Akan</SimpleValue></Value>
            <Value><SimpleValue>Macrolanguage</SimpleValue></Value>
            <Value><SimpleValue>Living</SimpleValue></Value>
        </Row>
        <!-- ... -->
    </SimpleCodeList>
</gc:CodeList>
```

Appendix 2 XML Map used in this paper

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

<!-- ############################################################ -->
<!-- 2011-05-15T15:58:15 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XML Mapper, 902000.3.6.20090116170000_v920 -->
<!-- ############################################################ -->
<!-- ###  Validation report                              ### -->
<!-- ############################################################ -->
<!-- XMLMap validation completed successfully. -->
<!-- ############################################################ -->
<SXLEMAP name="GenericodeFromExampleB3" version="1.2">

    <!-- ############################################################ -->
    <TABLE name="gc_CodeList">
        <TABLE-DESCRIPTION>gc:CodeList</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList</TABLE-PATH>

        <COLUMN name="gc_CodeList_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="xmlns_xsi">
            <PATH syntax="XPath">/gc:CodeList/@xmlns:xsi</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>41</LENGTH>
        </COLUMN>

        <COLUMN name="xmlns_gc">
            <PATH syntax="XPath">/gc:CodeList/@xmlns:gc</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>54</LENGTH>
        </COLUMN>

        <COLUMN name="xmlns_html">
            <PATH syntax="XPath">/gc:CodeList/@xmlns:html</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>29</LENGTH>
        </COLUMN>

    </TABLE>

    <!-- ############################################################ -->
    <TABLE name="Annotation">
        <TABLE-DESCRIPTION>Annotation</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList/Annotation</TABLE-PATH>

        <COLUMN name="gc_CodeList_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="Annotation_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/Annotation</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

    </TABLE>

    <!-- ############################################################ -->
    <TABLE name="Description">
        <TABLE-DESCRIPTION>Description</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList/Annotation/Description</TABLE-PATH>

        <COLUMN name="Annotation_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/Annotation</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="Description_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/Annotation/Description</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

    </TABLE>

    <!-- ############################################################ -->
    <TABLE name="html_p">
        <TABLE-DESCRIPTION>html:p</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList/Annotation/Description/html:p</TABLE-PATH>

        <COLUMN name="Description_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/Annotation/Description</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="html_p_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/Annotation/Description/html:p</INCREMENT-
PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="html_p">
            <PATH syntax="XPath">/gc:CodeList/Annotation/Description/html:p</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>75</LENGTH>
    </COLUMN>

</TABLE>

<!-- ############################################################ -->
<TABLE name="Identification">
    <TABLE-DESCRIPTION>Identification</TABLE-DESCRIPTION>
    <TABLE-PATH syntax="XPath">/gc:CodeList/Identification</TABLE-PATH>

    <COLUMN name="gc_CodeList_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="Identification_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/Identification</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="ShortName">
        <PATH syntax="XPath">/gc:CodeList/Identification/ShortName</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>8</LENGTH>
    </COLUMN>

    <COLUMN name="Version">
        <PATH syntax="XPath">/gc:CodeList/Identification/Version</PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>double</DATATYPE>
    </COLUMN>

    <COLUMN name="CanonicalUri">
        <PATH syntax="XPath">/gc:CodeList/Identification/CanonicalUri</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>41</LENGTH>
    </COLUMN>

    <COLUMN name="CanonicalVersionUri">
        <PATH syntax="XPath">/gc:CodeList/Identification/CanonicalVersionUri</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>45</LENGTH>
    </COLUMN>

    <COLUMN name="Agency">
        <PATH syntax="XPath">/gc:CodeList/Identification/Agency</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>32</LENGTH>
    </COLUMN>

</TABLE>

<!-- ############################################################ -->
<TABLE name="ColumnSet">
    <TABLE-DESCRIPTION>ColumnSet</TABLE-DESCRIPTION>
    <TABLE-PATH syntax="XPath">/gc:CodeList/ColumnSet</TABLE-PATH>

    <COLUMN name="gc_CodeList_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="ColumnSet_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>
```

```
</TABLE>

<!-- ############################################################ -->
<TABLE name="Column">
  <TABLE-DESCRIPTION>Column</TABLE-DESCRIPTION>
  <TABLE-PATH syntax="XPath">/gc:CodeList/ColumnSet/Column</TABLE-PATH>

  <COLUMN name="ColumnSet_ORDINAL" ordinal="YES">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN name="Column_ORDINAL" ordinal="YES">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Column</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN name="Id">
    <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/@Id</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>17</LENGTH>
  </COLUMN>

  <COLUMN name="Use">
    <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/@Use</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>8</LENGTH>
  </COLUMN>

  <COLUMN name="ShortName">
    <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/ShortName</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>12</LENGTH>
  </COLUMN>

  <COLUMN name="LongName">
    <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/LongName</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>13</LENGTH>
  </COLUMN>

</TABLE>

<!-- ############################################################ -->
<TABLE name="Data">
  <TABLE-DESCRIPTION>Data</TABLE-DESCRIPTION>
  <TABLE-PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/Data</TABLE-PATH>

  <COLUMN name="Column_ORDINAL" ordinal="YES">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Column</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN name="Data_ORDINAL" ordinal="YES">
    <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Column/Data</INCREMENT-PATH>
    <TYPE>numeric</TYPE>
    <DATATYPE>integer</DATATYPE>
  </COLUMN>

  <COLUMN name="Type">
    <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/Data/@Type</PATH>
    <TYPE>character</TYPE>
    <DATATYPE>string</DATATYPE>
    <LENGTH>6</LENGTH>
  </COLUMN>

</TABLE>
```

```
    <!-- ############################################################### -->
    <TABLE name="Parameter">
        <TABLE-DESCRIPTION>Parameter</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/Data/Parameter</TABLE-PATH>

        <COLUMN name="Data_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Column/Data</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="Parameter_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Column/Data/Parameter</INCREMENT-
PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="ShortName">
            <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/Data/Parameter/@ShortName</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>7</LENGTH>
        </COLUMN>

        <COLUMN name="Parameter">
            <PATH syntax="XPath">/gc:CodeList/ColumnSet/Column/Data/Parameter</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>8</LENGTH>
        </COLUMN>

    </TABLE>

    <!-- ############################################################### -->
    <TABLE name="Key">
        <TABLE-DESCRIPTION>Key</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList/ColumnSet/Key</TABLE-PATH>

        <COLUMN name="ColumnSet_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="Key_ORDINAL" ordinal="YES">
            <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Key</INCREMENT-PATH>
            <TYPE>numeric</TYPE>
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="Id">
            <PATH syntax="XPath">/gc:CodeList/ColumnSet/Key/@Id</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>12</LENGTH>
        </COLUMN>

        <COLUMN name="ShortName">
            <PATH syntax="XPath">/gc:CodeList/ColumnSet/Key/ShortName</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>12</LENGTH>
        </COLUMN>

    </TABLE>

    <!-- ############################################################### -->
    <TABLE name="ColumnRef">
        <TABLE-DESCRIPTION>ColumnRef</TABLE-DESCRIPTION>
        <TABLE-PATH syntax="XPath">/gc:CodeList/ColumnSet/Key/ColumnRef</TABLE-PATH>

        <COLUMN name="Key_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Key</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="ColumnRef_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/ColumnSet/Key/ColumnRef</INCREMENT-PATH>
        <RESET-PATH beginend="END" syntax="XPath">/gc:CodeList/ColumnSet/Key</RESET-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="Ref">
        <PATH syntax="XPath">/gc:CodeList/ColumnSet/Key/ColumnRef/@Ref</PATH>
        <TYPE>character</TYPE>
        <DATATYPE>string</DATATYPE>
        <LENGTH>12</LENGTH>
    </COLUMN>

</TABLE>

<!-- ############################################################## -->
<TABLE name="SimpleCodeList">
    <TABLE-DESCRIPTION>SimpleCodeList</TABLE-DESCRIPTION>
    <TABLE-PATH syntax="XPath">/gc:CodeList/SimpleCodeList</TABLE-PATH>

    <COLUMN name="gc_CodeList_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="SimpleCodeList_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/SimpleCodeList</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

</TABLE>

<!-- ############################################################## -->
<TABLE name="Row">
    <TABLE-DESCRIPTION>Row</TABLE-DESCRIPTION>
    <TABLE-PATH syntax="XPath">/gc:CodeList/SimpleCodeList/Row</TABLE-PATH>

    <COLUMN name="SimpleCodeList_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/SimpleCodeList</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="Row_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/SimpleCodeList/Row</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

</TABLE>

<!-- ############################################################## -->
<TABLE name="Value">
    <TABLE-DESCRIPTION>Value</TABLE-DESCRIPTION>
    <TABLE-PATH syntax="XPath">/gc:CodeList/SimpleCodeList/Row/Value</TABLE-PATH>

    <COLUMN name="Row_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/SimpleCodeList/Row</INCREMENT-PATH>
        <TYPE>numeric</TYPE>
        <DATATYPE>integer</DATATYPE>
    </COLUMN>

    <COLUMN name="Value_ORDINAL" ordinal="YES">
        <INCREMENT-PATH beginend="BEGIN" syntax="XPath">/gc:CodeList/SimpleCodeList/Row/Value</INCREMENT-PATH>
        <RESET-PATH beginend="END" syntax="XPath">/gc:CodeList/SimpleCodeList/Row</RESET-PATH>
        <TYPE>numeric</TYPE>
```

```
            <DATATYPE>integer</DATATYPE>
        </COLUMN>

        <COLUMN name="SimpleValue">
            <PATH syntax="XPath">/gc:CodeList/SimpleCodeList/Row/Value/SimpleValue</PATH>
            <TYPE>character</TYPE>
            <DATATYPE>string</DATATYPE>
            <LENGTH>13</LENGTH>
        </COLUMN>

    </TABLE>

</SXLEMAP>
```

Appendix 3 – SAS Code Generated by XML Mapper

```
/*********************************************************************************
 *  Generated by XML Mapper, 902000.3.6.20090116170000_v920
 *********************************************************************************/

/*
 *  Environment
 */
filename  B3Multip 'C:\schemas\genericode\documentation\B_3_Multiple_Key_Example.xml';
filename  SXLEMAP 'C:\schemas\genericode\SAS\GenericodeFromExampleB3.map';
libname   B3Multip xml xmlmap=SXLEMAP access=READONLY;

/*
 *  Catalog
 */

proc datasets lib=B3Multip; run;

/*
 *  Contents
 */

proc contents data=B3Multip.gc_CodeList varnum; run;
proc contents data=B3Multip.Annotation varnum; run;
proc contents data=B3Multip.Description varnum; run;
proc contents data=B3Multip.html_p varnum; run;
proc contents data=B3Multip.Identification varnum; run;
proc contents data=B3Multip.ColumnSet varnum; run;
proc contents data=B3Multip.Column varnum; run;
proc contents data=B3Multip.Data varnum; run;
proc contents data=B3Multip.Parameter varnum; run;
proc contents data=B3Multip.Key varnum; run;
proc contents data=B3Multip.ColumnRef varnum; run;
proc contents data=B3Multip.SimpleCodeList varnum; run;
proc contents data=B3Multip.Row varnum; run;
proc contents data=B3Multip.Value varnum; run;

/*
 *  Printing
 */

title 'Table gc_CodeList';
proc print data=B3Multip.gc_CodeList(obs=10); run;
title;

title 'Table Annotation';
proc print data=B3Multip.Annotation(obs=10); run;
title;

title 'Table Description';
proc print data=B3Multip.Description(obs=10); run;
title;

title 'Table html_p';
proc print data=B3Multip.html_p(obs=10); run;
title;

title 'Table Identification';
```

```
proc print data=B3Multip.Identification(obs=10); run;
title;

title 'Table ColumnSet';
proc print data=B3Multip.ColumnSet(obs=10); run;
title;

title 'Table Column';
proc print data=B3Multip.Column(obs=10); run;
title;

title 'Table Data';
proc print data=B3Multip.Data(obs=10); run;
title;

title 'Table Parameter';
proc print data=B3Multip.Parameter(obs=10); run;
title;

title 'Table Key';
proc print data=B3Multip.Key(obs=10); run;
title;

title 'Table ColumnRef';
proc print data=B3Multip.ColumnRef(obs=10); run;
title;

title 'Table SimpleCodeList';
proc print data=B3Multip.SimpleCodeList(obs=10); run;
title;

title 'Table Row';
proc print data=B3Multip.Row(obs=10); run;
title;

title 'Table Value';
proc print data=B3Multip.Value(obs=10); run;
title;


/*
 *  Local Extraction
 */

DATA gc_CodeList; SET B3Multip.gc_CodeList; run;
DATA Annotation; SET B3Multip.Annotation; run;
DATA Description; SET B3Multip.Description; run;
DATA html_p; SET B3Multip.html_p; run;
DATA Identification; SET B3Multip.Identification; run;
DATA ColumnSet; SET B3Multip.ColumnSet; run;
DATA Column; SET B3Multip.Column; run;
DATA Data; SET B3Multip.Data; run;
DATA Parameter; SET B3Multip.Parameter; run;
DATA Key; SET B3Multip.Key; run;
DATA ColumnRef; SET B3Multip.ColumnRef; run;
DATA SimpleCodeList; SET B3Multip.SimpleCodeList; run;
DATA Row; SET B3Multip.Row; run;
DATA Value; SET B3Multip.Value; run;
```

Appendix 4 SAS Code for Checks and Joins

```
/* This code currently works only if all columns in the genericode file are required */
/*   check that this is so */

proc sql noprint;

select count(use) into :NUses
from (select distinct use from work.column) ;
quit;

%put NUses=&NUses;

%macro Joins;
```

```
%IF &NUSES EQ 1 %THEN %DO;
                        * Since all columns are required, references in the value table
                          are optional and values and names may be joined bu position
                          (the ordinal variables)
                          values for id may contain characters invalid for SAS variable names.
                          Translate then to underscores   ;
proc sql;
 create table work.ColTable as
 select Column.ColumnSet_ORDINAL,
       Value.Row_ORDINAL,
       Column.Column_ORDINAL,
       Value.Value_ORDINAL,
       Value.SimpleValue,
       Column.Id as ColId,
       Column.Use,
       Column.Shortname,
       Column.Longname,
       Data.type,
       translate(Column.Id,
                 '_',
                 '-') as sasVarName
 from work.Value,work.Column, work.Data
 where Value.Value_ORDINAL=Column.Column_ORDINAL AND
      Column.Column_ORDINAL=Data.Column_ORDINAL
 order by ColumnSet_ORDINAL, Row_ORDINAL, Value_ORDINAL;
                       * the original table of keys ;
 create table work.keytable as
 select ColumnSet_ORDINAL,
       Key.Key_ORDINAL,
                                   ColumnRef_ORDINAL,
                                   Key.Id as KeyId,
                                   Key.ShortName as KeyShortName,
                                   ColumnRef.Ref as KeyRef
 from work.Key,work.ColumnRef
 where Key.Key_ORDINAL=ColumnRef.Key_ORDINAL
 order by ColumnSet_ORDINAL, Key_ORDINAL;
proc sql noprint;
 select max(ColumnRef_ORDINAL) into :maxColumnRef_ORDINAL
 from work.keytable;

%IF &maxColumnRef_ORDINAL NE 1 %THEN %DO;
%PUT WARNING: multi-column keys cannot be made into Formats.
Title 'WARNING: These multi-column keys cannot be made into Formats';
proc sql;
create table work.multikeys as select distinct Key.Key_ORDINAL,Id from work.Keytable where ColumnRef_ORDINAL>1;
create table work.singlekeys as select * from keytable where KeyId not in (select KeyId from work.multikeys);
select * from work.multikeys;
%END;
%ELSE %DO;
                       * here we want just the keys based on a single column  ;
create table work.singlekeys as select * from work.keytable;
%END;

create table work.colTablewithKeys as
select ColTable.*,
      singleKeys.Key_ORDINAL,
      singleKeys.KeyId,
      singleKeys.KeyShortName,
      singleKeys.KeyRef
from work.ColTable left join work.singleKeys
  on  singleKeys.KeyRef=ColTable.ColId
order by ColumnSet_ORDINAL, Row_ORDINAL, Column_ORDINAL;

%END;
%ELSE %DO;
                   * Optional columns were found. Tables will have to be joined by references not position.  ;
%put WARNING: Some columns are not required - Cannot process this genericode file;
%END;

Proc Transpose data=work.coltablewithkeys out=work.DisplayTable;
   by Row_ORDINAL;
   ID sasVarName;
   IDLAbel ShortName;
   Var SimpleValue;
```

```
run;

%mend Joins;
%Joins
```

Appendix 5 SAS Code for Make Formats

```
/*      create all possible  formats in a CNTLIN file  */
%LET FMTPREFIX=ISO639;
proc sql;
create table work.Gen_Cntlin as
select  cats("&FMTPREFIX._K" , put(keyCol,6.0) , 'C' ,   put(Column_ORDINAL, 6.0) , 'F') as fmtname,
        start,
        label,
        'C' as type,
        KeyCol,
        Column_Ordinal,
        KeyRow

from
(select SimpleValue as label, Row_Ordinal, Column_ORDINAL from ColtableWithKeys),
(select SimpleValue as start, Row_ORDINAL as KeyRow, Column_ORDINAL as KeyCol
 from ColtableWithKeys where key_Ordinal is not null)
where Row_Ordinal = KeyRow and Column_ORDINAL NE KeyCol
order by KeyCol, Column_Ordinal, KeyRow
;
```

Appendix 6 SAS Code for Test Formats

```
proc format CNTLIN=work.GEN_Cntlin(where=(FMTNAME='ISO639_K1C4F'));
run;

proc sql;
select start  label='Unformatted', put(start,$ISO639_K1C4F.) as Formatted
from work.Gen_Cntlin
where keycol=1 and Column_Ordinal=4;
```