Paper 099-30

Using XML Mapper and XMLMAP to Read Data Documented by Data Documentation Initiative (DDI) Files

Larry Hoyle Policy Research Institute, The University of Kansas, Lawrence, KS

ABSTRACT

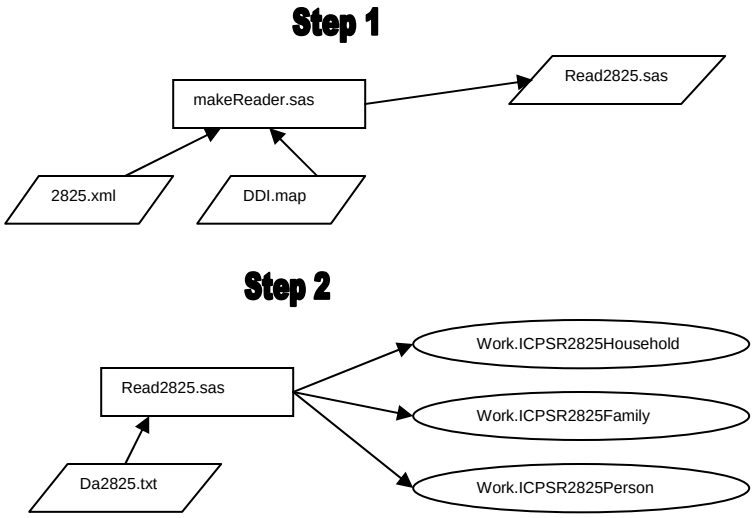
Data Documentation Initiative (DDI) files are XML "codebooks" for social science data. A DDI file can describe itself, the conditions under which the data were collected, the data file format, and the variables in the file. SAS XMLMAP provides the capability for a SAS® program to read these metadata from the XML file into a collection of SAS datasets. A SAS program uses this capability to write another SAS program that can read the data documented by the DDI file. This paper also describes how the XML Mapper tool was useful in developing the XMLMAP for this project.

OVERVIEW

The SAS program described in this paper "makeReader.sas" reads metadata from a DDI file using a SAS XMLMAP file and writes a new SAS program. The new program can read the data file described in the DDI file and makes one or more SAS datasets.

The figures at right illustrate this process for the ICPSR study 2825 file. The DDI file "2825.xml" describes the data file "Da2825.txt". The SAS program "makeReader.sas" reads 2825.xml and writes a SAS program "Read2825.sas".

When Read2825.sas is run, it reads DA2825.txt and creates three SAS datasets "work.ICPSR2825Household" "work.ICPSR2825Family" and "work.ICPSR2825Person"



WHAT IS DDI?

The front page for the Data Documentation Initiative (DDI), http://www.icpsr.umich.edu/DDI/, describes DDI as "an international effort to establish a standard for technical documentation describing social science data". DDI provides a means to structure metadata as an XML document. The metadata can include the information necessary to read the raw data file as well as extensive information about the metadata file itself, the context of the data file (e.g. the study), the data file, the variables within the file, the questions associated with the variables and more. Some DDI elements map to the "Dublin Core" resource description standard allowing for cataloging of the data and documentation. For more about Dublin Core see http://dublincore.org/about/.

DDI XML format is defined by a Document Type Definition (DTD) available at http://www.icpsr.umich.edu/DDI/users/dtd/index.html. In that DTD, the root element <**CodeBook**> of a DDI file is defined as

```
<!ELEMENT codeBook   (docDscr*
                     , stdyDscr+
                     , fileDscr*
                     , dataDscr*
                     , otherMat*) >
```

The elements of the DDI file within the root element "codebook" are:

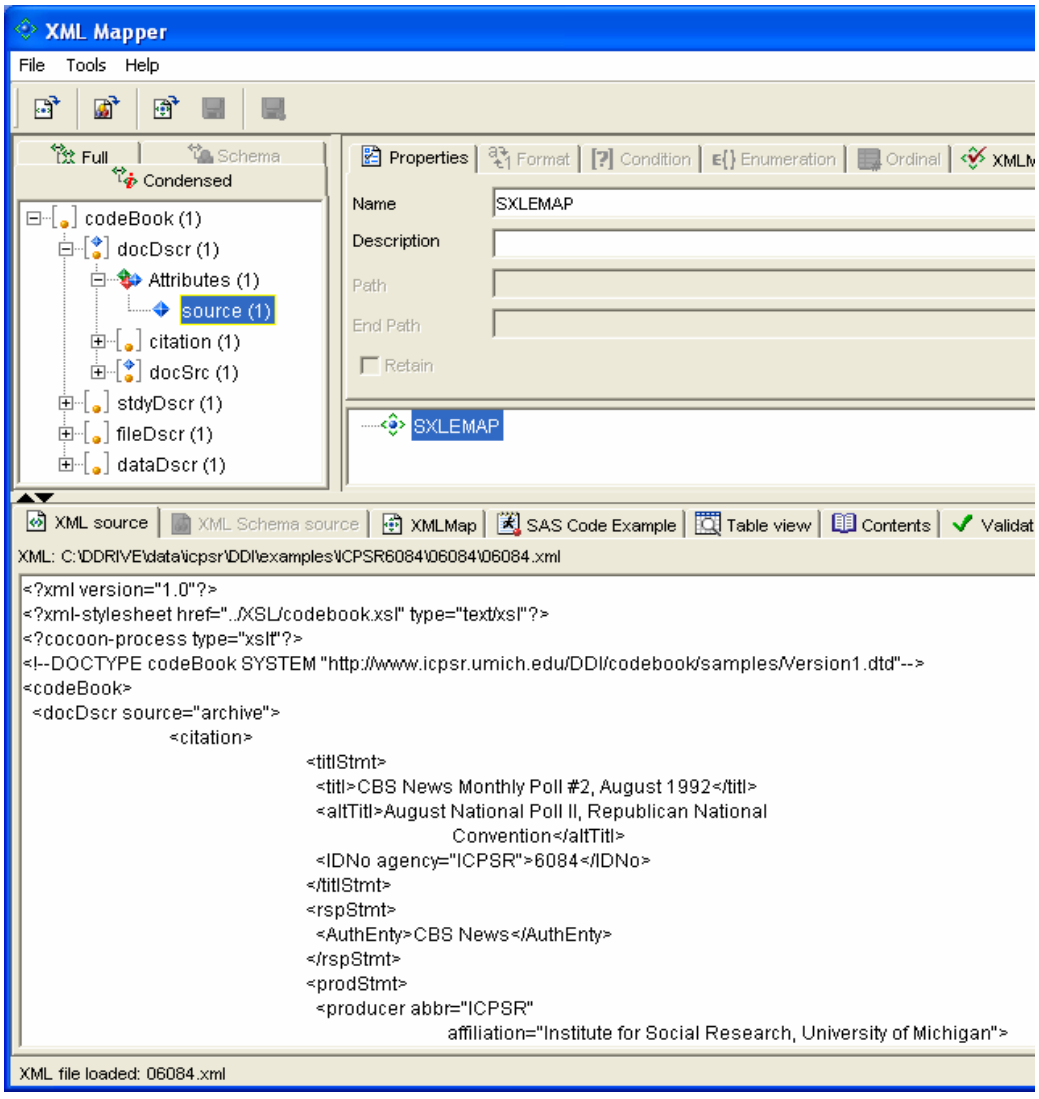
- *docDscr* – describes the DDI document itself (this element is optional)
  - can include a citation, status, source, notes and more
- *stdyDsc* - describes the study (this element is required)
  - can include study information, method, terms of use and more. **Must** include a citation.
- *fileDscr* - describes the data file (this element is optional)
  - can include a URI for locating the data file; information about content, structure, format, number of records, record length and more.
- *dataDscr* - describes the variables within the data file (this element is optional)
  - can include information about variable groups, ncube groups as well as information about variables – name, weight, question, location(start position, end position, width, record segment) and more
- *otherMat* - describes other material (this element is optional)
  - could include questionnaires, SAS code to read the data, maps etc.

The only required element in a DDI file is a title within the <stdyDscr> element. The following is a valid DDI file, (but would not be of much help to anyone).

```
<?xml version="1.0"?>
<codeBook>
 <stdyDscr>
  <citation>
   <titlStmt>
    <titl>Valid but Useless Metadata</titl>
   </titlStmt>
  </citation>
 </stdyDscr>
</codeBook>
```

A SAMPLE DDI FILE

This paper will use the codebook for ICPSR study 6084 – "CBS News Monthly Poll #2, August 1992" as one of the example DDI files to be read. DDI example files are available at: http://www.icpsr.umich.edu/DDI/samples/index.html. The SAS XML Mapper can be used to view the DDI file. The pane with the "Condensed" tab allows one to drill down into the hierarchy, expanding and contracting elements by clicking on the ⊞ or ⊟ icons. The pane labeled "XML source" shows the full contents of the XML file. In the figure below, the <docDscr source="Archive"></docDscr> element has been expanded to show that it has one attribute and contains a <citation></citation> element and a <docSrc></docSrc> element. The expanded attribute list reveals the attribute of the <docDscr> element to be its "source". Several of the elements within the <citation></citation> can be seen, some of which, in turn, contain other elements.



WHAT IS SAS XMLMAP?

In order to retrieve data from a hierarchical XML file into SAS datasets there has to be a way to specify a "mapping" between the hierarchy and one or more rectangular tables. The SAS XMLMap file is that mapping.

The SAS XMLMap file is an XML file that describes which elements of an XML target file define rows and which elements define columns for the extracted dataset. In the example below, a SAS dataset named *fileInfo* is created with one row for each <*fileTxt*></*fileTxt*> element that is nested in a <*fileDscr*></*fileDscr*> element. The dataset has two columns, one taken from the <*filename*></*filename*> element and one from the <*caseQnty*> </*caseQnty*> element.

The critical elements of the XMLMap file below are the tags like:

```
<TABLE name="fileInfo">                                  names the resultant dataset
   <TABLE-PATH>/codeBook/fileDscr/fileTxt</TABLE-PATH>   identifies the element that defines rows

   <COLUMN name="fileName">                              names the column "fileName"
     <PATH>/codeBook/fileDscr/fileTxt/fileName</PATH>    identifies the element that contains the column data
```

```
<!-- 2003-05-18T12:31:09.041 -->
<!-- SAS XML Libname Engine Map -->
<!-- Generated by XMLAtlas, Version 9.0.1 -->

<SXLEMAP version="1.1" name="DDIMAP">
<TABLE name="fileInfo">
   <TABLE-PATH>/codeBook/fileDscr/fileTxt</TABLE-PATH>

   <COLUMN name="fileName">
     <PATH>/codeBook/fileDscr/fileTxt/fileName</PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>42</LENGTH>
   </COLUMN>

   <COLUMN name="caseQnty">
     <PATH>/codeBook/fileDscr/fileTxt/dimensns/caseQnty</PATH>
     <TYPE>character</TYPE>
     <DATATYPE>STRING</DATATYPE>
     <LENGTH>5</LENGTH>
   </COLUMN>

 </TABLE>

</SXLEMAP>
```

The following SAS program prints the dataset described by the SAS XMLMap file above.

```
/* read a two column table from DDI file for ICPSR 6084 */
filename  DDIfile 'D:\projects\sugs\sugi29\xmlCodebooks\06084noStylesheet.xml';
filename  SXLEMAP 'D:\projects\sugs\sugi29\sascode\fileInfo.map';
libname   DDIfile xml xmlmap=SXLEMAP access=READONLY;

proc print data=DDIfile.fileInfo;
run;
```

The resulting output is:

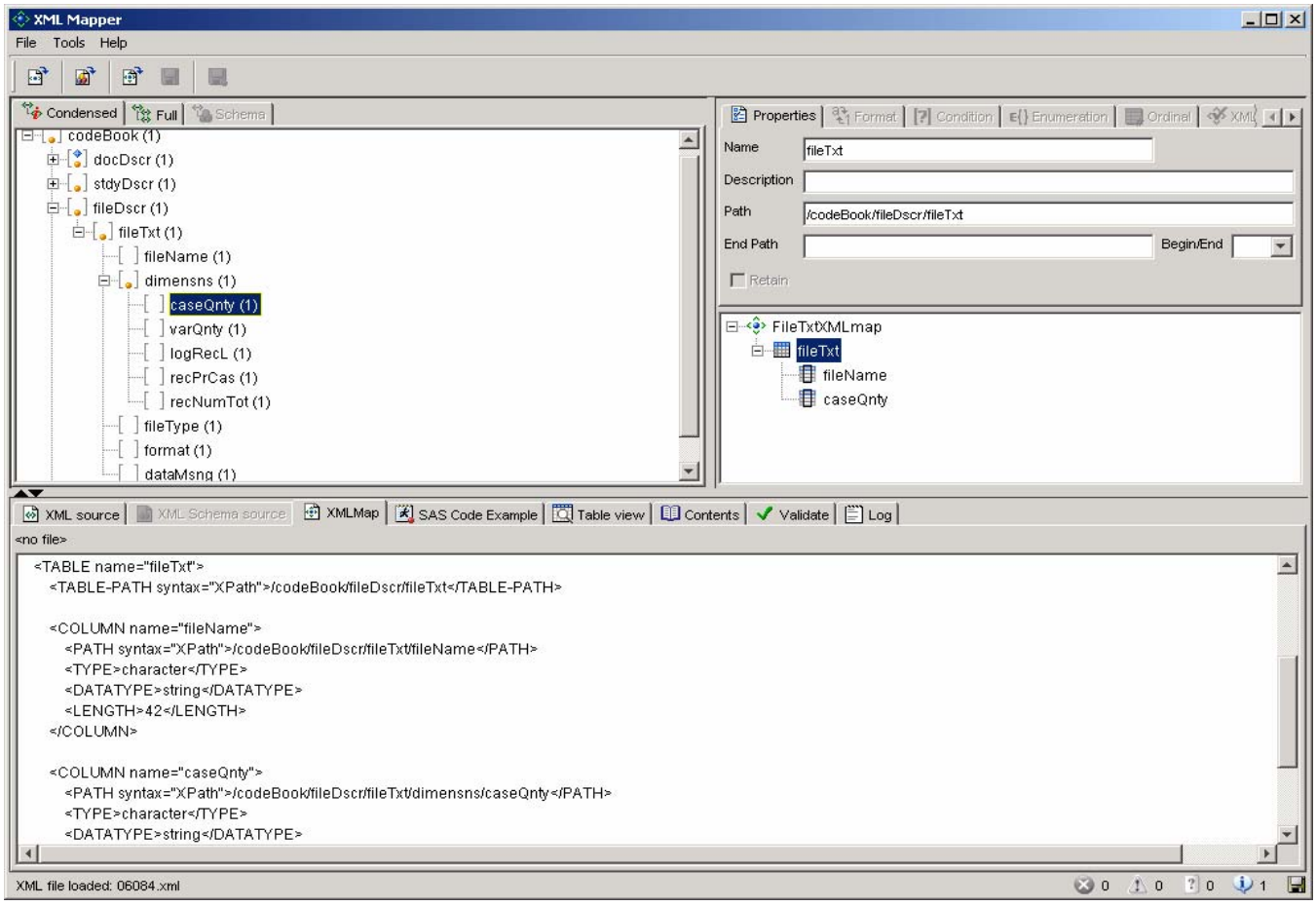
Obs	fileName	caseQnty
1	DS1: CBS News Monthly Poll #2, August 1992	1,546

MAKING AN XMLMAP FILE WITH XML MAPPER

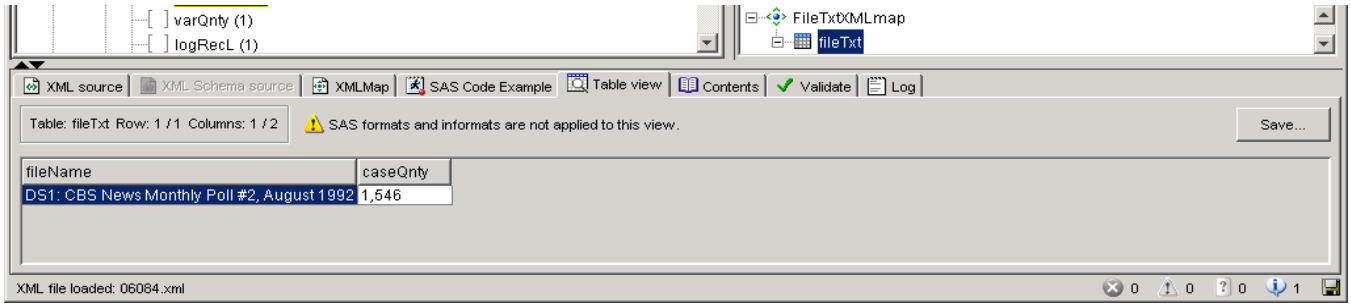
SAS XML Mapper in SAS 9.1 is a graphical user interface for creating SAS XMLMap files. It allows drag and drop creation of the XMLMap file and automatically creates the SAS file to reference it. Using Mapper was an easy way to create an initial XMLMap file to read DDI files. Here is a Mapper view of the files we've been working with so far.

XML elements can be dragged from the upper left pane to the upper right pane, creating the design of the tables to be extracted. The bottom pane shows the XMLMap file that Mapper generates. The sample SAS code that will access that XMLMap file could be viewed by clicking the "SAS Code Example" tab.

The upper right pane shows that the resultant dataset will be named "fileinfo". There will be one row for each /codeBook/fileDscr/fileTxt element. That is, each <fileTxt ></fileTxt > nested within a <fileDscr ></fileDscr > nested within a <codeBook ></codeBook >. There will be a column for **one** /codeBook/fileDscr/fileTxt/filename and for **one** /codeBook/fileDscr/fileTxt/dimensns/caseQnty within that <fileTxt ></fileTxt >. Note that if there could be more than one <filename></filename> within a given <fileTxt ></fileTxt > element, to get **every** one the rows would need to be defined at the /codeBook/fileDscr/fileTxt/filename level.



Starting with SAS 9.13, XML Mapper will also display the actual data table to be read using the XML Map file.



EXTRACTING SAS DATASETS FROM THE HIERARCHICAL DDI FILE

Only a small subset of the elements in the DDI file are actually **needed** to write a SAS program to read the data file, but it is useful to read additional elements to document the code. In order to allow for repetition of values, a number of SAS datasets need to be created from the DDI file. The current iteration of the SAS program that reads the DDI file (makeReader.sas) creates over 40 tables with a total of over 200 columns. Some of those table and column paths are listed below. The paths with the '@' like "@agency" refer to attributes of XML elements.

A table with title(s) of the documentation

```
<TABLE-PATH>/codeBook/docDscr/citation/titlStmt/titl</TABLE-PATH>
      <PATH>/codeBook/docDscr/citation/titlStmt/titl</PATH>
```

A table with title(s), subtitle(s), alternate title(s) and ID number(s) of the study

```
<TABLE-PATH>/codeBook/stdyDscr/citation/titlStmt</TABLE-PATH>
      <PATH>/codeBook/stdyDscr/citation/titlStmt/titl</PATH>
      <PATH>/codeBook/stdyDscr/citation/titlStmt/subTitl</PATH>
      <PATH>/codeBook/stdyDscr/citation/titlStmt/altTitl</PATH>
      <PATH>/codeBook/stdyDscr/citation/titlStmt/IDNo</PATH>
      <PATH>/codeBook/stdyDscr/citation/titlStmt/IDNo/@agency</PATH>
```

A table with 77 columns describing each variable in the data file (the first 5 columns are shown here)

```
<TABLE-PATH>/codeBook/dataDscr/var</TABLE-PATH>
      <PATH>/codeBook/dataDscr/var/@ID</PATH>
      <PATH>/codeBook/dataDscr/var/@name</PATH>
      <PATH>/codeBook/dataDscr/var/@wgt</PATH>
      <PATH>/codeBook/dataDscr/var/@wgt-var</PATH>
      <PATH>/codeBook/dataDscr/var/@weight</PATH>
```

A table with labels and missing indication for values

```
<TABLE-PATH>/codeBook/dataDscr/var/catgry</TABLE-PATH>
      <PATH>/codeBook/dataDscr/var/catgry/@ID</PATH>
      <PATH>/codeBook/dataDscr/var/@ID</PATH>
      <PATH>/codeBook/dataDscr/var/@name</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/catValu</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/labl</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/labl/@level</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/txt</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/txt/@level</PATH>
      <PATH>/codeBook/dataDscr/var/catStat</PATH>
      <PATH>/codeBook/dataDscr/var/catStat/@URI</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/labl/Link/@refs</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/@missing</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/@missType</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/@country</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/@sdatarefs</PATH>
      <PATH>/codeBook/dataDscr/var/catgry/@excls</PATH>
```

A table that has information about record groups (for hierarchical files)

```
<TABLE-PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp</TABLE-PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@ID</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@recGrp</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@rectype</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@keyvar</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@rtypeloc</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@rtypewidth</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@rtypevtype</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/@recidvar</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/labl</PATH>
      <PATH>/codeBook/fileDscr/fileTxt//fileStrcrecGrp/recDimnsn/varQnty</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/recDimnsn/caseQnty</PATH>
      <PATH>/codeBook/fileDscr/fileTxt/fileStrc/recGrp/recDimnsn/logRecL</PATH>
```

A SAS DATASTEP WRITES A SAS DATASTEP

These datasets contain the information necessary to write a SAS program to read the data file documented by the DDI file. The SAS program to be output consists of a large comment at the beginning, containing metadata not needed for the mechanics of reading the file. Here is the beginning of that comment for the ICPSR 6084 data file:

```
/*  SAS program to read ICPSR  6084
CBS News Monthly Poll #2, August 1992
August National Poll II, Republican National Convention

File Name:
     DS1: CBS News Monthly Poll #2, August 1992
Number of Cases:
     1,546
Number of Variables:
     70
Logical Record Length
     80
Records Per Case:
     3
```

The content of these elements needs to be edited to ensure that there won't be a "*****/" printed in the middle of the comment. The DDI file for ICPSR 6084 also contained many tab characters which are converted to spaces for the comment.
The SAS macro used to edit the elements was:

```
%macro putInCmnt(v=name, ve=vEdited, h=heading);
    &ve=compbl(tranwrd(translate(&v,' ','09'x),'*/','*_/'));
    if &ve ne ' ' then put / "&h"   / '     '  &ve;
%mend putInCmnt;
```

The code to print the part of the beginning comment shown above was:

```
set DDIfile.titlstmt;
    put '/*  SAS program to read ' agency ' ' IDNo ;
    titl= compbl(tranwrd(translate(titl,' ','09'x),'*/','*_/'));
    put titl;
    altTitl=compbl(tranwrd(translate(altTitl,' ','09'x),'*/','*_/'));
    put altTitl;
    put //;
    set DDIfile.Filetxt;
    %putInCmnt(v=fileName, ve=vEdited,  h=File Name:);
    %putInCmnt(v=caseQnty, ve=vEdited,  h=Number of Cases:);
    %putInCmnt(v=varQnty, ve=vEdited,  h=Number of Variables:);
    %putInCmnt(v=logRecl, ve=vEdited,  h=Logical Record Length);
    %putInCmnt(v=recPrCas, ve=vEdited,  h=Records Per Case:);
```

Since SAS format names are still restricted in length, the expedient thing to do was to construct format names like "V00015f" and then place a comment with the corresponding variable name. SAS missing values must also be assigned labels. The DDI elements *catValu* and *labl* were used to write value statements for a PROC FORMAT as in this example.

```
proc format;
value V00015f      /* format for variable Q1 */
1 ='Yes'
2 ='No'
.B ='DK/NA'
9 ='DK/NA'
```

In the example above the value "9" represents missing and will be replaced with the SAS missing value ".B" in the datastep.

Writing the DATA step to read the file described by the DDI file involves several SAS “DATA ._null_;” steps. One, shown below, writes the input statement, using the FILETXT, TITLSTMT, and a revised DATADSCRVAR datasets. Another writes a sequence of IF statements to substitute SAS unique missing values for the original missing values. Another writes variable label statements. A final “DATA _null_” statement assigns formats to variables.

```
data _null_;
                        /* begin writing datastep to read the variables */
  set DATADSCRVAR end=last;
  file reader lrecl=1024 mod;
  length ftype $ 3;
  if _n_=1 then do;
     set DDIfile.Filetxt;
     set DDIfile.titlstmt;
    put ///;
                                       /* remove unsafe characters */
    agency=translate(agency,'_______','''';/*&%');
    IDNo=translate(IDNo,'_______','''';/*&%');
    logrecl=translate(logrecl,'_______','''';/*&%');
    RecSegNo=translate(RecSegNo,'_______','''';/*&%');
    ftype=translate(ftype,'_______','''';/*&%');
    StartPos=translate(StartPos,'_______','''';/*&%');
    EndPos=translate(EndPos,'_______','''';/*&%');

    put 'data ' agency +(-1) IDNo ';';
    put "infile  '&targetPath' LRECL=" logrecl    " PAD;";
    put "input ";
  end;
                                     /*  position of each variable */
  if RecSegNo ne ' ' and StartPos ne ' ' and EndPos ne ' ' then do;
    if upcase(formatType) = 'NUMERIC' then ftype = '   ';
    else ftype = ' $ ';
    put "#" RecSegNo safename ftype StartPos +(-1) "-" EndPos ;
  end;

  if last then do;
    put ";";
  end;
run;
```

Here is an abbreviated copy of the SAS program (readICPSR6084.sas) written by the DDI processing SAS program (makeReader.sas).

```
/*  SAS program to read ICPSR  6084
CBS News Monthly Poll #2, August 1992
August National Poll II, Republican National Convention
File Name:
     DS1: CBS News Monthly Poll #2, August 1992
Number of Cases:
     1,546
Number of Variables:
     70
Logical Record Length
     80
Records Per Case:
     3
*/
proc format;
value V00015f      /* format for variable Q1 */
1 ='Yes'
2 ='No'
.B ='DK/NA'
9 ='DK/NA'
;
value V00067f      /* format for variable Q33a */
1 ='Under'
2 ='Over'
2 ='Won''t specify/Refused'
.I ='Won''t specify/Refused'
;
data ICPSR6084 ;
infile  'D:\data\icpsr\data\6084\da06084.txt' LRECL=80  PAD;
input
#1 cardno   1-1
#1 respno   2-6
#1 Q1   30-30
#2 cardno   1-1
#2 respno   2-6
#2 Q37   57-64
#3 cardno   1-1
#3 respno   2-6
#3 weight $ 49-54;
/* replace missing data with unique SAS missing values */
  if Q1  = 9  then Q1  = .B ;
  label Q1 ='Some people are registered to vote and others are not.
Are you registered to vote in the precinct or election district where you live,
or aren''t you? ';
/*  This section will associate formats with each variable that has labeled
categories */
/*  you may want to comment it out. */
  format Q1  V00015f.;
run;
```

REFINING THE XMLMAP FILE MANUALLY

Using the XML Mapper was a great way to get started, but because DDI files have only the <titl></titl> element required, no one DDI file will have all of the elements that should be read in order to make a general tool. This means examining the official definition of the DDI file, in this case a document type definition file (DTD) and hand editing the XMLMap file.

For document types defined by an XML schema, SAS XML Mapper (for SAS version 9.13 and above) can open the schema file and XMLMap files can be created directly from the schema. This is the ideal way to proceed when possible. For some document types, however, there may only be a DTD, or, as in the case of DDI files the schema may be complex enough to be beyond XML Mappper's capacity. In either of these cases the XMLMap file can be edited manually outside of XMLMapper.

RECURSION

The DTD for DDI files allows recursion of several elements. A record group may contain other record groups, for example. This makes creating a truly general tool a challenge. The current version of this SAS program should handle most rectangular files and hierarchical files where the record group identifier is in the same position on each type of record. For hierarchical files, the program will generate code that creates a separate dataset for each type of record.

VARIATIONS IN DATA FILE LAYOUTS

DDI files are capable of describing several types of data files including files with data in fixed columns, files with delimiter separated fields, and files with multiple record types. Each of these file types requires a different syntax in the SAS code that reads it.

FILES WITH MULTIPLE RECORD TYPES

A data file with multiple record types might be read in several ways. It might represent a hierarchy – say, for example, items within purchase orders, or persons within households. It might just represent several independent tables. The indicator for record type could be quite complex, being in some combination of different columns in the data file. For the current version of makeReader.sas, the DDI processing program produces SAS code that looks for a record type indicator in a fixed location and writes all of the records for a given record type to their own SAS dataset. ICPSR study 2825 is an example of such a file. It contains records for individuals, families and households, with different variables for each record type. The generated SAS program creates 3 datasets from this file. Part of the generated code (dealing with family records) is shown below.

```
data ICPSR2825FAMILY  (keep =  FRECORD  FH_SEQ FFPOS FKIND    …
     ICPSR2825HOUSEHOLD  (keep =  HFOODMO HRECORD H_SEQ HHPOS HUNITS      …
     ICPSR2825PERSON  (keep = FILLER PRECORD PH_SEQ PPPOS A_LINENO A_PARENT
                              A_EXPRRP   …
o
o
o

infile  'D:\data\icpsr\data\2825\da2825.txt' LRECL=852        PAD;
                       /*  read the record group identifier */
input #1  _RecordSetIdentifier $ 1 - 1  @;

if left(_RecordSetIdentifier) eq left("2 ") then
input
#1 FRECORD $ 1-1
#1 FH_SEQ $ 2-6
#1 FFPOS $ 7-8
#1 FKIND $ 9-9
#1 FTYPE $ 10-10
o
o
o
if left(_RecordSetIdentifier) eq left("2 ") then DO;
  output ICPSR2825FAMILY ;
END;
o
o
o
```

PROBLEMS IN THE METADATA

Unfortunately, having validly structured metadata doesn't always mean having metadata that will translate seamlessly into code that can read the data. The content of the DDI file could just be wrong (e.g. describing a field containing text as numeric, listing the wrong columns for a field, and so on). Since only the <titl></titl> element is required, there could be just not enough information to interpret the data file. There could also be too much information. That was the case for two of the first three ICPSR sample files tested in developing this application.

In the case of our example ICPSR 6084 file and the ICPSR 2825 file, there are some variables (like Q33a from 6084) with values that have two different labels assigned. The format definition would fail if the duplicates were listed separately, so the SAS program concatenates the duplicates.

Format for variable Q33a from ICPSR 6084 without concatenating duplicates:

```
1    proc format;
2    value V00067f      /* format for variable Q33a */
3    1 ='Under'
4    2 ='Over'
5    2 ='Won''t specify/Refused'
ERROR: This range is repeated, or values overlap: 2-2.
6    .I ='Won''t specify/Refused'
7    ;
```

After concatenation:

```
8    proc format;
9    value V00067f      /* format for variable Q33a */
10   1 ='Under'
11   2='_EITHER_ Over _OR_ Won''t specify/Refused'
12   .I ='Won''t specify/Refused'
13   ;
NOTE: Format V00067F has been output.
```

THE SAS PROGRAM AND XML MAP FILE

The SAS program to read the DDI file and write a SAS program as well as the resultant SAS program will be available on the Web at http://www.ku.edu/pri/ksdata/sashttp/sugi30. Contact the author at:

Larry Hoyle
Policy Research Institute
University of Kansas, Blake Hall
1541 Lilac Lane
Lawrence, KS 66044-3177
Email: LarryHoyle@ku.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.