Paper 2481-2015

Managing Extended Attributes With a SAS® Enterprise Guide Add-In

Larry Hoyle, Institute for Policy & Social Research, University of Kansas

ABSTRACT

SAS® 9.4 introduced extended attributes, which are name-value pairs that can be attached to either the data set or to individual variables. Extended attributes are managed through PROC DATASETS and can be viewed through PROC CONTENTS or through Dictionary.XATTRS. This paper describes the development of a SAS® Enterprise Guide custom add-in that allows for the entry and editing of extended attributes, with the possibility of using a controlled vocabulary. The controlled vocabulary used in the initial application is derived from the lifecycle branch of the Data Documentation Initiative metadata standard (DDI-L).

INTRODUCTION

SAS 9.4 introduced extended attributes – (name, value) pairs which can be attached to either datasets or variables. These are stored with the dataset (in an auxiliary file) and in this way allow for the inclusion of metadata as a part of that dataset. (see Olson 2013, or Hemedinger 2013). Before this innovation the only option for storing metadata to be transmitted with a file was to place it into an external file (or dataset). Now, a wide range of metadata can be included in a dataset: variable descriptions, the text of a survey question, the universe from which a measure was taken, measurement units, the funding source of a study and more.

Extended attributes are managed through PROC DATASETS as shown in the example below.

```
proc datasets lib=SASDATA nolist ;
    modify MYSASDATA2_XA ;
XATTR SET VAR fee (Question='How much did you pay?') ;
XATTR SET VAR fee (MeasurementUnits='Total payment in Euros.') ;
XATTR SET DS Abstract='dataset abstract';
```

Extended attributes can also be viewed with proc contents, as seen in Figure 1.

Alphabetic List of Extended Attributes on Variables

Extended Attribute	Attribute Variable	Numeric Value	Character Value
MeasurementUnits	fee	.	Total payment in Euros.
Question	fee	.	How much did you pay?

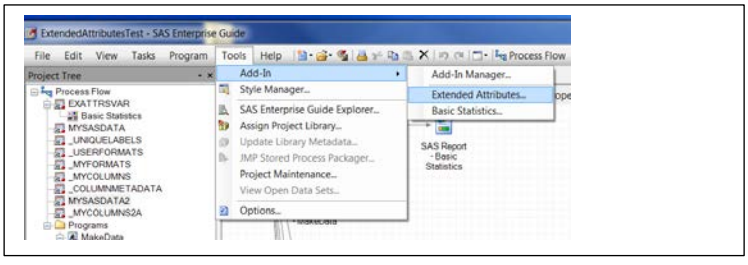
Alphabetic List of Data Set Extended Attributes

Extended Attribute	Numeric Value	Character Value
Abstract	.	dataset abstract

Figure 1 - Viewing Extended Attributes with Proc Contents

Managing extended attributes with PROC DATASETS offers great flexibility. For many data creators, e.g. researchers, writing PROC DATASETS code may not be the most approachable technique for entering metadata. In addition, the unrestricted nature of the attribute names can pose challenges for data discovery and for machine actionability of the metadata. Providing some way of offering an extensible controlled vocabulary for the attribute names is desirable. This could be done through some Web interface, but it also can be done through an Enterprise Guide add in.

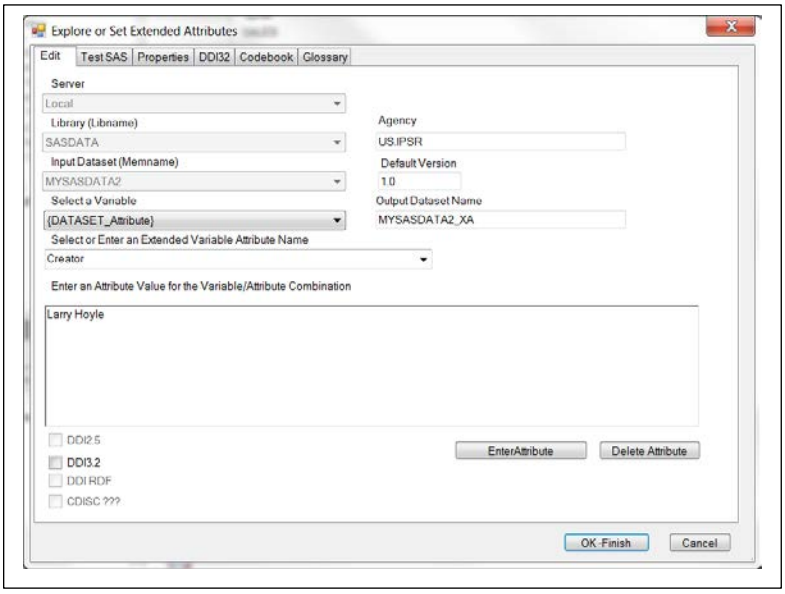
Enterprise Guide offers the ability to integrate add-ins which can read and write SAS datasets, run SAS code, and generate output files. Add-ins are .Net applications written in C# or Visual Basic. Hemedinger (2012) has written a useful guide to creating EG add ins. This paper describes the development of an EG add- in for managing extended attributes.



Display 1 - Invoking an EG Add In

THE EXTENDED ATTRIBUTES ADD-IN

The initial version of the Extended Attributes Add-in is designed to work with the Data Documentation Initiative (DDI) metadata standard. Its main interface allows a user to enter an agency identifier (registered with the DDI Alliance), a default version for the dataset and associated metadata, an output dataset name, and the name of the output dataset (see the upper right side of Display 2).



Display 2 - The Edit Screen of the Extended Attributes Add-In

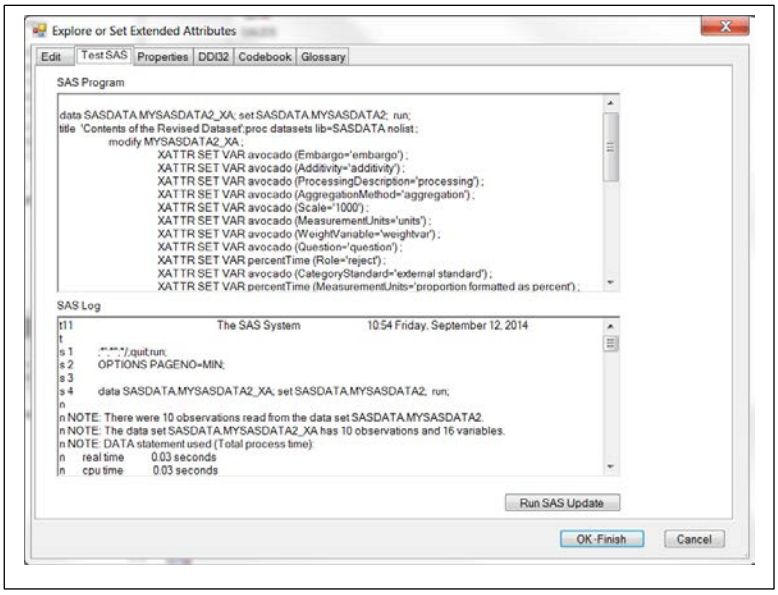
The user selects either the dataset, as seen in Display 2, or a variable name (not shown) from the “Select a Variable” drop down box.

The user then selects either an attribute name from the controlled vocabulary or enters a new attribute name. Definitions for the predefined attributes are displayed in a Glossary tab. Once the attribute name is selected the user enters the attribute value. In the Display 2 example the dataset is given the attribute “Creator” with the value “Larry Hoyle”. Clicking the “Enter Attribute” button adds the (name, value) pair to the list being generated.

In this version of the add-in the dataset and variable attribute names are compiled into the add-in code as in the list of variable attributes in the code below. These lists could be implemented as a user-configurable option.

```
// The controlled vocabulary for variable attribute names
       public List<string> variableAttributes = new List<string>()
        {    "AccessRights", "Additivity", "AggregationMethod", "AnalysisUnit",
             "CategoryStandard", "Concept", "Description",
             "Embargo", "VariableIdentifier",
            "ImputationDescription", "LevelOfMeasurement",
             "MeasurementUnits", "Notes", "ProcessingDescription", "Question",
             "RelevantFormats", "Role", "Scale", "SourceUnit",
             "Universe", "WeightVariable"
        };
```

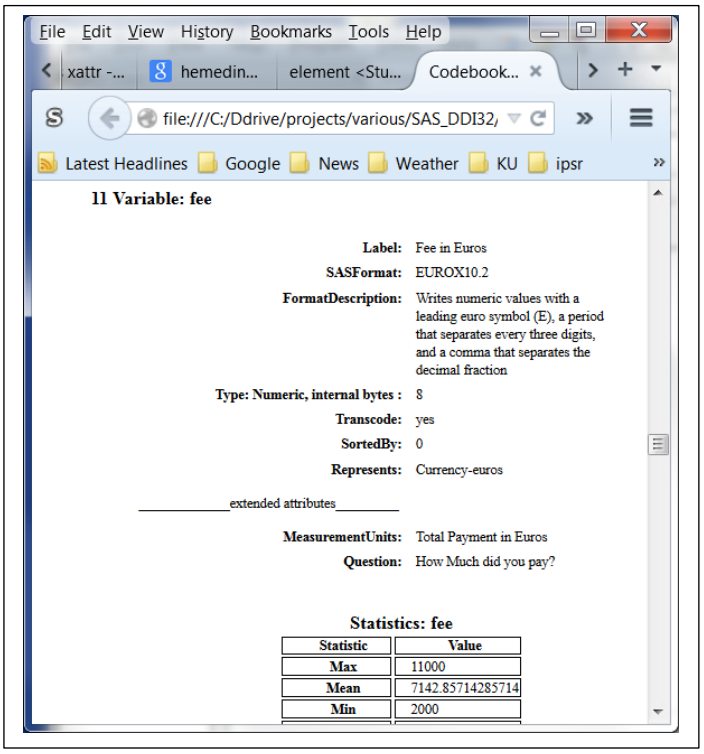
Another tab allows the user to test and view the SAS code and log for the PROC DATASETS being generated by the choices made in the Edit tab.



Display 3 - The Generated Proc Datasets

The Codebook tab allows the user to generate a codebook, currently an XHTML file. The codebook includes not only extended attributes, but also all other metadata that can be gathered from Dictionary tables and current defined and assigned formats and informats, as well as either statistics or frequencies depending on the cardinality of the variable as compared to the cardinality of its assigned format.

Frequencies are generated for numeric data if all discrete values are individually labeled by a user format. Mean, standard deviation, etc. are generated for other numeric variables.

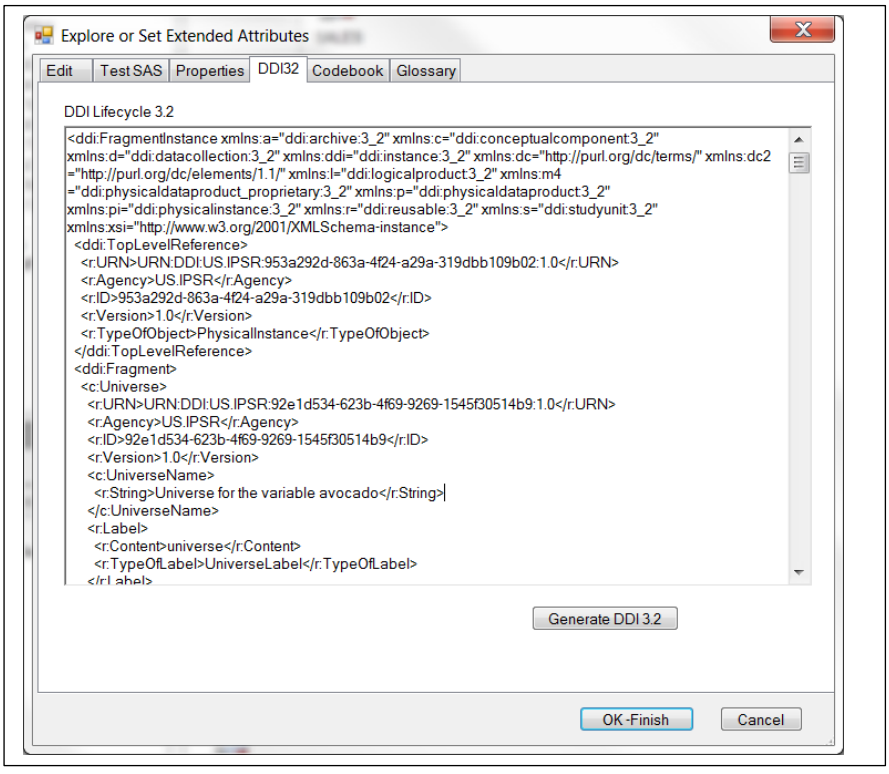


Display 4 - The Codebook Tab

THE DATA DOCUMENTATION INITIATIVE METADATA STANDARD (DDI)

DDI (see DDI 2009) is an established metadata standard for the social and behavioral sciences and can be useful in other disciplines. The lifecycle branch of DDI offers a structure for metadata generated throughout the data lifecycle. Archives such as ICPSR use DDI to structure their metadata as do multiple national statistical agencies. Since the add-in described here uses a set of attribute names corresponding to DDI elements, it is possible to generate a DDI3.2 file in addition to the codebook.

While the add-in metadata are structured as a set of simple key-value pairs, the structure of a DDI instance is much more complex. The add-in provides both a suggested set of keys and programming logic to transform the key-value pairs into the hierarchical DDI structure. Display 5 shows a snippet of DDI code generated by the add-in.



Display 5 – Example metadata structured in DDI3.2

While the current versions of DDI are defined via XML schema, the next version of DDI (DDI4) will be defined by an abstract model, viewable in UML. This will allow expression in both XML and Resource Description Framework (RDF). The DDI Alliance also is developing a vocabulary for data discovery in RDF. A DDI4 based EG add-in could produce either XML or RDF.

As with the codebook, both the extended attributes and the other metadata collected from the SAS dataset (variable labels, integrity constraints, formats, etc.) can be included in the DDI file. For more about representing metadata from a SAS dataset in DDI see Hoyle and Wackerow 2008.

DEVELOPING THE ADD-IN

The custom task add-in was developed in Microsoft Visual Studio. The project contains three main files: *ExtendedAttributesForm.cs, ExtendedAttributesSettings.cs, and ExtractMetadata.sas*. In Visual Studio the form file works together with a visual design tool to develop the user interface. The form file contains the classes for initializing the controls in the form and for handling events generated as the user interacts with the form. The Settings file contains the classes having methods and data structures that do the work of the application. The SAS file contains SAS code that scrapes metadata from the dataset.

In Enterprise Guide the user first selects a dataset and then starts the add-in. Custom task APIs provide access to the library and dataset selected. The variable selection drop-down list can then be populated by a query to DICTIONARY.COLUMNS. Any existing extended attributes can be captured by a query to DICTIONARY.XATTRS. The application makes use of several hashtables. One hashtable contains the extended attributes read from the initial dataset and entered by the user. In this hashtable the key is the variable name and the value is an (attribute name, value) pair stored as a structured string. The code below shows the class which loads extended attributes from the initial dataset into the hashtable.

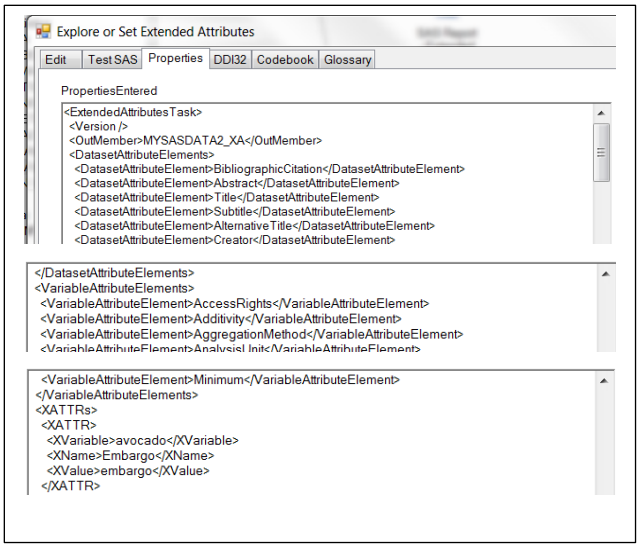
```
private void LoadAttributes()
        {
            SasServer sasServer = new SasServer(ActiveServer);
            datasetVariableName = Settings.getDatasetVariableName();
            using (_connection = sasServer.GetOleDbConnection())
            {
                try
                {
                    _connection.Open();
                    String attrQuery = String.Format("select * from
                       DICTIONARY.XATTRS where xtype='char' and libname='{0}' and
                       memname='{1}' ", ActiveLibrary, ActiveMember);
                    OleDbCommand command = new OleDbCommand(attrQuery,
                                           _connection);
                    using (OleDbDataReader dataReader = command.ExecuteReader())
                    {
                        int fields = dataReader.FieldCount;
                        logger.InfoFormat(" Loading the Hash Table");
                        while (dataReader.Read())
                        {
                            // field 2 is variable name
                            //field 3 is extended attribute name
                            string KeyVarName = dataReader[2].ToString();
                            if (KeyVarName == "") KeyVarName =
                                                    datasetVariableName;
                            string VarAttrKey =
                                             Settings.PackVarAttrKey(KeyVarName,
                                             dataReader[3].ToString());
                            //field 6 is extended attribute value
                            Settings.hashVarAttrs.Add(VarAttrKey,
                                                    dataReader[6].ToString());
                            string unpackedVar;
                            string unPackedAttribute;
                            Settings.UnPackVarAttrKey(VarAttrKey, out unpackedVar,
                                                    out unPackedAttribute);
                            //check to see if this attribute name is already in
                                the list for the comboBox, if not add it
                            if (unpackedVar == datasetVariableName)
                            {
                                if (!
                               Settings.datasetAttributes.Contains(unPackedAttribute))
                                {
                                Settings.datasetAttributes.Add(unPackedAttribute);
                                }
                            }
```

```
                        else
                        {
                            if (!
                        Settings.variableAttributes.Contains(unPackedAttribute))
                            {
                            Settings.variableAttributes.Add(unPackedAttribute);
                            }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                logger.InfoFormat("/*Error trying to read input data: {0} */
                                  \n", ex.Message);
            }
        }
        // reload the combo box, since there might be user defined attributes
in the dataset
        LoadDatasetAttributeNames();
    }
```

Preserving Settings

When the add-in closes it runs a PROC DATASETS to add the extended attributes to a new copy of the dataset.

Enterprise Guide retrieves the SAS code through a call to a specific method. It also needs to be able to restore the state of any critical data structures. These are preserved in an XML structure that can be viewed in the “Properties” tab.



Display 6 - The properties needed for the final Proc Datasets

SAS Code

Since this add-in runs within the Enterprise Guide environment it can generate and run SAS code. The add-in does this for two purposes. On closing the task runs SAS code to add all of the extended attributes to a copy of the original dataset. One method generates a PROC DATASETS to do this. This method can also be invoked from the “Test SAS” tab shown in Display 3.

The other SAS program is contained in the ExtractMetadata.sas file. This code first runs a PROC FORMAT cntlout= procedure to capture all of the currently defined formats and informats. It then runs a DATA step which builds a dataset adding descriptive metadata to some of the predefined SAS formats. Some formats convey information about a variable to which they are associated. An example would be the Euro format. A reasonable assumption about a variable formatted with this format is that it contains a value with measurement units of “currency in Euros”.

The ExtractMetadata program then extracts data from DICTIONARY.COLUMNS, DICTIONARY.XATTRS, DICTIONARY.TABLES, DICTIONARY. CONSTRAINT_COLUMN_USAGE, and DICTIONARY.LIBNAMES. It combines all of this metadata into a dataset (work._ColumnMetadata) comprising a comprehensive description of each variable, including information from associated SAS supplied and user formats. The structure of the dataset is shown in Output 1.

```
create table WORK._COLUMNMETADATA( bufsize=163840 )
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   name char(32) label='Column Name',
   type char(4) label='Column Type',
   length num label='Column Length',
   npos num label='Column Position',
   varnum num label='Column Number in Table',
   label char(256) label='Column Label',
   format char(49) label='Column Format',
   informat char(49) label='Column Informat',
   idxusage char(9) label='Column Index Type',
   sortedby num label='Order in Key Sequence',
   xtype char(12) label='Extended Type',
   notnull char(3) label='Not NULL?',
   precision num label='Precision',
   scale num label='Scale',
   transcode char(3) label='Transcoded?',
   fmtName char(200),
   xname char(200),
   xlabel char(200),
   represents char(32) label='the type of entity the format represents e.g. euro is
currency-euros',
   formatDescription char(200),
   fmtKey char(200),
   infmtKey char(200) label='A unique identifier for an informat, distinguishes
numeric from character for formats with the same
name',
   infmtType char(1) label='Numeric (N) or Character (C) informat',
   uFmtName char(32) label='User Format Name',
   hasRange num label='True (1) if the format has different start and end values',
   FmtLevels num label='number of unique levels for the format',
   NLevels num format=BEST8. label='Number of Levels',
   NMissLevels num format=BEST8. label='Number of Missing Levels',
   NNonMissLevels num format=BEST8. label='Number of Nonmissing Levels',
   UnformattedValues num label='Formatted variable has no ranges and some values are
unformatted',
   ConstraintNote char(200) label='A description of any integrity constraints on the
variable'
  );
```

Output 1 - Structure of the ColumnMetadata table

A few variables in this dataset are computed for use in generating the structured metadata (DDI and codebook). Since SAS formats and informats for character and numeric variables may have the same names, it is useful to have a key (fmtKey, infmtKey) combining the (in)format name and datatype for the format (N or C). SAS formats may also label ranges of values. These need to be treated differently in DDI, so a logical variable (hasRange) indicating that a column is formatted via ranges is useful.

Frequencies or Summary Statistics?

The program also runs either PROC MEANS or PROC FREQ to generate statistics or frequencies on each variable. It looks at the data type and compares the cardinality (number of unique values) of each variable with the cardinality of any associated format to make this decision. Output 1 shows the structure of the table containing metadata for each column.

The following SAS code builds lists of variable names which are either continuous, categorical numeric, or categorical character based on several criteria which have been included in the master metadata table. A variable is treated as continuous, for example, if:

- The variable type is numeric and
- the variable is not formatted as Date/Time
- the variable has no user format (uFmtName = " ")
- the variable does not have a format with ranges for values (hasRange is true)
- the variable has more unique values than the number of distinct formatted values (unformattedValues is true)

```
proc sql noprint;
select name into :ContinVarList separated by " "
from work._columnMetadata
where  type="num" AND
       not represents in ("Date", "Time", "DateTime") AND
       (uFmtName = " " OR
        (hasRange OR UnformattedValues)
       )
;

select name into :CatVarListN  separated by " "
from work._columnMetadata
where type="num" AND
      uFmtName ne " " AND
      not (hasRange OR UnformattedValues);

select name into :CatVarListC  separated by " "
from work._columnMetadata
where type="char" AND
      uFmtName ne " " AND
      not (hasRange OR UnformattedValues);
```

A PROC MEANS is run on the variables in the &ContinVarList list. A PROC FREQ is run on the variables in the other two lists. A combined table contains the results of both procedures as shown in Display 7. These are then included for each variable in the codebook and the DDI xml.

	variableName	StatisticOrCode	Value
49	sex	.	1
50	sex	1	6
51	sex	2	3
52	sexC		1
53	sexC	F	3
54	sexC	M	6
55	tempC	Max	121.111...
56	tempC	Mean	29.1111...
57	tempC	Min	-...
58	tempC	P25	0
59	tempC	P50	21.1111...
60	tempC	P75	48.8888...
61	tempC	Range	161.111...
62	tempC	StdDev	50.8954...

Display 7- Frequencies and Summary Statistics

SUGGESTIONS FOR SAS

The ability to attach extended attributes to variables and datasets is an important addition to SAS for documenting datasets. With SAS 9.4, though, extended attributes do not follow variables through their reuse in other datasets. It would be useful if extended attributes followed variables just like the built-in attributes (label, format, etc.). If a variable is selected for a new dataset by a SET statement in a DATA step, or SELECT statement in an SQL query, it would make sense for it to keep the extended attributes it had in the original dataset, even across simple renaming of the variable.

If extended attributes traveled with variables, a variable could be assigned a Universally Unique Identifier (UUID) upon creation. This would allow tracking and building provenance chains for variables across datasets. A transformation such as `bmi=weight/height**2;` could even automatically add the UUIDs for weight and height to an attribute "basedOn" for variable bmi. A transformation like `x = x*2;` could generate a new UUID and add the original UUID to the provenance chain. This metadata could then be used for machine generated documentation of data dependencies.

CONCLUSION

This custom add-in was developed as a demonstration project and seems to have resulted in a usable tool for capturing structured metadata at the point of creation of a dataset. Waiting until later stages of a research project to record metadata carries the risk of inaccuracy as memory of what specifically was done fades. The ability to include metadata in extended attributes would also be compatible with metadata driven generation of metadata rich SAS datasets as is being done in the survey world (Iverson 2009).

Enterprise Guide allows for the development of sharable user interfaces which can invoke the full power of SAS code. The complete .NET and SAS code from this project is available to others who wish to build on it. Future enhancements could include an improved user interface allowing for more hierarchy in the metadata. Multiple collaborators, for example, might have different roles and for each role might have a different degree of contribution. One possibility for storing such information in extended attributes would be to use structured attribute values. XML might be a logical format for that structure. These could be represented in a program as a hash of hashes.

Another area of extension might be to capture the E.G. process flow – nodes, links, and code within nodes into the structured metadata. DDI offers a generation instruction element which can record code used to generate a variable. A simplistic approach would be to record all of the code from the path up to the target dataset and place a reference to that code in the metadata for each variable. More sophisticated processing could parse the code and extract just the code relevant to generating each variable. This might not be straightforward if part of the generation of a variable involves a statistical procedure (e.g. a residual variable from a regression).

Finally, this approach is not limited to DDI as a metadata structure. Metadata corresponding to other standards, e.g. CDISC, could be collected at the dataset and variable level via key-value pairs. The appropriate dataset and variable level keys could be identified, and programming logic developed to map those pairs into the appropriate structure.

PROJECT ARCHIVE

Complete code for this project has been archived at:

http://kuscholarworks.ku.edu/dspace/handle/1808/12488

REFERENCES

- Allen, Liz, Jo Scott, Amy Brand, Marjorie Hlava & Micah Altman (2014). *Publishing: Credit where credit is due?* Nature 508, 312–313 (17 April 2014), doi:10.1038/508312a.
- DDI Alliance. "Data Documentation Initiative". 2009 Available at http://www.ddialliance.org/ .
- Hemedinger, Chris. 2012. *Custom Tasks for SAS® Enterprise Guide® Using Microsoft .NET*. Cary, N.C. : SAS Institute
- Hemedinger, Chris "How to store data about your data in your data." The SAS Dummy A SAS® blog for the rest of us. October 17, 2013, Available at http://blogs.sas.com/content/sasdummy/2013/10/17/extended-attributes-sas-94/.
- Hoyle, Larry and Joachim Wackerow. "Exporting SAS Datasets to DDI 3 XML files Data, metadata, and more metadata". *Proceedings of the SAS Global Forum 2008.* Cary, NC: SAS Institute. Available at http://www2.sas.com/proceedings/forum2008/137-2008.pdf.

- Iverson, Jeremy 2009 “Metadata-Driven Survey Design.” IASSIST Quarterly Spring-Summer 2009. International Association for Social Science Information Services & Technology.
- Olson, Diane. 2013. “Developer Reveals: Extended Data Set Attributes.” *Proceedings of the SAS Global Forum 2013*. Cary NC: SAS Institute. Available at http://support.sas.com/resources/papers/proceedings13/135-2013.pdf.
- W3C. “Resource Description Framework (RDF)”. 15 March 2014. Available at http://www.w3.org/RDF/.

ACKNOWLEDGMENTS

Thanks to Chris Hemedinger for some useful tips in getting started with this project.

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, Suite 607 Blake Hall
Lawrence, KS, 66045-3129
LarryHoyle@ku.edu