PharmaSUG 2014 - Paper AD10

DefKit : A Micro Framework for CDISC Define-XML Application Development

Lei Zhang, Celgene Corp., Summit, NJ

ABSTRACT

CDISC Define-XML specification [1] provides an extended ODM model to describe the clinical data and statistical analyses submitted for FDA review. However, because of the inherent complexity, creating define.xml files from clinical studies poses many challenges to SAS® programmers. In this paper, we introduce a micro-framework called DefKit to facilitate the development of CDISC/XML applications.

DefKit is designed as a rapid development framework, and is built specifically for SAS programmers who want to have a simple way to create CDISC/XML applications, such as define.xml generators. The toolkit that the DefKit framework provides consists of a very small set of macros and user defined functions (UDFs), and employs hash objects for fast data retrieval and XML generation. This paper first introduces the organization and usages of the DefKit framework components, and then provides examples to illustrate how to generate define.xml elements with DefKit. Apart from explaining how to implement these elements, it demonstrates how DefKit's unique features make it easy to develop and maintain CDISC/XML applications .

INTRODUCTION

CDISC define.xml files are part of key documents required by FDA for electronic submissions of CDISC datasets, such as SDTM and ADaM. They act as a road map to guide regulatory reviewers in their exploration and scrutiny of the submitted documents. Therefore, a define.xml has to be built according to CDISC Define-XML specifications so that reviewers can have a consistent way to navigate study metadata and understand the clinical studies submitted for approval. However, creating define.xml is not an easy task, it requires a good set of XML programming skills that many SAS programmers lack [2]. This paper describes a micro framework called DefKit that is aimed to ease the construction of define.xml files and related applications.

Hello World Example

Perhaps the easiest way to understand the DefKit framework is to jump right in and do it with a simple example. Let's suppose we want to create an ODM/ XML file that contains greetings for each of students in the well-known dataset SASHELP.CLASS. The output file HelloWorld.xml would be like this

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ODM ODMversion="1.2">
	<Study OID="Hello World">
		<MetaDataVersion>
			<Rows>
				<row>
					<msg>Hello, John!</msg>
				</row>
				<row>
					<msg>Hello, Alice!</msg>
				</row>
				. . .
			</Rows>
		</MetaDataVersion>
	</Study>
</ODM>
```

Before using the DefKit to create a XML file, the DefKit library has to be set up first. This can be done in three easy steps.

Step 1 – Create a folder to store the DefKit macros and user-defined functions

The DefKit framework consists of two SAS files: DefKitMacs.sas, and DefKitUDFS.sas. Both are provided in the appendix of this paper. The first step you have to do is create a DefKit library folder to store them.

Step 2 – Compile the user-defined functions (UDFs) provided in the DefKitUDFS.sas

In order to use the DefKit user defined functions, you need to run the DefKitUDFS.sas once from a SAS interactive session to create the compiled UDFs datasets in the DefKit library folder.

Step 3 – Create a setup.sas for DefKit macros and UDFs

In order to use the macros and compiled user-defined functions stored in the DefKit library, you need create a SETUP.sas to configure a session with the DefKit library. Assuming *U:\DefKit* is the DefKit library folder, you can create a `SETUP.sas` like below

```
/*  A sample setup.sas program for DefKit micro framework */
LIBNAME DefKit "U:\DefKit";   /*  Create a library for the DefKit micro framework */

options sasautos = ("U:\DefKit",sasautos) cmplib=(DefKit.DefKit);

%DefKitMacUtil;  /*  Initialize the DefKit macros */
```

After executing the SETUP.sas, you can use following code to create the `HelloWorld.xml` in the SAS session.

```
/* HelloWorld.SAS */
DATA HelloWorld(keep=XMLLINE);
  LENGTH XMLLINE $8192;
  LENGTH XMLVersion XMLEncoding ODMVersion OID $16;
  if 0 then set SASHELP.CLASS; /* Define dataset variables for the hash object */

  %XHash(H1:ITER1[NAME/NAME],SASHELP.CLASS, multidata:'n')

  XMLVersion="1.0"; XMLEncoding="ISO-8859-1";ODMVersion="1.2"; OID="Hello World";
  %XOUT(T0, "<?xml version=@ encoding=@?>", XMLVersion, XMLEncoding)
  %XOUT(T0, "<ODM ODMversion=@>", ODMVersion)
  %XOUT(T1, "<Study OID=@>",OID)
  %XOUT(T2, "<MetaDataVersion>")
  %XOUT(T3, "<Rows>")
  RC1=ITER1.FIRST();
  DO WHILE (RC1=0);
    %XOUT(T4, "<row>")
    %XOUT(T5, "<msg>Hello, @t0!</msg>",name)
    %XOUT(T4, "</row>")

    RC1=ITER1.NEXT();
  END;
  %XOUT(T3, "</Rows>")
  %XOUT(T2, "</MetaDataVersion>")
  %XOUT(T1, "</Study>")
  %XOUT(T0, "</ODM>")
Run;

%TODEFXML(
  HelloWorld
  ,OUTDIR=U:\temp
  ,OutFile=HelloWorld.xml
)
```

The above code consists of two steps: the first step is to use a data step to create a XML dataset that contains a character variable XMLLine for the XML code created in the data step. The data step first defines a hash object with the macro %XHASH for the direct dataset access, and then calls %XOUT to generate the XML code and store it in the variable XMLLine. The second step is to call %ToDefXML to simple convert the XML dataset into an ODM/XML file. It is obvious that the above code is much simpler and clearer than the code written with data step put statements.

COMPONENTS IN THE DEFKIT MICRO FRAMWORK

The DefKit framework consists of a few macros and user-defined functions with very small foot prints. Below are its primary components that a user has to know.

Inline Macros

DefKit provides three inline macros to help users to create a XML dataset with a data step. %XHash allows users to define a hash object for define-xml meta dataset access in a concise way. %XOUT and %XCAT let users to create lines of the XML output through user-defined text line templates, which we'll cover later.

Regular Macro

DefKit provides a regular macro called %ToDefXML to help users to combine XML datasets and create a XML file.

User-Defined Functions
Defkit defines two data step subroutines via PROC FCMP to help users to parse complex CDISC variables and extract sub-components for CDISC/XML document generation.

DefKit has an open structure. Users can add more macros and UDFs to the framework for their own development needs.

USAGES OF DEFKIT MACROS AND UDFS

DefKit employs two important programming concepts in SAS® 9 for its implementation. One is the hash object, which lets users to directly access datasets in a data step with little or no prior data sorting, setting or merging. The other is user-defined function or UDF, implemented via PROC FCMP facility. The direct data-addressing mechanism that the hash object brings about can significantly speed up the data retrieval and manipulation and the user-defined function facility allows users to define their own functions, or subroutines for data step use. For more information on how to use them, please see [6] [7] respectively.

1. DefKit Macros

Users can use %XHASH, %XCAT, %XOUT, and %ToDEFXML to create XML datasets and files within the DefKit framework.

1.1 %XCAT – Create a XML string via a line template

%XCAT is an inline macro function for data steps, acting just like any built-in data step functions. It has following arguments,

```
%XCAT(template, exp1, exp2, . . .,expn )
```

Where the first parameter template defines a line template, and the rest are character variables, or expressions to be formatted according to the line template. The line template is composed of *literal substrings* and format specifiers. The format specifiers start with the meta-character @ optionally followed by one or two control letters and numbers. They instructs the macro how to trim and/or quote the character values from the corresponding variables, or expressions and where to insert the character values. For example, the following code

```
dsname="  adqsadas  "; fileext="xpt";
s=%XCAT("<def:title>@t0.@s2</def:title>", dsname, fileext);
put s;
```

would create the text <def:title> adqsadas."xpt"</def:title> from variables dsname and fileext via the line template <def:title>@t0.@s2</def:title>, where the two format specifiers in the line template have following meanings:

@t0 tells %XCAT to right trim variable *DSName* and replace itself with the trimmed character value, and

@s2 tells %XCAT to strip variable *fileext*, enclose its value with double quotes, and replace itself with the quoted string.

The general syntax of a format specifier is

```
@<trim-modifier><quotation-mark-modifier>
```

The modifiers are defined in the following table

Modifier Type	Modifier Character	Meanings
Trim	S	Strip a character variable or expression (default).
	T	Right trim a character variable or expression.
Quotation Mark	2	Enclose a character variable or expression in double quotes (default).
	1	Enclose a character variable or expression in single quotes.
	0	No quotes added.

Here are a few more examples :

Format Specifers	Meanings
@ or @2	Strip a character variable or expression and enclose it in double quotes
@s1 or @1	Strip a character variable or expression and enclose it in single quotes
@s0 or @0	Strip a character variable or expression
@t2	Right trim a character variable or expression and enclose it in double quotes
@t1	Right trim a character variable or expression and enclose it in single quotes
@t0	Right trim a character variable or expression

1.2 %XOUT - Create a XML string via a line template and store it in a XML dataset

%XOUT is an inline macro for data steps too, but it is not an inline macro function like %XCAT. Its main purpose is to create an indented and formatted XML string via a line template, assign it to a specified character variable (by default, XMLLINE) and then output it to a XML dataset. It consists of following arguments:

```
%XOUT(indent, template, exp1, exp2, . . ., expn, LINEVAR=XMLLINE, DSNAME=)
```

Where the first positional parameter *indent* takes following two indentation modifiers:

```
Tn  -   add n tabs to the beginning of a XML string
Sn  -   add n spaces to the beginning of a XML string
```

Where *n* >=0.

The rest of positional parameters are same as those defined in the %XCAT( ). Besides, you can use two additional key parameters to assign the generated XML string to a different variable (instead XMLLINE) and/or output it to a specified XML dataset. The first key parameter *LINEVAR*= provides the name of a character variable a user chooses to store the generated XML string. By default, the default variable name is XMLLINE. The second key parameter *DSNAME*= is used to optionally assign a dataset name that the resulting XML string can be outputted to. Here is a simple example:

```
dsname="adqsadas"; fileext="xpt";
%XOUT(T2, "<def:title>@.@</def:title>", dsname, fileext, LineVar=s, DSName=XMLDS1);
put xmlline;
```

The above code , once executed in a data step, will create a XML string like previous example but with additional two tab characters at the beginning of the string, and the output will go to the dataset XMLDS1.

1.3 Other Inline Utility Macros

The DefKit framework also includes a couple more of inline macro utilities for end users, such as %XESC(s), which is used to escape the XML control characters embedded in a character variable. Please see their definitions and usages in the appendix.

1.4 %XHash – Define and instantiate a hash object

%XHash allows a user to define and instantiate a hash object, including the corresponding hash iterator, in a shorthand way in a data step. It has following interface:

```
%XHash(
   HashExp     /*  An expression for a hash, and its hash iterator               */
  ,DSNEXP      /*  Dataset name associated with the hash object                  */
  ,HashOpts    /*  A list of hash options separated by spaces                    */
);
```

Where

HashExp has following general syntax

```
Hash-name<:Hash-iterator-name>[Key1 Key2  Key3 . . . </Val1 Val2 Val3 . . . >]
```

Where `Key1, Key2, Key 3, . . .` are key variables to the hash object, and `Val1, Val2, Val3, . . .` are value variables of the hash object. They are the variables in the the source dataset defined by the parameter `DSNEXP`.

DSNEXP is the name of the data set that the hash object uses, optionally followed by common dataset options such as `WHERE=`, and `KEEP=`.

HashOpts is a list of hash options separated by spaces, such as `multidata:'Y'` and `ordered:'a'`.

%XHASH lets a users define and instantiate a hash object easily in a data step, for example, the following macro call

```
%XHash(H1:ITER1[NAME/Height Weight],SASHELP.CLASS, multidata:'n' ordered:'a')
```

is equivalent to a block of following code for the hash object declaration and instantiation

```
DECLARE HASH H1(dataset:"SASHELP.CLASS",multidata:'n');
DECLARE HITER ITER1('H1');
RC=H1.defineKey('NAME');
RC=H1.definedata('HEIGHT', 'WEIGHT');
RC=H1.DefineDone();
```

Once a defined hash object is instantiated, you can retrieve the data from a hash object with following two code patterns.

Pattern 1: Access all the key/value pairs in a hash object

```
%XHash(h1:iter1, dataset, multidata:'Y',. . .);
. . .
rc1=Iter1.FIRST()
DO While (rc1 =0);
  /* Processing the variables fetched from the hash object */
  . . .
  rc1=ITER1.NEXT();
END;
```

Pattern 2: Access all key/value pairs in a parent hash object and its associated child hash object

```
%XHash(parent:parent_iter,dataset,options);
%XHash(child, dataset, multidata:'Y');
. . .
rc1=parent_iter.FIRST()
DO While (rc1 =0);
    /* Processing the variables fetched from the parent hash object */
    . . .
    rc2=child.FIND();
    DO WHILE(rc2 = 0);
      /* Processing the variables fetched from the both parent and child hash objects */
      . . .
      rc2 = child.FIND_NEXT();
    END;
    . . .
    rc1=parent_iter.NEXT();
END;
```

To illustrate how to uses the code patterns to creates a XML dataset, let's take the CDISC def:leaf element as an example.

An instance of the `def:leaf` element in CDISC Define-XML version 2 looks like this

```
<def:leaf ID="LF.ADRG" xlink:href="analysis-data-reviewers-guide.pdf">
  <def:title>Analysis Data Reviewer's Guide</def:title>
</def:leaf>
```

Assuming we use the define-xml meta dataset *`model.links`* described in [2], we can write the following code to generate `def:leaf` instances in a XML dataset.

```
DATA Leaf(keep=XMLLINE);
  LENGTH XMLLINE $8192;
  if 0 then set model.links;
  %XHash(H1:ITER1[leafID],MODEL.LINKS, multidata:'n');

  RC1=ITER1.FIRST();
  DO WHILE (RC1=0);
    If lengthn(LeafTitle)=0 then LeafTitle=LeafID;
    %XOUT(T1, '<def:leaf ID="LF.@s0" xlink:href=@>', LeafID, lowcase(LeafHref))
    %XOUT(T2, "<def:title>@s0</def:title>", %XESC(LeafTITLE))
    %XOUT(T1, "</def:leaf>")
    RC1=ITER1.NEXT();
  END;
Run;
```

For more examples of how to use `%XHash` , `%XCAT`, and `%XOUT`, please see the next section.

1.5 %ToDEFXML - Create a XML file from a list of XML datasets

`%ToDEFXML` helps users to create a XML file from a list of XML datasets. It is defined as follows.

```
%TODEFXML(
  DSNLST               /* A list of properly-ordered XML datasets (required)  */
  ,OUTDIR=             /* Output directory for the xml file (required)        */
  ,OUTFILE=DEFINE.XML  /* Define file name, default is DEFINE.XML             */
  ,LINESIZE=8192       /* Line size for variable XMLLINE. Default is 8192     */
  ,OPTS=               /* Options:                                            */
                       /*    ODMEnd:  Close the XML file with ODM ending tags */
);
```

Here is a simple example copied from [2].

```
%TODEFXML(
  HDR ValueListDef WhereClauseDef ItemGroupDef ItemDef CodeList MethodDef CommentDef leaf
  ,OUTDIR=define-file-folder
  ,OUTFILE=Define.XML
  ,OPTS=ODMEND
)
```

2. User-Defined Functions

DefKit micro framework encourages users to develop CDISC/XML applications according to Object-Relational model [2]. This is because CDSISC/XML applications often have to deal with complex variables with analytical sub-components, such as comments with embedded user-defined links, or range checks with comparators. Since the user-defined function facility in SAS® 9 can be considered as a kind of type-extension mechanism for the SAS language, we want to treat complex variables as conceptual objects and use the `variable + user-defined-functions` approach to simulate them [2]. The thumb of rule used here is, if a complex variable contains values that need to be parsed into sub-components for further use, we treat it like a new type of variables and associate a couple of user-defined functions or subroutines with it. Because comment variables, and range check variables are the variables of this type in the CDISC define-xml specification, the DefKit framework provides following two subroutines to ease the sub-component extraction from their values during the ODM/XML document generation.

2.1 GetDocRefs() - a subroutine to extract document reference links from a comment variable

GetDocRefs() is a data step subroutine that is used to extract document links form a comment variable. The document links embedded in the comment variable are supposed to be enclosed in tags like <LF.*links*>. The subroutine has following syntax

```
Subroutine GetDocRefs(VAR $,LeafRef[*] $,n,trunc);
```

Where

Var - comment variable that may contain the tags like <LF.links> as the documents reference links.

LeafRef[*] - an array that contains document reference links extracted from the common variable.

N - actual number of document reference links extracted.

Trunc - flag variable to indicate whether all the documents reference links are extracted into the array.

Here is an example to illustrate how to use this subroutine

```
LENGTH doc $8192 num 8 trunc 8 idx 8;
ARRAY  DOCREFS[32] $1024 _temporary_;

doc="For mathod 1, please see <LF.method-1>; otherwise please see <LF.method-2>";
num=0; trunc=0;
Call GetDocRefs(doc, docrefs, num, trunc);

put num= trunc=;
do idx=1 to num;
  put DOCREFS[idx]=;
end;
```

The log output from the above code would look like this.

```
num=2 trunc=0
DOCREFS[1]=LF.method-1
DOCREFS[2]=LF.method-2
```

2.2 GetRangeChecks() - a subroutine to extract comparators from a range check variable

GetRangeChecks() is a user-defined subroutine to handle the range checks in a def:WhereClauseDef element introduced in CDISC Define-XML version 2. The def:WhereClauseDef element is associated with value level metadata. It is used to describe the conditions to obtain a subset of the dataset rows that share similar metadata. Each value definition may have a Where Clause attached to it in order to describe when that value applies. Where Clauses define a condition using one or more range checks, which allows the construction of complex where clauses. The difficult part of implementing a def:WhereClauseDef element is how to create child RangeCheck elements. One of the solutions is to treat it as a complex variable and ask users to annotate each of where clauses with phases like followings:

Rangechcek 1: Variable1 *Comparator_1* [*Val11 Val12* ...]
Rangechcek 2: Variable2 *Comparator_2* [*Val21 Val22* ...]
...

If variables in the range check comes from another dataset, then use following format

RangeCheck *n*: *Dataset.Variable Comparator_n* [*VALn1 VALn2* . . .]

Since a range check variable is treated as a complex variable, a subroutine called getRangeChecks() is developed to extract all range checks from the where clause paragraphs to create child RangeCheck elements. The subroutine is defined with following syntax

```
SUBROUTINE GetRangeChecks(VAR $,Comparator[*] $,Entry[*] $,n,trunc);
```

Where

Var – where-clause variable that contains range checks defined above.

Comparator[*] - an array that contains comparators extracted from Var.

`Entry[*]` - an array that contains value entries extracted from `Var`for the corresponding comparators.

`N` - actual number of comparators extracted

`Trunc` - flag variable to indicate whether all the comparators are collected in the array

Here is an example to illustrate how to use the subroutine

```
    LENGTH doc $8192 num 8 trunc 8 idx 8;
    ARRAY  Comparators[64] $1024 _temporary_;
    ARRAY  Items[64] $1024 _temporary_;

    doc="RangeCheck 1:PARAMCD EQ[ACTOT]; RangeCheck 2:PARAMCD EQ[ACTOT DCTOT]";
    num=0;trunc=0;
    Call GetRangeChecks(DOC, Comparators, Items, num,trunc);

    put num= trunc=;
    do idx=1 to num;
      put Comparators[idx]= Items[idx]= ;
    end;
```

The log output from the above code would look like this once executed.

```
num=2 trunc=0
Comparators[1]=PARAMCD EQ Items[1]=ACTOT
Comparators[2]=PARAMCD EQ Items[2]=ACTOT DCTOT
```

CONSTRUCT DEFINE-XML ELEMENTS WITH DEFKIT

In this section, we will illustrate how to use DefKit micro framework to create Define-XML v2.0 ADaM elements based on the OR model proposed in [2], and provide you more examples of using DefKit utilities. It is assumed that users have a good understanding of CDISC ADaM elements specified in the Define-XML Specification v2.0 (published in March 2013).

Example 1: Generating ItemGroupDef Elements

An *ItemGroupDef* element is used to describe the metadata about a dataset. It contains a set of attributes, child elements that refer to *ItemDef* elements, and the link to .XPT files. Below is an example of *ItemGroupDef* element that describes an ADSL dataset.

```
     <!-- ItemGroup Definition (ADSL) -->

     <ItemGroupDef OID="IG.ADSL" Name="ADSL" SASDatasetName="ADSL" Repeating="No" IsReferenceData="No"
Purpose="Analysis" def:Structure="one record per subject"

       def:Class="SUBJECT LEVEL ANALYSIS DATASET" def:CommentOID="COM.ADSL" def:ArchiveLocationID="LF.ADSL">

       <Description>

         <TranslatedText xml:lang="en">Subject-Level Analysis</TranslatedText>

       </Description>

       <ItemRef ItemOID="IT.ADSL.STUDYID" OrderNumber="1" Mandatory="No"/>

       <ItemRef ItemOID="IT.ADSL.USUBJID" OrderNumber="2" Mandatory="No" KeySequence="1"/>

       <ItemRef ItemOID="IT.ADSL.SITEGR1" OrderNumber="3" Mandatory="No" MethodOID="MT.ADSL.SITEGR1"/>

      . . .

       <ItemRef ItemOID="IT.ADSL.DCREASCD" OrderNumber="47" Mandatory="No" MethodOID="MT.ADSL.DCREASCD"/>

       <def:leaf ID="LF.ADSL" xlink:href="adsl.xpt">

         <def:title>adsl.xpt </def:title>

       </def:leaf>
```

```
      </ItemGroupDef>
```

In order to create `ItemGroupDef` elements, we use two hash objects for the model datasets from [2]: `DOMAIN` and `DOMVARS`. Below is the complete code to create the XML dataset for ItemGroupDef element.

```
DATA ItemGroupDef(keep=XMLLINE);
    LENGTH XMLLINE $8192 KeySequence 8;
    LENGTH TMP1 TMP2 $1024;

    if 0 then set MODEL.DOMAIN MODEL.DOMVARS;

    %XHash(H1:ITER1[DOMNAME],MODEL.DOMAIN, multidata:'n');
    %XHash(H2[DOMNAME]      ,MODEL.DOMVARS,multidata:'y');

    RC1=ITER1.FIRST();
    DO WHILE (RC1=0);

        %XOUT(T1, '<ItemGroupDef OID="IG.@s0" Name=@ SASDatasetName=@ Repeating=@
IsReferenceData=@ Purpose="Analysis" def:Structure=@ def:Class=@ def:CommentOID="COM.@s0"
def:ArchiveLocationID="LF.@s0">',
                DOMNAME, DOMNAME, DOMNAME, DOMREP, DOMISREF,
                DOMSTRUCT, DOMCLASS, DOMNAME, DOMNAME);

        %XOUT(T2, "<Description>");
        %XOUT(T3, '<TranslatedText xml:lang="en">@t0</TranslatedText>', %XESC(DOMDESC));
        %XOUT(T2, "</Description>");

        RC2= H2.FIND();
        DO WHILE(RC2 = 0);
            KeySequence=Findw(upcase(strip(DOMKeys)),upcase(strip(VARNAME)),", ","E");
            TMP1="";
            if KeySequence > 0 then TMP1=%XCAT("KeySequence=@", KeySequence);
            TMP2="";
            if VARORIGIN="Derived" then TMP2=%XCAT('MethodOID="MT.@0.@0"', DOMNAME, VARNAME);
            %XOUT(T2, '<ItemRef ItemOID="IT.@s0.@s0" OrderNumber=@ Mandatory=@ @t0 @t0/>',
                    DOMNAME, VARNAME, VARORDER, VARREQD, TMP1, TMP2);
            RC2 = H2.FIND_NEXT();
        END;

        %XOUT(T2, '<def:leaf ID="LF.@s0" xlink:href="@s0.xpt">', DOMNAME, lowcase(DOMNAME));
        %XOUT(T3, "<def:title>@t0.xpt</def:title>", lowcase(DOMNAME));
        %XOUT(T2, "</def:leaf>");
        %XOUT(T1, "</ItemGroupDef>");

        RC1=ITER1.NEXT();
   END;
Run;
```

Example 2: Generating ItemDef Element

An *`ItemDef`* element is used to represent variable metadata. Variables are associated with datasets through *`ItemRefs`* contained in *`ItemGroupDef`* elements. Additional variable metadata that are associated with *`ItemDefs`* and *ItemRefs* are provided with following elements:

- *`Codelist`* for controlled terminology
- *`def:ValueListDef`* for value level metadata
- *`MethodDef`* for computational method
- *`def:CommentDef`* for comments
- *`def:Origin`* for variable origin

A variable can have both a codelist and a valuelist associated with since they provide different semantic information for the variable. A codelist provides a list of allowable values that a variable can accept while a valuelist is used to define metadata based on the value of another variable in order to support data review and analysis when the variable metadata itself is not sufficient. Here are a few instances of *`ItemDef`* elements.

```
      <ItemDef OID="IT.ADQSADAS.PARAMN" Name="PARAMN" SASFieldName="PARAMN" DataType="integer" Length="8"
```

```
def:CommentOID="COM.ADQSADAS.PARAMN">

        <Description>

          <TranslatedText xml:lang="en">Parameter (N)</TranslatedText>

        </Description>

        <CodeListRef CodeListOID="CL.PARAMN_ADQSADAS"/>

        <def:Origin Type="Assigned"/>

      </ItemDef>

. . .

   <ItemDef OID="IT.ADQSADAS.AVAL" Name="AVAL" SASFieldName="AVAL" DataType="integer" Length="8">

        <Description>

          <TranslatedText xml:lang="en">Analysis Value</TranslatedText>

        </Description>

        <def:ValueListRef ValueListOID="VL.ADQSADAS.AVAL"/>

      </ItemDef>

. . .

      <ItemDef OID="IT.ADQSADAS.AWTARGET" Name="AWTARGET" SASFieldName="AWTARGET" DataType="integer" Length="8"
def:CommentOID="COM.ADQSADAS.AWTARGET">

        <Description>

          <TranslatedText xml:lang="en">Analysis Window Target</TranslatedText>

        </Description>

        <def:Origin Type="Assigned"/>

      </ItemDef>

. . .

      <ItemDef OID="IT.ADQSADAS.ANL01FL" Name="ANL01FL" SASFieldName="ANL01FL" DataType="text" Length="1">

        <Description>

          <TranslatedText xml:lang="en">Analysis Record Flag 01</TranslatedText>

        </Description>

        <CodeListRef CodeListOID="CL.Y_BLANK"/>

        <def:Origin Type="Derived"/>

      </ItemDef>
```

In order to create *ItemDef* elements, we use three hash objects to directly access the model datasets: DOMAIN, DOMVARS and PVL. Below is the sample code to create a XML dataset with instances of *ItemDef* element.

```
DATA ItemDef(keep=XMLLINE);
    LENGTH XMLLINE $8192;
    LENGTH TMP1 TMP2 TMP3 $1024;

    if 0 then set MODEL.DOMAIN MODEL.DOMVARS MODEL.PVL;

    %XHash(H1:ITER1[DOMNAME],MODEL.DOMAIN, multidata:'n');
    %XHash(H2[DOMNAME]      ,MODEL.DOMVARS,multidata:'y');
    %XHash(H3[DOMNAME VARNAME/PVLID],MODEL.PVL,multidata:'n');

    RC1=ITER1.FIRST();
    DO WHILE (RC1=0);
```

```
        RC2= H2.FIND();
        DO WHILE(RC2 = 0);
          TMP1="";
          if compare(varorigin,"Assigned", "IL:")=0 then TMP1=%XCAT('
def:CommentOID="COM.@0.@0"',DOMNAME,VARNAME);
          TMP2="";
          if not missing(VARFMT) then TMP2=%XCAT(" def:DisplayFormat=@", VARFMT);
           TMP3="";
          if lowcase(VARTYPE)="float" then do;
              TMP3=%XCAT(" SignificantDigits=@", substrn(VARFMT,findc(VARFMT,'.')+1));
          end;

          %XOUT(T1, '<ItemDef OID="IT.@s0.@s0" Name=@ SASFieldName=@ DataType=@ Length=@ @t0 @t0
@t0>',
                     DOMNAME, VARNAME, VARNAME, VARNAME, VARTYPE, VARLEN, TMP1 ,TMP2 ,TMP3);

          %XOUT(T2, "<Description>");
          %XOUT(T3, "<TranslatedText xml:lang=""en"">@t0</TranslatedText>", %XESC(varlabel));
          %XOUT(T2, "</Description>");

          if not missing(VARCT) then do;
              %XOUT(T2, '<CodeListRef CodeListOID="CL.@s0"/>', VARCT);
          end;

          if compare(varorigin,"Predecessor", "IL:")=0 then do;
            %XOUT(T2, "<def:Origin Type=""Predecessor"">");
            %XOUT(T3, "<Description>");
            %XOUT(T4, "<TranslatedText xml:lang=""en"">@t0</TranslatedText>", %XESC(varcmnt));
            %XOUT(T3, "</Description>");
            %XOUT(T2, "</def:Origin>");
          end;else  do;
            if H3.check() ne 0 then do;
              %XOUT(T2, "<def:Origin Type=@/>",propcase(varorigin));
            end;
          end;

          if H3.check()=0  then do;
              %XOUT(T2, '<def:ValueListRef ValueListOID="VL.@s0.@s0"/>', DOMNAME, VARNAME);
          end;

          %XOUT(T1, "</ItemDef>");

          RC2 = H2.FIND_NEXT();
        END;
        RC1=ITER1.NEXT();
    END;

Run;
```

Example 3: Generating MethodDef And def:CommentDef Elements

A *MethodDef* element can be part of metadata about variables and values in a valuelist. It is must be provided if any variables or values are defined as Derived. Each MethodDef element must contain a child Description element. In addition, it can have one or multiple def:DocumentRef element that point to external files. Below are a couple of examples of *MethodDef* and *def:CommentDef* elements.

```
      <MethodDef OID="MT.ADQSADAS.AVISIT" Name="CM.ADQSADAS.AVISIT" Type="Computation">

        <Description>

          <TranslatedText xml:lang="en">Derived based on windowing algorithm described in SAP,
Section 8.2</TranslatedText>

        </Description>

      </MethodDef>
```

```
. . .

      <MethodDef OID="MT.ADSL.CUMDOSE" Name="CM.ADSL.CUMDOSE" Type="Computation">

        <Description>

          <TranslatedText xml:lang="en">For ARMN=0 or 1: CUMDOSE=TRT01PN*TRTDUR. --- For ARMN=2:
CUMDOSE will be based on 54mg per day for the # of days subj was in 1st dosing interval (i.e.,
visit4date-TRTSTDT+1 if 1st interval completed, TRTEDT-TRTSTDT+1 if subj discontinued &lt;=visit
4 and &gt; visit 3), 81mg per day for the # of days subj was in 2nd dosing interval (i.e.,
visit12date-visit4date if 2nd interval completed, TRTEDT-visit4date if subj discontinued &lt;=
visit 12 and &gt; visit 4), and 54mg per day for the # of days subj was in 3rd dosing interval
(i.e., TRTEDT - visit12date if subj continued after visit 12).</TranslatedText>

        </Description>

      </MethodDef>

. . .

      <MethodDef OID="MT.ADQSADAS.AVAL.ACTOT" Name="CM.ADQSADAS.AVAL.ACTOT" Type="Computation">

        <Description>

          <TranslatedText xml:lang="en">Sum of ADAS scores for items 1, 2, 4, 5, 6, 7, 8, 11, 12,
13, and 14, see Analysis Data Reviewers Guide (Page 3) for details on adjusting for missing
values.</TranslatedText>

        </Description>

        <def:DocumentRef leafID="LF.ADRG">

          <def:PDFPageRef PageRefs="3" Type="PhysicalRef"/>

        </def:DocumentRef>

      </MethodDef>

. . .

      <def:CommentDef OID="COM.ADSL">

        <Description>

          <TranslatedText>Screen Failures are excluded since they are not needed for this study
analysis</TranslatedText>

        </Description>

      </def:CommentDef>

      <def:CommentDef OID="COM.ADQSADAS">

        <Description>

          <TranslatedText>See referenced dataset creation program and Analysis Data Reviewer's
Guide, Section 2.1</TranslatedText>

        </Description>

        <def:DocumentRef leafID="LF.ADQSADAS.PGM"/>

        <def:DocumentRef leafID="LF.ADRG">

          <def:PDFPageRef PageRefs="Section2.1" Type="NamedDestination"/>

        </def:DocumentRef>

      </def:CommentDef>
```

The tricky part of implementing a MethodDef element is how to know whether there are one or more child def:DocuemntRef elements for additional or external documents to be attached. One of the solutions is to ask users to make a tag-like phrase for each of the document references at the end of each of method fields of the metadata Excel sheets. The tags are required to follow following pattern

<LF.*linkID*>

Where, the *linkID* is the leaf ID defined in the LINK dataset. The method variable is treated as a complex variable, and the subroutine getDocRefs() is used to extract all document references from the method fields for the creation of the child *def:DecumnetREF* elements. Below is the sample implementation with the DOMVARS and PVL datasets.

```
DATA MethodDef(keep=XMLLINE);
    LENGTH XMLLINE $8192 loc1 8;
    LENGTH num 8 trunc 8 idx 8 linktype $16 linkanchor $64;
    ARRAY  DOCREFS[32] $1024 _temporary_;

    if 0 then set MODEL.DOMVARS MODEL.PVL MODEL.LINKS;

    %XHash(H1:ITER1[DOMNAME VARNAME],MODEL.DOMVARS, multidata:'n');
    %XHash(H2:ITER2[DOMNAME VARNAME PVLID],MODEL.PVL, multidata:'n');
    %XHash(H3:ITER3[leafID],MODEL.LINKS, multidata:'n');

    RC1=ITER1.FIRST();
    DO WHILE (RC1=0);
      IF compare(VAROrigin, "Derived", "IL:")=0 THEN DO;
        num=0;trunc=0;

        Call GetDocRefs(varcmnt, docrefs, num, trunc);

        %XOUT(T1, '<MethodDef OID="MT.@0.@0" Name="CM.@0.@0" Type="Computation">',DOMName,
VARNAME ,DOMName, VARNAME);
        %XOUT(T2, "<Description>");
        %XOUT(T3, "<TranslatedText xml:lang=""en"">@t0</TranslatedText>",%XESC(VARCMNT));
        %XOUT(T2, "</Description>");

        DO IDX=1 TO NUM;
          linkID=docrefs[idx]; LeafID=substr(linkID, index(linkID,".")+1);

          if H3.check() ne 0 then do;
            %XOUT(T2, "<def:DocumentRef leafID=@/>", docrefs[idx]);
          end; else do;
            linktype="Page"; linkanchor="1";
            rc=H3.find();
            loc1=Findc(LeafHREF,"#","B");
            if loc1 > 0 then do;
                linkanchor=substr(LeafHref,loc1+1);
                if anyalpha(linkanchor) then do;
                  linktype="NameDestination";
                end;
            end;
            %XOUT(T2, "<def:DocumentRef leafID=@>", docrefs[idx]);
            %XOUT(T3, '<def:PDFPageRef PageRefs=@ Type=@/>',linkanchor, linktype);
            %XOUT(T2, "</def:DocumentRef>");
          end;
        END;

        %XOUT(T1, "</MethodDef>");
      END;

      RC1=ITER1.NEXT();
    END;

    RC2=ITER2.FIRST();
    DO WHILE (RC2=0);
      IF compare(PVLORIGIN, "Derived", "IL:")=0 THEN DO;
        num=0;trunc=0;
        Call GetDocRefs(PVLCMNT, docrefs, num, trunc);
```

```
        %XOUT(T1, '<MethodDef OID="MT.@0.@0.@0" Name="CM.@0.@0.@0" Type="Computation">',DOMName,
VARNAME, PVLID, DOMName, VARNAME, PVLID);
        %XOUT(T2, "<Description>");
        %XOUT(T3, "<TranslatedText xml:lang=""en"">@t0</TranslatedText>",%XESC(PVLCMNT));
        %XOUT(T2, "</Description>");

        DO IDX=1 TO NUM;
          linkID=docrefs[idx]; LeafID=substr(linkID, index(linkID,".")+1);
          if H3.check() ne 0 then do;
            %XOUT(T2, "<def:DocumentRef leafID=@/>", docrefs[idx]);
          end; else do;
            linktype="Page"; linkanchor="1";
            rc=H3.find();
            loc1=Findc(LeafHREF,"#","B");
            if loc1 > 0 then do;
                linkanchor=substr(LeafHref,loc1+1);
                if anyalpha(linkanchor) then do;
                  linktype="NameDestination";
                end;
            end;
            %XOUT(T2, "<def:DocumentRef leafID=@>", docrefs[idx]);
            %XOUT(T3, '<def:PDFPageRef PageRefs=@ Type=@/>',linkanchor, linktype);
            %XOUT(T2, "</def:DocumentRef>");
          end;
        END;

        %XOUT(T1, "</MethodDef>");
      END;

      RC2=ITER2.NEXT();
    END;

Run;
```

def:CommentDef is a new element introduced in Define-XML 2.0. It is intended to replace the deprecated ODM Comment attribute. Although it involves three datasets DOMAIN, DOMVARS, and PVL, it can be created with the same approach as MethodDef element. Therefore, its implementation will not be further discussed.

You can find many more examples from [2] on how to implement other CDSIC ADaM elements, and re-write them with DefKit utilities in a more simple way.

CONCLUSION

DefKit micro framework is designed with SAS programmers in mind. The inline macros and user-defined function and subroutines it provides can be easily used inside SAS data steps like a built-in function. They are non-intrusive, do not require users to create specific data structures in order to use it, and have no external dependencies. The idea behind them is to separate the specification of the business logic and computation required to generate CDISC/XML l elements from the specification of how these chunks of xml text is presented. Because of this, the DefKit brings following extra benefits:

- It simplifies the development of define-xml applications.
- It encourages modular programming through XML datasets, user-defined functions or subroutines.
- It separates XML templates from business logics specified in the Define-XML specifications.
- It encourages the development of reusable XML templates for a family of define-xml applications.
- It lets XML templates serve as clear documentation for the implementation of define-xml elements.

Especially, with the DefKit micro framework, typical define.xml generators for SDTM and ADaM models can be developed with following steps:

1. Create Excel files to collect Define-xml metadata for a clinical study using an OR model, which allows users to enter study metadata conveniently and collectively.

2. Convert the Excel metadata sheets into Define-xml meta datasets via PROC IMPORT facility.

3. Convert study datasets into .XPT files with XPORT library engine and PROC COPY.

4. Create individual XML datasets using %XCAT, %XOUT and %XHash, together with the user-defined functions associated with complex or compound variables.

5. Call %ToDEFXML to combine all the XML datasets generated and create the final DEFINE.XML file.

The approach and techniques proposed above will not only help you overcome many challenges in the implementation of CDISC define.xml generators, but also give you a new way to develop many other applications within the SAS® System.

REFERENCES

1. CDISC Define-XML Specification, V2.0, Mar., 2013 (http://www.cdisc.org/define-xml)
2. Lei Zhang, "*Implementation of DEFINE.XML Generators using SAS Hash Objects and User-Defined Functions*", WUSS 2013.
3. CDISC Specification for the Operational Data Model (ODM), V1.3, Dec. 2006 (http://www.cdisc.org)
4. Lex Jansen, "*Understanding the define.xml File and Converting It to a Relational Database*" SAS Global Forum 2010.
5. Paul Brown, "*Object-Relational Database Development: A Plumber's Guide*", Prentice Hall, 2001.
6. Yves Deguire & Xiyun Wang, "*Using SAS PROC FCMP in SAS System Development – Real Examples*", SAS Global Forum 2013
7. Paul M. Dorfman & Koen Vyverman, "*Data Step Hash Objects as Programming Tools*", SUGI 30, 2005.

CONTACT INFORMATION

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

Lei Zhang
Celgene Corporation
Summit, NJ, USA
E-mail: lezhang@celegene.com

Appendix: DefKit Framework Source Code

1. DefKitMacs.sas

```
%Macro DefKitMacs;

/*****************************************************************************
    %XCAT() -  Create a XML string  via a line template.
 *****************************************************************************/
%Macro XCAT/PARMBUFF;

  %local paramlst param loc explst len paramlen;
  %local catexp VAREXPS TEMPLATE;
  %local MOD regid kviterm ch stpch quch dqch blk;
  %local news repword count found rawfound;
  %local start oldstart stop position oldposition length;
  %local RegexID val tmpl xlen xch;

  %let paramlst=;
  %if %length(&syspbuff) > 2 %then %do;
    %let paramlst=%qsubstr(&syspbuff,2,%length(&syspbuff)-2);
    %let paramlst=%qsysfunc(left(&paramlst));
  %end;

  /* Get the first parameter Template */
  %let param=%sysfunc(scan(&paramlst,1,%str(,),q));

  %let len=%length(&paramlst);
  %let paramlen=%length(&param);

  %let template=;
```

```
%let explst=;
%if &paramlen > 1 %then %do;
  %let template=&param;
  %if &len > &paramlen +1 %then %do;
    %let explst=%qsubstr(&paramlst,&paramlen+1);
  %end;
%end; %else %do;
  %let template=;
%end;

%if %index(%nrbquote(&explst),%str(,))=1 %then %let explst=%qsubstr(%nrbquote(&explst),2);

%let VAREXPS=;
%if %length(&explst) %then %do;
  %let VAREXPS=%enpkg(%unquote(&explst));
%end;

%let catexp=;

%if %length(&TEMPLATE) = 0 %then %do;
  %let catexp="";
  %goto quitbye;
%end;

%let TEMPLATE=%qsysfunc(dequote(&TEMPLATE));

%if %length(&VAREXPS) = 0 %then %do;
  %let catexp=%qsysfunc(quote(&TEMPLATE));
  %goto quitbye;
%end;

/* Perform the replacement */
%let tmpl=%qsysfunc(dequote(&TEMPLATE));
%let RegexID = %sysfunc(prxparse(/@[st]?[012]?/i));

%let news=; /* New String */
%let count=0;

%let start = 1;
%let oldstart=1;
%let stop = %length(&tmpl);

%let position=0;
%let length=0;

%syscall prxnext(RegexID, start, stop, tmpl, position, length);

%do %while (&position > 0);
       %let count=%eval(&count+1);

       %let rawfound =%qsubstr(&tmpl, &position, &length);

       /* get the sub-commands after @ */
       %let found=%qsubstrx(&rawfound,2);

       /* get the ith expression */
       %let val=%qsysfunc(scan(&VAREXPS,&count,%str( ),q));
       %if %length(&val) %then %do;
          %let val=%qsysfunc(dequote(&val));
       %end; %else %let val=;

       %if %length(&val) > 0 %then %do;
        %let xlen=%length(&found);
        %if &xlen=0 %then %do;
          %let repword=catq("aqs2",&val);
        %end; %else %Do;
          %if &xlen=1 %then %do;
              %let xch=&found;
              %if &xch=0 %then %do;
                  %let repword=strip(&val);
              %end; %if &xch=1 %then %do;
                  %let repword=catq("aqs1",&val);
              %end;%else %if &xch=2 %then %do;
                  %let repword=catq("aqs2",&val);
              %end;%else %if &xch=s %then %do;
                  %let repword=catq("aqs2",&val);
              %end;%else %if &xch=t %then %do;
                  %let repword=catq("aqt2",&val);
              %end;
          %end; %else %do;
              %let xch=%substr(&found,1,1);
```

```
                %if &xch=s %then %do;
                    %let repword=strip(&val);
                %end; %if &xch=t %then %do;
                    %let repword=trimn(&val);
                %end;

                %let xch=%substr(&found,2,1);
                %if &xch=1 %then %do;
                    %let repword=catq("aq1",&repword);
                %end; %if &xch=2 %then %do;
                    %let repword=catq("aq2",&repword);
                %end;
             %end;
          %end;
         %end; %else %let repword=%sysfunc(quote(&rawfound));

         %if &count=1 %then %do;
              %if &position > &oldstart %then %do;
                %let news=%sysfunc(quote(%qsubstrx(&tmpl, &oldstart, &position-&oldstart)))%str(,)&repword;
              %end; %else %let news=&repword;
         %end; %else %let news=&news%str(,)%sysfunc(quote(%qsubstrx(&tmpl, &oldstart, &position-
&oldstart)))%str(,)&repword;

         %let oldstart=&start;
         %let oldposition=%eval(&position+1);
         %syscall prxnext(RegexID, start, stop, tmpl, position, length);
    %end;

    %if %length(&tmpl)>=&oldstart %then %do;
      %if &count > 0 %then %do;
        %let news=&news%str(,)%sysfunc(quote(%qsubstrx(&tmpl, &oldstart, %length(&tmpl)-&oldstart+2)));
      %end; %else %let news=&tmplx;
    %end;%else %if &count=0 %then %let news=&tmplx;
    %let catexp=cat(&news);

%quitbye:
%unquote(&catexp)
%Mend;

/*****************************************************************************
  %XOUT:  Create a XML string via a line template and store it in a XML dataset.
*****************************************************************************/
%Macro XOUT/PARMBUFF;
  %local paramlst paramlst1 kviterm outvar outdsn indent nindent template;
  %local regid loc idx len;
  %local indentcmd ch mod;

/* decompose the key/value parameters */
  %let paramlst=;
  %if %length(&syspbuff) > 2 %then %do;
    %let paramlst=%qsubstr(&syspbuff,2,%length(&syspbuff)-2);
    %let paramlst=%qsysfunc(left(&paramlst));
  %end; %else %do;
    %let paramlst=;
    %let outvar=;
    %let outdsn=;
    %let indent=;
    %goto codegen;
  %end;

  /* Extract the key=value parameters */
  %let regid=%sysfunc(prxparse(/%str(,)\s*\w+\s*=\s*(\w+)?$/i));
  %let loc=%sysfunc(prxmatch(&regid,%qsysfunc(trimn(&paramlst))));
  %Do %While(&loc>1);
    %let kviterm=%qsubstr(&paramlst,&loc+1);
    %let kviterm=%qsysfunc(compress(&kviterm));
    %if %sysfunc(compare(&kviterm,OUTVAR=,il:))=0 and %length(&OUTVAR)=0 %then %do;
        %if %length(&kviterm)>=8 %then %do;
          %let OUTVAR=%substr(&kviterm,8);
        %end;
    %end; %else %if %sysfunc(compare(&kviterm,OUTDSN=,il:))=0 and %length(&OUTDSN)=0 %then %do;
        %if %length(&kviterm)>=8 %then %do;
          %let OUTDSN=%substr(&kviterm,8);
        %end;
    %end;

    %let paramlst=%qsubstr(&paramlst,1,&loc-1);
    %let loc=%sysfunc(prxmatch(&regid,%qsysfunc(trimn(&paramlst))));
  %end;

  %let len=%length(&paramlst);
```

```
  %let loc=%index(&paramlst,%str(,));
  %if &loc>1 %then %do;
       %let Indent=%substr(&Paramlst,1,&loc-1);
       %if &len > &loc %then %let paramlst=%substr(&paramlst,&loc+1);
       %else %let paramlst=;
  %end; %else %do;
      %let Indent=;
      %if &loc=1 and &len>1 %then %let paramlst=%substr(&paramlst,2);
      %else %let paramlst=;
  %end;

%codegen:
  %let indentcmd=;
  %if %length(&INDENT) %then %do;
    %let ch=%upcase(%substr(&INDENT,1,1));
    %let nindent=%substr(&Indent,2);
    %let indentcmd=;
    %if &ch=T %then %do;
      %if %eval(&nindent) > 0 %then %do;
        %let indentcmd=repeat(Byte(9),%eval(&nindent -1));
      %end;
    %end; %else %if &ch=S %then %do;
      %if %eval(&nindent) > 0 %then %do;
        %let indentcmd=repeat(' ',%eval(&nindent -1));
      %end;
    %end;
  %end;

  %if %length(&OUTVAR)=0 %then %let OUTVAR=XMLLINE;

  %if %length(&indentcmd) > 0 %then %do;
    &OUTVAR=&indentcmd||%XCAT(%unquote(&paramlst));OUTPUT &OUTDSN;
  %end; %else %do;
    %if %length(&paramlst) > 0 %then %do;
      &OUTVAR=%XCAT(%unquote(&paramlst));OUTPUT &OUTDSN;
    %end; %else %do;
      &OUTVAR=%XCAT( );OUTPUT &OUTDSN;
    %end;
  %end;
%MEND;

/*********************************************************************
     %XESC: Escape the XML special characters in a string
*********************************************************************/
%Macro XESC(
    varexp         /* variable or expression     */
);
TRIMN(TRANWRD(TRANWRD(TRANWRD(TRANWRD(TRANWRD(TRANWRD(&varexp, '&', '&amp;'),"<", '&lt;'),">", '&gt;'),"'",
'&apos;'),'"', '&quot;'),'0a'x, '&#xA;'))
%Mend;

/*************************************************************************
      %Enpkg: packing parameters using a quoted space-delimiter string (local)
*************************************************************************/
%Macro Enpkg(p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15,p16,p17,p18,p19,p20,p21,p22,p23,p24);
%local pkg idx;

%let pkg=;
%let idx=1;
%Do %while(%length(&&p&idx)>0 AND &idx<16);
  %if &idx=1 %then %let pkg=%qsysfunc(quote(&&p&idx));
  %else %let pkg=&pkg %qsysfunc(quote(&&p&idx));
  %let idx=%eval(&idx+1);
%End;
&pkg
%Mend;

/*************************************************************************
   %XHash: Define and instantiate a hash object.
*************************************************************************/
%Macro XHash(
  HasHexp          /*  Hassh-nanme<:Hash-iterator>[key1 key2 . . . /val1 val2 . . .]      */
  ,DSNEXP          /*  DSN with common dataset options such as keep=, rename= and where=  */
  ,HashOpts        /*  Hash options such as multidata:'yes'                               */
);

%local Hashnm HashExp1 Iternm DSNnm DSNnm1 VARlst Keylst1 Datalst1  vname vtype vlen;
%local exp exp2 exp3 exp4 exp5 exp6 exp7;
%local loc loc1 idx item item_p1 Item_p2 HashOpts1;
%local Objname Objtype ObjParams _DSNEXP _Viewdef _CodeDef _OUTDSN;
```

```
%if %length(&HashExp) = 0 OR %length(&DSNExp) = 0 %then %do;
  %put ERROR: No hash expression or dataset expression are provided.;
  %goto quitbye;
%end;

/* Normalize the hash options */
%let Hashopts1=%NormHashOpts(&hashOpts);

/* Parsing the Hash expression */
%let Hashnm=;
%let Iternm=;
%let Keylst1=;
%let Datalst1=;

%let loc=%index(&HashExp,%str([));
%if &loc = 0 %then %do;
  %let HashExp1=&Hashexp;
  %let Keylst1=%GetVarsFromDSN(&DSNEXP);
  %let Datalst1=&Keylst1;
%end; %else %do;
  %let HashExp1=%substr(&Hashexp,1,&loc-1);
  %let item=%substr(&Hashexp, &loc+1, %length(&HashExp)-&loc -1);
  %let loc1=%index(&item,%str(/));
  %if &loc1 > 0 %then %do;
      %let Keylst1=%substr(&item,1,&loc1-1);
      %if &loc1 < %length(&item) %then %let Datalst1=%substr(&item,&loc1+1);
  %end; %else %do;
      %let Keylst1=&item;
  %end;
%end;

%if %length(&Datalst1)=0 %then %let Datalst1=%GetVarsFromDSN(&DSNEXP);
%let loc1=%index(&HashExp1,:);
%if &loc1 > 0 %then %do;
      %let Hashnm=%substr(&HashExp1,1,&loc1-1);
      %if &loc1 < %length(&HashExp1) %then %let Iternm=%substr(&HashExp1,&loc1+1);
%end; %else %do;
      %let Hashnm=&HashExp1;
%end;

%let _DSNEXP=dataset:%sysfunc(quote(%superq(DSNEXP)));

/* Create Hash object and associated iterator */
%if %length(&HashOpts1) %then %do;
  %if %length(&_DSNEXP) %then %do;
    %let exp2=DECLARE HASH &Hashnm(&_DSNEXP,&HashOpts1);
  %end; %else %do;
    %let exp2=DECLARE HASH &Hashnm(&HashOpts1);
  %end;
%end; %else %do;
  %let exp2=DECLARE HASH &Hashnm(&_DSNEXP);
%end;

%if %length(&ITERnm) %then %do;
  %let exp3=DECLARE HITER &ITERnm(%sysfunc(catq(a1,&Hashnm)));
%end;%else %do;
  %let exp3=;
%end;

%let exp4=&HashNM..defineKey(%InsComma(&Keylst1,1));

%let exp5=;
%If %length(&datalst1) %then %do;
  %let exp5=&HashNM..definedata(%inscomma(&Datalst1,1));
%end;

&exp2;
%if %length(&exp3) %then %do;
&exp3;
%end;

___DefHashRC=&exp4;
%if %length(&exp5) %then %do;
   ___DefHashRC=&exp5;
%end;
___DefHashRC=&Hashnm..DefineDone();

%quitbye:
%Mend;

/*****************************************************************************
```

```
   %TODEFXML: Create a DEFINE.XML file from a list of XML datasets.
*******************************************************************************/
%MACRO TODEFXML(
  DSNLST               /* A list of XML datasets (required)                    */
  ,OUTDIR=             /* Output directory for the XML file (required)         */
  ,OUTFILE=DEFINE.XML  /* Define file name, default is DEFINE.XML (required)   */
  ,LINESIZE=8192       /* Line size. Default is 8192                           */
  ,OPTS=               /* Options:                                             */
                       /*    ODMEnd:  Close the XML file with ODM ending tags  */
);

DATA _NULL_;
  LENGTH XMLLINE $&LINESIZE _len_  8.  _ERRMSG_ $1024;
  file "&OUTDIR\&OUTFILE" notitles lrecl=&LINESIZE;
  set &DSNLST end=last;
  _len_=lengthn(trimn(XMLLINE));
  if _len_ > &LINESIZE then do;
    _ERRMSG_=%XCAT("ER%str(ROR): the length of XML statement (=@t0)is longer than &LINESIZE.", _len_);
    Put _ERRMSG_;
    _len_=&LINESIZE;
  end;
  put XMLLINE $varying&LINESIZE.. _len_;

  %if %sysfunc(compare(&OPTS,ODMEND,IL:))=0 %then %do;
  if last then do;
      XMLLINE=repeat(byte(9),2)||"</MetaDataVersion>";_len_=lengthn(XMLLINE);put XMLLINE $varying&LINESIZE..
_len_;
      XMLLINE=repeat(byte(9),1)||"</Study>";_len_=lengthn(XMLLINE);put XMLLINE $varying&LINESIZE.. _len_;
      XMLLINE="</ODM>";_len_=lengthn(XMLLINE);put XMLLINE $varying&LINESIZE.. _len_;
  end;
  %end;
Run;
%MEND;

/*******************************************************************************
    %GetVarsFromDSN: Get the dataset variables (local)
*******************************************************************************/
%Macro GetVarsFromDSN(
        DSN     /* Dataset variable list with print optiosn:      */
                /* It has following format:                       */
                /* LIB.DSN                                        */
)
/des="Get all the varables of a dataset"
;

%local dsid rc nvar idx vname vtype vlist;

%if %length(&DSN) = 0 %then %do;
        %let DSN=&syslast;
        %if %upcase(&DSN) = _NULL_ %then %goto quitx;
%end;

%let dsid= %sysfunc(OPEN(&dsn,is));

%if &dsid = 0 %then %do;
       %let rc= %sysfunc(CLOSE(&dsid));
       %put Can%str(%')t open dataset %upcase(&DSN);
       %goto quitx;
%end;

%let VLIST=;
%let NVARS=%Sysfunc(ATTRN(&dsid, NVARS));

%Do idx=1 %to &NVARS;
    %let VNAME=%sysfunc(VARNAME(&DSID, &IDX));
    %let VLIST=&VLIST &VNAME;
%End;

%Let rc= %sysfunc(CLOSE( &dsid ));

%quitx:
&VLIST
%Mend;

/*******************************************************************************
  %InsComma: quote a list of items and insert a comma between terms.
*******************************************************************************/
%Macro InsComma(
    varlst        /*  SAS variable list                                      */
    ,opts         /*  option.                                                */
                  /*     BLANK --  no quotation marks  added to the variables */
```

```
                  /*         1 --  add single equotation mark                    */
                  /*         2 --  add double equotation mark                    */
);

%local rtnlst;
%let rtnlst=;

%if %length(&varlst)=0 %then %return;

%if %length(&opts)=0 %then %do;
  %let rtnlst=%sysfunc(translate(%sysfunc(compbl(&varlst)),%str(,), %str( )));
%end; %else %do;
  %if %index(&opts,1) > 0 %then %do;
    %let rtnlst=%sysfunc(catq(a1c,%sysfunc(translate(%sysfunc(compbl(&varlst)),%str(,), %str( )))));
  %end; %else %if %index(&opts,2) > 0 %then %do;
    %let rtnlst=%sysfunc(catq(a2c,%sysfunc(translate(%sysfunc(compbl(&varlst)),%str(,), %str( )))));
  %end; %else %do;
    %put WARNING: wrong option <&opts>. It is treated as a blank.;
    %let rtnlst=%sysfunc(translate(%sysfunc(compbl(&varlst)),%str(,), %str( )));
  %end;
%end;
&rtnlst
%Mend;

/*****************************************************************************
    %NorMHashOpts: Normalize the hash options
*****************************************************************************/
%Macro NormHashOpts(
    HASHOPTS      /* Hash options used in the hash definition       */
    ,OPTS=        /* Options (reserved)                             */
);

%local objopts objopts1 idx item item_p1 Item_p2 ;
%let objopts=&&HashOPTS;
%if %length(&objopts)=0 %then %return;

%let ObjOpts=%compact(&ObjOpts,:);

%let idx=1;
%let item=%sysfunc(scan(&ObjOpts,&idx,:,Q));

%do %while(%length(&item));
  %if &idx=1 %then %let ObjOpts1=&item;
  %else %let ObjOpts1=&ObjOpts1%str(:)&item;

  %let idx=%eval(&idx + 1);
  %let item=%sysfunc(scan(&ObjOpts,&idx,:,Q));
%end;

%let objOpts=&objopts1;

%let idx=1;
%let item=%sysfunc(scan(&ObjOpts,&idx,%str( ),Q));

%do %while(%length(&item));
  %if &idx=1 %then %let ObjOpts1=&item;
  %else %let ObjOpts1=&ObjOpts1%str(,)&item;

  %let idx=%eval(&idx + 1);
  %let item=%sysfunc(scan(&ObjOpts,&idx,%str( ),Q));
%end;
%unquote(&ObjOpts1)
%Mend;

%Mend;
```

2. DefKitUDFs.SAS

```
%LET PGMPATH=%SYSGET(SAS_EXECFILEPATH); /* FOR WINDOWS SYSTEM ONLY */
%LET PGMDIR=%SUBSTR(&PGMPATH,1, %SYSFUNC(FINDC(&PGMPATH,\, -%LENGTH(&PGMPATH)))-1);

LIBNAME DefKit "&PGMDIR";
options cmplib=(DefKit.DefKit);
PROC FCMP outlib=DefKit.DefKit.string;

  /*****************************************************************************
      GetRangeChecks() -- a subroutine to extract document reference links
                          from a comment variable
```

```
*******************************************************************************/
SUBROUTINE GetRangeChecks(
   DOC $
  ,Comparator[*] $
  ,Entry[*] $
  ,n
  ,trunc
);
  outargs Comparator, Entry, n, trunc;

  Length RegexID 8 Text $1024 found $32  count 8 maxrows 8 loc1 8 loc2 8;
  Length found $1024;

  maxrows=dim(entry);
    RegexID = prxparse('/RangeCheck\s*\d*\s*:/im');
    start = 1;
    stop = lengthn(DOC);
  prev_loc=0; count=0;n=count; trunc=0;
  if stop=0 then return;
    call prxnext(RegexID, start, stop, DOC, position, length);
      do while (position > 0);
      if prev_loc > 0 then do;
        count=count+1;
      if count > maxrows then do; trunc=1; count=ccount-1;end;
      else do;
            found = strip(substrn(DOC, prev_loc+1, position - prev_loc -1));
        loc1=findc(found,'['); loc2=findc(found,']', -lengthn(found));
        if loc1 ne 0 and loc2 ne 0 then do;
          Comparator[count]=substrn(found,1,loc1-1);
          Entry[count]=substrn(found, loc1+1, loc2-loc1-1);
        end; else do;
          Comparator[count]="";
          Entry[count]=found;
        end;
      end;
    end;

    prev_loc=position+length-1;
        call prxnext(RegexID, start, stop, DOC, position, length);
      end;

  found='';
  if (prev_loc > 0) then do;
      found = strip(substrn(DOC, prev_loc+1, stop-prev_loc));
  end; else if count=0 then do;
    found=doc;
  end;

  if lengthn(found) > 0 then do;
    count=count+1;
    if count > maxrows then do; trunc=1; count=count-1; end;
    else do;
      loc1=findc(found,'['); loc2=findc(found,']', -lengthn(found));
      if loc1 ne 0 and loc2 ne 0 then do;
        Comparator[count]=substrn(found,1,loc1-1);
        Entry[count]=substrn(found, loc1+1, loc2-loc1-1);
      end; else do;
        Comparator[count]="";
        Entry[count]=found;
      end;
    end;
  end;
  n=count;
ENDSUB;

/*******************************************************************************
    GetDocRefs() -- a subroutine to extract comparators from a range check variable
*******************************************************************************/
SUBROUTINE GetDocRefs(
   DOC $
  ,LeafRef[*] $
  ,n
  ,trunc
);

  outargs LeafRef, n, trunc;

  Length RegexID 8 Text $1024 found $1024  count 8 maxrows 8 loc1 8 loc2 8;

  maxrows=dim(LeafRef);
    RegexID = prxparse('/\<\s*LF\..+?\>/im');
```

```
      start = 1;
      stop = lengthn(DOC);
    prev_loc=0; count=0;n=count;trunc=0;
    if stop=0 then return;

      call prxnext(RegexID, start, stop, DOC, position, length);
        do while (position > 0);
        count=count+1;
      if count > maxrows then do; trunc=1; count=count-1; end;
      else do;
            found = substr(DOC, position+1, length-2);
        LeafRef[count]=found;
            call prxnext(RegexID, start, stop, DOC, position, length);
      end;
    end;

    n=count;
  ENDSUB;

RUN;QUIT;
```