Paper 021-2013

The Hash-of-Hashes as a "Russian Doll" Structure: An Example with XML Creation

Joseph Hinson, Princeton, NJ, USA

ABSTRACT

SAS®9 hash objects have inspired novel programming techniques. The recent discovery that hash tables can contain even other hash objects: “hash of hashes” opens the door to their application to hierarchical data processing. Because hierarchies, like Russian dolls, can be considered "containers within containers.” Thus, nested hash objects could model XML, a hierarchical data structure increasingly finding its way into clinical trial data. Clinical programmers now have to deal with hierarchical as well as tabular and relational data sets. SAS® now provides tools like the XML libname engine and XML mapper. This paper aims to show, using a simplified CDISC LAB model, that the hash object could well be another tool for creating XML.

INTRODUCTION

Hierarchical data structures are ubiquitous, even in today’s world dominated by flat relational databases. The world is currently seeing a proliferation of XML-based documents in many sectors: financial, multimedia, telecommunication, publishing, health, and library science, to name a few. The pharmaceutical industry's clinical trial divisions are particularly undergoing such a transformation, with the Clinical Data Interchange Standards Consortium (CDISC) systematically developing several XML-based models for the electronic acquisition, submission, and archival of clinical trial data. Models such as the Operational Data Model (ODM), the Study Design Model (SDM), and the Laboratory Data Model (LAB) are all in XML format. In fact, the CDISC's long-term desired outcome is the utilization of the XML format to establish a holistic approach to standards, facilitating data interchange between sponsor sites and the regulatory bodies. The US Federal Drug Administration (FDA) is also joining the bandwaggon, proposing the replacement of the SAS® version 5 transport file format with an XML format, as well as the provision of an XML-based metadata document called "define.xml", along with schemas and stylesheets, all XML documents. Thus increasingly, the XML structure is becoming very important for clinical trial data processing and submission. This poses quite a challenge to the SAS® programmer's thought process, which is most familiar with the data set, a relational and tabular structure of observations and columns (variables), also seen in RDBMS tables. In a typical SAS® data set, every observation has the same fields carrying the same attributes. By contrast, data structures like XML are tree-like, with multilevel nested information. A data item in an XML structure relates to other data items by the "ancestors", "siblings", and "descendants" that they share. To retrieve a data item, one has to literally traverse all of its ancestors, much unlike a traditional SAS® data set where one can fetch a data item by simply accessing the key fields on the same observation. SAS® Institute has been aware of the "XML revolution" taking place at CDISC and FDA and has already developed a number of XML-processing tools, notably: the XML libname engine, the XML mapper, the CDISC procedure, SAS® Clinical Standards Toolkit, and the Clinical Data Intergration Studio.

What has not been apparent is that the SAS® hash objects, made available with version 9, can also be used to process hierarchical data like XML. As will be shown in this paper, this is possible because of (a) the unique ability of hash objects to contain other hash objects, a situation analogous to a Russian doll structure, and (b) hierarchical data, like XML, can be considered “containers within containers”.

HASH OBJECTS

SAS® currently provides three sets of DATA Step Component Objects: hash and hash iterator objects, Java objects, and logger and appender objects. These "component objects" are data elements consisting of attributes, methods, and operators. They are handled by the Data Step Component Interface within the DATA step and therefore can only be used inside a DATA step.

The Hash-of-Hashes Technique:

Ever since the introduction of hash objects programming in version 9, the technique has been popular in SAS® programming tasks in which fast table look-ups and sortless merging are advantageous. But new uses of the technique keep appearing. For instance the present author had demonstrated the use of hash objects for table transposition[1]. However the real surprising application came about when a few years ago, Richard DeVenezia revealed that SAS® hash objects could contain other hash objects. Until then, it was assumed that hash tables could only contain numeric and character data. With a very elegant example, data set splitting[2], DeVenezia and Paul Dorfman demonstrated that indeed hash tables could contain references to other hash objects as data. This fact has

successfully been exploited in very complex coding tasks, such as (in order of increasing complexity), the splitting of data sets into multiple files[2], the analyses of stock holdings[3], and the processing of healthcare claims[4]. In the data set-splitting application where distinct groups in a data set were channeled into new tables, one did not have to know in advance the number of new data sets required, unlike data set creation with the DATA statement alone or Proc SQL by itself. And even though the use of the popular technique called the Do Loop of Whitlock (or "DOW") could have achieved the same splitting, the DOW method requires prior sorting and BY processing for initial grouping.

The gem of the hash-of-hashes technique is the ability to use dynamic instantiation of hash objects (using the operator " *_new_ hash()* ") to insert hash objects into other hash objects repeatedly, reusing existing hash objects. Since in a hash table, a key is associated with a data item, the same holds true when that data item is another hash object. As a hash key changes value, a new hash object associated with the new key is instantiated on the fly. Such multiple instantiations allow data to be compartmentalized in a nested fashion, associating objects with unique key values, leading to hierarchical grouping.

The Hash-of-Hashes as a Hierarchical Structure:

The present author has extended the notion of putting hash objects in other hash objects to hierarchical data processing in general. After all, data hierarchies can be considered as “containers” within other “containers” (Figure 1), analogous to Russian Dolls, where bigger dolls contain smaller dolls, which also contain even smaller dolls. Thus, a CONTINENT group can contain COUNTRIES, and each COUNTRY group can contain STATES, which can further contain CITIES.

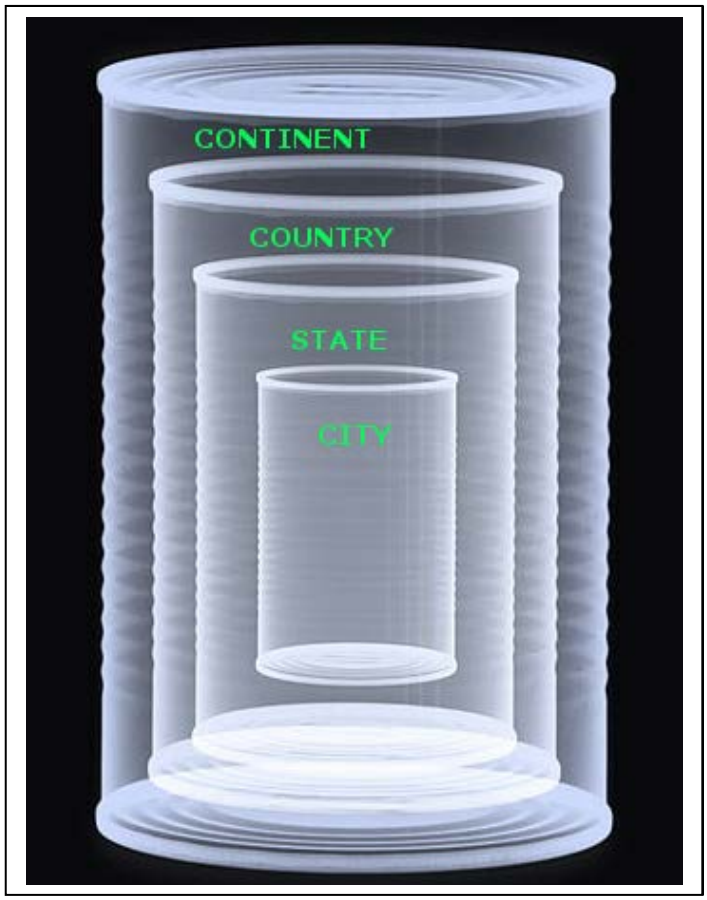

Figure 1. Data hierarchies as containers within containers

The multilevel nesting of hash objects is based on unique hash keys. For instance, to make a hash object, `geo`, contain another hash object `state` based on country, the following process is followed:

```
geo.defineKey("country");
geo.defineData("country", "state");
geo.defineDone();
```

The hash object `geo` has the key country and another hash object, `state`, as data.Thus each different country would have its own table of states kept in the `state` hash object:

Hash Key (`"country"`) →	USA	Canada	UK
	`state` object contents	`state` object contents	`state` object contents
	VERMONT	QUEBEC	SCOTLAND
	FLORIDA	ONTARIO	IRELAND
	TEXAS	ALBERTA	WALES
	CALIFORNIA		ENGLAND
	ILLINOIS		

The individual **`state`** objects could also contain **`city`** objects using **`state`** as the key, and a **`city`** hash as data. Such that, key=ENGLAND would have a hash object containing *London, Liverpool, Bristol*, whereas key=SCOTLAND would have an object containing *Edinburgh, Glasgow, Aberdeen*, and key=WALES would be associated with a hash object containing *Cardiff and Swansea*.

In summary, the hash-of-hashes uses the hash key to segregate groups into other hashes, also containing new hashes with their own key-related groupings, and the process of making certain hashes contain other hash objects creates the hierarchy, and the hierarchical order determined by the order of hash definitions:

```
cont.defineKey("continent");
cont.defineData("continent", "cntry");
      - -
      - -
cntry.defineKey("country");
cntry.defineData("country", "sta");
      - -
      - -
sta.defineKey("state");
sta.defineData("state", "cit");
      - -
      - -
cit.defineKey("city");
cit.defineData("city", "population_data");
```

XML AS A HIERARCHICAL DATA STRUCTURE

XML stands for eXtensible Markup Language and its documents are made up of strings of characters, divided into markup and content. A markup, also called a **tag**, begins with **<** and ends with **>**. There are three kinds:

start-tags: **<word>**
end-tags: **</word>**
empty-element tags: **<word />**

The sequence start-tag, content, end-tag, together form an *element*: for example,

<city> Houston </city>

This element can also be written longitudinally as:

<city>
Houston
</city>

So a typical geographical XML document would appear as:

```
<CONTINENT>
     North America
       <COUNTRY>
         USA
          <STATE>
            Texas
             <CITY>
               Houston
                <POPULATION>
```

```
                    2 million
                </POPULATION>
            </CITY>
        </STATE>
    </COUNTRY>
</CONTINENT>
```

demonstrating the hierarchical presentation of elements: *the population of cities within states within countries within a continent.*

HOW THE HASH-OF-HASHES CAN MODEL THE XML STRUCTURE

One could consider the XML structure as "containers within containers", with the base element CONTINENT, being the outermost "container". CONTINENT "contains" the COUNTRY vessel, also "containing" the STATE vessel, and so on. In a similar fashion, one could conceive of a CONTINENT hash table containing a COUNTRY object filled with a STATE object which itself contains a CITY object carrying population data per city. This is the basis for the use of hash-of-hashes to convert flat, 2-dimensional tables into hierarchical data structures like XML.

PROGRAMMING HASHES-OF-HASHES TO CREATE AN XML STRUCTURE FROM A SAS DATA SET

The hash-of-hashes algorithm for XML is in three steps:

(1) Set up hash objects according to the desired hierarchy of variables.

(2) Input flat data into hash objects causing the values to be channeled into nested groups, as determined by hash keys and the type of objects in the hash table:

(3) Use the hash iterators to traverse each hash objects in the hash-of-hashes, starting from the outermost “container” all the way to the innermost "container", issuing PUT commands to lay out XML elements:

I. APPLICATION TO A SIMPLIFIED CDISC LAB MODEL

The hash-of-hashes technique can be applied to laboratory data to create a LAB model document.

The XML-based LAB model was developed by CDISC for the transfer of clinical lab data from vendors to sponsors.

Because the objective of this section is just to demonstrate the concept of creating hierarchies with hash-of-hashes, the section uses limited data extracted from the LAB model BaseSampleData.xml[6] , with several elements and attributes removed for simplicity, leaving just ***Study***, ***Site***, ***Subject***, ***Visit***, ***BaseBattery***, ***BaseTest***, and ***SingleResult*** elements. The programming steps for transforming the lab data set into an XML document are as follows:

- The list of variables in the flat input data can be in any order,

```
input  Subject $ 1-4 Visit $ 6-7 BaseTest $ 9-22 SingleResult $ 24-27
BaseBattery $ 29-38 Site $ 40-41 Study $ 43-47;
```

  since the hierarchical order is determined by how the variables are listed in the hash objects and which object contains what object.

- To facilitate, the order is established with an enumerated list of macro variables:

```
%let element1=Study;
%let element2=Site;
%let element3=Subject;
%let element4=Visit;
%let element5=BaseBattery;
%let element6=BaseTest;
%let element7=SingleResult;
```

1. CREATION OF HASH-OF-HASHES:

a. Creation of the root element hash object:

The root element hash, hash1, is made to contain element1 data as well as hash2 object and hiter2 for the next element in the hierarchy, element2:

```
hash1.defineKey("&element1");
hash1.defineData("&element1","hash2","hiter2");
```

b. The next step involves a key process in hash-of-hashes programming, *dynamic instantiation of hashes* using the "_new_" operator, as data are read in:

```
declare hash hash2;   ←----hash declaration without instantiation
declare hiter hiter2; ←----hiter declaration without instantiation

do until(done);
set simplelabdata end=done;
if hash1.find() ne 0 then ←-----------check to see if hash2 and
                            hiter2 exist in hash1, else create it
do;
      hash2=_new_ hash();
      hash2.defineKey("&element2");
      hash2.defineData("&element2","hash3","hiter3");
      hash2.defineDone();
      hiter2=_new_ hiter("hash2");
      rc1=hash1.replace();
end;
```

c. The process is continued for all 7 element hashes, the last element hash containing no hash or hiter.

2. **FORMATION OF XML DOCUMENT:**

This involves reading out the data in all the hashes within hashes, using the iterator objects, and reading according to the position in the hierarchy. When a hiter method is activated, a row of hash table data is made available. The data is used by the sub-hashes to select their appropriate row of data. Thus, all the hierarchical data are obtained in the proper order, leading to the correct XML document. Small macros are used to handle the tags and text output to the XML file:

```
rc=hiter1.first();

do until (hiter1.next() ne 0);
val=&element1;
element=vname(&element1);
%tagon(element, val);

do while (hiter2.next() eq 0);
val=&element2;
element=vname(&element2);
%tagon(element, val);

do while (hiter3.next() eq 0);
      -
      -
```

The process is continued for all the 7 element hashes. End tags are also handled by macros.

```
element=vname(&element1);
%tagoff(element);
```

3. **XML OUTPUT:**

Figure 2. Simple LAB model XML showing a hierarchy of elements generated with the Hash-of-Hashes Technique

II. APPLICATION TO COMPLEX CDISC LAB MODEL

(WHERE ELEMENTS CONTAIN ATTRIBUTES)

Clinical data in XML format are far more intricate than the previous examples in this paper. For instance the actual CDISC LAB model has more elements as well as the presence of a variety of attributes associated with the elements and sub-elements.

The root element for the LAB XML document is named "Good Transmission Practice" (GTP), which provides

information about either the transmission as a whole or a particular record within it.

The model has 12 key elements:available:

1. GoodTransmissionPractice
2. Study
3. Site
4. Investigator
5. Subject
6. Visit
7. Accession
8. RecordExtensionType
9. BaseSpecimen
10. BaseBattery
11. BaseTest
12. BaseResult

A typical key element in the model, BaseResult, would contain attributes as well as sub-elements with their own attributes:

```
- <BaseResult ReportedResultStatus="F" ReportedDateTime="2001-05-10T04:58:10-05:00">
      -<SingleResult ResultClass="R" ResultType="N">
            <TextResult Value="78" />
            <NumericResult Value="78" Precision="" />
            <ResultReferenceRange ReferenceRangeLow="61" ReferenceRangeHigh="84" />
            <ResultUnits Value="g/L" CodeListID="ISO 1000" />
      </SingleResult>
```

In the above complex XML segment, TextResult, NumericResult, ResultReferenceRange, and ResultUnits can be considered "sub-elements", possessing their own attributes and having a different end tag: "/>".

Thus programming the creation of a full-blown LAB model is not trivial. The code has to distinguish between key elements, sub-elements, attributes, and even tolerate a variable number of attributes within an element. Yet, it is quite feasible. Conceptually, the attributes can be considered non-hash data within the internal hash objects, since they normally do not have "children". (However, sub-elements can have descendants, adding to the complexity). In fact, the hash-of-hashes tables usually are mixtures of hash objects and non-hash numerical/character data.

Thus one can conceive of three main approaches for handling such complex tasks:

(a) Using the schema itself to pick element and attribute names for entry into hash objects. This approach would have the additional advantage of validating the XML document at the same time.

(b) Using regular expressions to pick up main elements, sub-elements, and attributes for the program to process into hash objects.

(c) Putting the intact attribute statements as observations, with elements and sub-elements as variables, as input data for hash processing.

The present author had employed the third approach in an earlier publication[7]. The input data for the code had elements as variables and entire attribute texts as data. Where a sub-element lacks a particular attribute, a null observation was presented. For example, to read data for the elements BaseBattery, BaseTest, and the sub-element PerformingLab, the code segment would look like:

```
infile datalines dsd ;
input  BaseBattery  : & $70./
BaseTest : & $70./
PerformingLab : & $70./

datalines;
ID="RC3266" Name="CHEMISTRY",
Status="D" TestType="S",
ID="L1234" Name="Central Lab ABC-Chicago",
```

In the present paper, the above approach has further been improved by first creating separate data sets of attributes, and reading the data sets individually to programmatically compose element statement texts containing attributes. For

instance, BaseBattery can have its own dataset of attributes as shown below:

```
data BaseBatteryAttributes;
infile datalines missover;
input  ID : $12. Name : $12.;
datalines;
RC3266 CHEMISTRY
HM197 HEMATOLOGY
CM792 URINALYSIS
;
```

A line of code can subsequently read the observations using data access functions, and compose a complete element text:

```
dsid=open("BaseBatteryAttributes"); <------open data set
rc1=fetchobs(dsid,1); <--------------------get an observation
name1=varname(dsid,2);<--------------------get the name of attribute
val1=getvarc(dsid,2); <--------------------get the value of attribute
name2=varname(dsid,3);
val2=getvarc(dsid,3);

*****COMPOSE AN ELEMENT TEXT WITH ATTRIBUTES*********************
element=name1||'='||'"'||val1||'"'||' '||name2||name2||'='||'"'||val2||'"';
```

This would give a text value of element as:

ID="RC3266" Name="CHEMISTRY"

It should be pointed out that the observations in the input data could be in random order, since the hash algorithm is able to pick the proper order of elements and attributes. The hash keys and data determine which elements are key ones and the order in which they occur.

PROGRAM:

To create this hierarchy of XML elements:

```
GTP
    BaseBattery
        BaseTest
            PerformingLab
            LabTest
            LOINCTestCode
            BaseResult
                SingleResult
                    TextResult
                    NumericResult
                    ResultReferenceRange
                    ResultUnits
```

The hash definitions are as follows:
(Items in blue are hash objects. Items in red are variables with text data that contain element names and attributes).

```
gt.defineKey("GTP");
gt.defineData("GTP","bat","hibat");

bat.defineKey("BaseBattery");
bat.defineData("BaseBattery","tes","hites");

tes.defineKey("BaseTest");
tes.defineData("BaseTest","PerformingLab", "LabTest", "LOINCTestCode",
"res","hires");

res.defineKey("BaseResult");
res.defineData("BaseResult", "sin","hisin");
```

```
sin.defineKey("SingleResult");
sin.defineData("SingleResult","TextResult","NumericResult",
"ResultReferenceRange","ResultUnits");
```

For BaseTest and SingleResult, sub-elements are also included in the same hash table. The read-out process using hash iterator and macros is the same as described before for the simple LAB model example.

4. **XML OUTPUT:**

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <GTP ModelVersion="01-0-01" CreationDateTime="2012-10-08T16:50:09:20-05:00">
  - <BaseBattery ID="HM197" Name="HEMATOLOGY">
    - <BaseTest Status="D" TestType="S">
        <PerformingLab ID="L1234" Name="Central Lab ABC-Chicago NA" />
        <LabTest ID="HMT13" Name="Platelets" />
        <LOINCTestCode Value="26515-7" CodeListID="LOINC V3.7" />
      - <BaseResult ReportedResultStatus="F" ReportedDateTime="2001-06-23T11:38:05-05:00">
        - <SingleResult ResultClass="C" ResultType="N">
            <TextResult Value="348" />
            <NumericResult Value="348" Precision="" />
            <ResultReferenceRange ResultRangeLow="140" ResultRangeHigh="400" />
            <ResultUnits Value="X10^3/uL" CodeListID="Lab ABC Conventional Units" />
          </SingleResult>
        </BaseResult>
      </BaseTest>
    </BaseBattery>
  - <BaseBattery ID="CM792" Name="URINALYSIS">
    - <BaseTest Status="D" TestType="S">
        <PerformingLab ID="L1234" Name="Central Lab ABC-Chicago NA" />
        <LabTest ID="CMT49" Name="Urine Protein (3+)" />
        <LOINCTestCode Value="2887-8" CodeListID="LOINC V3.7" />
      - <BaseResult ReportedResultStatus="F" ReportedDateTime="2001-07-20T15:22:49-05:00">
        - <SingleResult ResultClass="S" ResultType="T">
            <TextResult Value="Neg" />
            <NumericResult Value="" Precision="" />
            <ResultReferenceRange ResultRangeLow="" ResultRangeHigh="" />
            <ResultUnits Value="" CodeListID="" />
          </SingleResult>
        </BaseResult>
      </BaseTest>
    </BaseBattery>
  - <BaseBattery ID="RC3266" Name="CHEMISTRY">
    - <BaseTest Status="D" TestType="S">
        <PerformingLab ID="L1234" Name="Central Lab ABC-Chicago NA" />
        <LabTest ID="RCT15" Name="SerumSodium" />
        <LOINCTestCode Value="2951-2" CodeListID="LOINC V3.7" />
      - <BaseResult ReportedResultStatus="F" ReportedDateTime="2001-05-10T04:19:32-05:00">
        - <SingleResult ResultClass="R" ResultType="N">
            <TextResult Value="143" />
            <NumericResult Value="143" Precision="" />
            <ResultReferenceRange ResultRangeLow="132" ResultRangeHigh="147" />
            <ResultUnits Value="mmol/L" CodeListID="ISO1000" />
          </SingleResult>
        </BaseResult>
      </BaseTest>
    </BaseBattery>
</GTP>
```

Figure 3: Complex LAB model XML showing attributes and sub-elements generated with the Hash-of-Hashes Technique

FUTURE ENHANCEMENTS FOR COMPLEX XML PROGRAMMING

Instead of generating several data sets of element attributes as input data, an Excel workbook can rather be created where the worksheets represent the various key elements and each worksheet contains the data for the attributes associated with a particular element. Dynamic Data Exchange (DDE) can then be used to read the data into SAS®. Relying on Excel would make the input data more scalable and easily modifiable. An extensive use of macro variables can also make it uneccesary to hard-code element names as done for the complex LAB model in this paper.

CONCLUSION

The present paper demonstrates the capability of the hash-of-hashes technique in processing and transforming a tabular data structure into a hierarchical type. This is possible because of the ability to make hash objects contain other hash objects as data. Since most hierachies can be modeled as containers within containers, a hash object containing other hash objects assumes a hierarchical model by nature. Thus a 2-dimensional SAS® data set can readily be transformed hierarchically by regrouping the data into hashes and sub-hashes, using dynamic instantiation of hash objects.

The technique has the potential of being useful for clinical hierarchical data processing. This has been illustrated in this paper with a simplified version of the LAB model. The ability to process complex XML has also been demonstrated with a segment of the LAB model containing attributes and sub-elements.

The whole process can be summed up as first, using the hash-of-hashes algorithm to distribute data into hierarchies, and second, reading out the created hash hierarchies to generate an XML document.

It is worth mentioning that while the code contains a series of "PUT" statements for assembling the XML document, this is coded as a template, just for one observation of data. Through iteration, the process is repeated many times for as many elements and observations as exist in the data.

The present use of a simplified version of the CDISC LAB model XML structure was just to illustrate the concept and the potential usefulness of the technique. It is also hoped that the demonstrated concept would spur more innovative hash programming techniques by others, leading to more complex applications, like the processing of *define.xml*. It is hoped that in this paper, the capability of the hash-of-hashes technique to create hierarchical data has been revealed.

REFERENCES

1. Hinson J, Shi C. “Transposing tables from long to wide: a novel approach using Hash Objects”. In PharmaSUG 2012: Proceedings of the Pharmaceutical Industry SAS® Users Group; 2012 May 13-16; San Francisco, CA, USA.

2. Dorfman P, Vyverman K. “The SAS® Hash Object in action “. In SGF 2009: Proceedings of the SAS® Global Forum, 2009.

3. Keintz M. “From stocks to flows: using SAS® Hash objects for FIFO, LIFO, and other FO’s “. In NESUG 2011: Proceedings of the Northeast SAS® Users Group, 2011.

4. Loren J, DeVenezia R. “Building provider panels: an application for the Hash of Hashes “. In SGF 2011: Proceedings of the SAS® Global Forum, 2011.

5. CDISC, Laboratory Data Model [document on the Internet]. Avalable from: http://www.cdisc.org/content1058.

6. Source of LAB data extract: http://www.cdisc.org/stuff/contentmgr/files/0/0024dedd3bc7e43bf6c92af66d126f7c/misc/lab1_0_1_basesampledata.xml

7. Hinson J,”Processing of hierarchical data with hash objects Part 1 - Creation of XML documents”. *Pharmaceutical Programming*, Volume 5, Numbers 1-2, December 2012 , pp. 10-28(19), London, UK: Maney Publishing.

ACKNOWLEDGMENTS

The author wishes to express his sincere gratitude to his manager and mentors while at Merck: Margaret Coughlin and Changong Shi, for teaching, guiding, and providing a very intellectually nourishing environment for the author.

The author is also greatly indebted to Dr. Paul Dorfman for providing a wealth of information and tutorials about the hashing technique, and for generating so much enthusiasm and ideas for novel applications.

The author also wishes to acknowledge Richard DeVenezia for discovering the hash-of-hashes technique, and providing many elegant examples, making it possible for the author to develop code and produce this paper.

CONTACT INFORMATION

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

Name:	Joseph Hinson, PhD
City, State ZIP:	Princeton, NJ, 08542
Work Phone:	1-609-540-1309
E-mail:	jwhinson@gmail.com

APPENDIX

Full Code:

```
************************** PLEASE FIRST CHANGE XML OUTPUT LOCATIONS BELOW*****************************************;
%let xmlfilelocation1=C:\Documents and Settings\hinsonj\Desktop\XMLfiles\simplelabfig1.xml; *<--MAKE OWN PATH;
%let xmlfilelocation2=C:\Documents and Settings\hinsonj\Desktop\XMLfiles\complexlabfig1.xml; *<--MAKE OWN PATH;


*=============================================================================================
          PART-1:     CREATION OF SIMPLE XML
                        WITH HASH-OF-HASHES
=============================================================================================;
data simplelabdata;
input  Subject $ 1-4 Visit $ 6-7 BaseTest $ 9-22 SingleResult $ 24-27
BaseBattery $ 29-38 Site $ 40-41 Study $ 43-47;
 datalines;
8222 01 TotalBilirubin 9    CHEMISTRY  11 AB123
8222 01 SerumSodium    143  CHEMISTRY  11 AB123
8222 01 Creatinine     71   CHEMISTRY  11 AB123
8222 02 Platelets      348  HEMATOLOGY 11 AB123
8222 03 Blood          Neg  URINALYSIS 11 AB123
8222 03 UrineProtein   Neg  URINALYSIS 11 AB123
8577 01 TotalProtein   6.6  CHEMISTRY  11 AB123
8577 02 Lymphocytes    3.12 HEMATOLOGY 11 AB123
8577 03 UrineGlucose   Neg  URINALYSIS 11 AB123
9904 01 Albumin        63   CHEMISTRY  36 AB123
9904 01 SerumSodium    129  CHEMISTRY  36 AB123
9904 02 Hematocrit     40   HEMATOLOGY 36 AB123
9904 02 Hemoglobin     183  HEMATOLOGY 36 AB123
9904 03 Blood          Neg  URINALYSIS 36 AB123
;
run;


data _null_;
%let element1=Study;
%let element2=Site;
%let element3=Subject;
```

```
%let element4=Visit;
%let element5=BaseBattery;
%let element6=BaseTest;
%let element7=SingleResult;

        k=0;
*-------------------------------------------------------------------------------------------------------------------------------------------------------------
        SECTION-1B:             C R E A T I O N  O F  M A I N  H A S H  O B J E C T
                                        (for the root element of XML)
-------------------------------------------------------------------------------------------------------------------------------------------------------------;
        if (1=2) then set simplelabdata; *<----(This is just a trick for getting
                                                        variables into PDV for access
                                                        by hash objects. The statement
                                                        is false so is NOT executed !);
        ****FOR &element1 PROCESSING;
        declare hash hash1();
                hash1.defineKey("&element1");
                hash1.defineData("&element1","hash2","hiter2");
                hash1.defineDone();
        declare hiter hiter1("hash1");

        *-------------------------------------------------------------------------------------------------------------------------------------------------
                SECTION-1C:             D Y N A M I C  C R E A T I O N  O F  S U B - H A S H E S
        -------------------------------------------------------------------------------------------------------------------------------------------------;

        ******** Step-1: OBJECT DECLARATION ***************;
        declare hash hash2;
        declare hiter hiter2;
        declare hash hash3;
        declare hiter hiter3;
        declare hash hash4;
        declare hiter hiter4;
        declare hash hash5;
        declare hiter hiter5;
        declare hash hash6;
        declare hiter hiter6;
        declare hash hash7;
        declare hiter hiter7;

        ******** Step-2: HIERARCHICAL CREATION OF HASHES WITHIN HASHES *************;
        do until(done);
                set simplelabdata end=done;

                if hash1.find() ne 0 then
                        do;
                                hash2=_new_ hash();
                                hash2.defineKey("&element2");
                                hash2.defineData("&element2","hash3","hiter3");
                                hash2.defineDone();
                                hiter2=_new_ hiter("hash2");
                                rc1=hash1.replace();
                        end;

                if hash2.find() ne 0 then
                        do;
                                hash3=_new_ hash();
                                hash3.defineKey("&element3");
                                hash3.defineData("&element3", "hash4","hiter4");
                                hash3.defineDone();
                                hiter3=_new_ hiter("hash3");
                                rc2=hash2.replace();
                        end;

                if hash3.find() ne 0 then
                        do;
```

```
                    hash4=_new_ hash();
                    hash4.defineKey("&element4");
                    hash4.defineData("&element4","hash5","hiter5");
                    hash4.defineDone();
                    hiter4=_new_ hiter("hash4");
                    rc3=hash3.replace();
            end;

    if hash4.find() ne 0 then
            do;
                    hash5=_new_ hash();
                    hash5.defineKey("&element5");
                    hash5.defineData("&element5","hash6","hiter6");
                    hash5.defineDone();
                    hiter5=_new_ hiter("hash5");
                    rc4=hash4.replace();
            end;

    if hash5.find() ne 0 then
            do;
                    hash6=_new_ hash();
                    hash6.defineKey("&element6");
                    hash6.defineData("&element6","hash7","hiter7");
                    hash6.defineDone();
                    hiter6=_new_ hiter("hash6");
                    rc5=hash5.replace();
            end;

    if hash6.find() ne 0 then
            do;
                    hash7=_new_ hash();
                    hash7.defineKey("&element7");
                    hash7.defineData("&element7");
                    hash7.defineDone();
                    hiter7=_new_ hiter("hash7");
                    rc6=hash6.replace();
            end;
                    rc7=hash7.replace();

end;*[set simplelabdata...];
call missing(&element1,&element2,&element3,&element4,&element5,&element6,&element7);


*-------------------------------------------------------------------------------------------------------------------------------------------------------
        SECTION-1D:     F O R M A T I O N  O F  S I M P L E  X M L  D O C U M E N T
-------------------------------------------------------------------------------------------------------------------------------------------------------;


length ELEMENT $15 VAL $15;
options nonumber nodate nocenter; title ;

********** Step-1: XML OUTPUT DESTINATION ************************************************************;
filename xmlout "&xmlfilelocation1";
file xmlout;

********** Step-2: MACRO DEFINITIONS FOR CREATING XML SYNTAX *******************************;
%let tag1="<";
%let tag2=">";
%let tag0="/";

%macro tagon (ex, vx);
        xtext=strip(left(&tag1.))||strip(left(&ex.))||strip(left(&tag2.));
        vtext=&vx.;
        put xtext;
        put vtext;
```

```
%mend tagon;
%macro tagoff(ex);
        ztext=strip(left(&tag1.))||strip(left(&tag0.))||strip(left(&ex.))||strip(left(&tag2.));
        put ztext;
%mend tagoff;

********** Step-3: READ-OUT OF HASH-OF-HASHES TO POPULATE XML DOCUMENT **************;
put '<?xml version="1.0" encoding="ISO-8859-1" ?>';

rc=hiter1.first();
do until (hiter1.next() ne 0);
        val=&element1;
        element=vname(&element1);
        %tagon(element, val);
        call missing(element,val);

        do while (hiter2.next() eq 0);
                val=&element2;
                element=vname(&element2);
                %tagon(element, val);
                call missing(element,val);

                do while (hiter3.next() eq 0);
                        val=&element3;
                        element=vname(&element3);
                        %tagon(element, val);
                        call missing(element,val);

                        do while (hiter4.next() eq 0);
                                val=&element4;
                                element=vname(&element4);
                                %tagon(element, val);
                                call missing(element,val);

                                do while (hiter5.next() eq 0);
                                        val=&element5;
                                        element=vname(&element5);
                                        %tagon(element, val);
                                        call missing(element,val);

                                        do while (hiter6.next() eq 0);
                                                val=&element6;
                                                element=vname(&element6);
                                                %tagon(element, val);
                                                call missing(element,val);

                                                do while (hiter7.next() eq 0);
                                                        val=&element7;
                                                element=vname(&element7);
                                                %tagon(element, val);
                                                *call missing(element,val);

                                                element=vname(&element7);
                                                %tagoff(element);
                                                call missing(element);
                                                end;*hiter7;

                                        element=vname(&element6);
                                        %tagoff(element);
                                        call missing(element);
                                        end;*hiter6;

                                element=vname(&element5);
                                %tagoff(element);
                                call missing(element);
```

```
                                        end;*hiter5;

                                    element=vname(&element4);
                                    %tagoff(element);
                                    call missing(element);
                                    end;*hiter4;

                                element=vname(&element3);
                                %tagoff(element);
                                call missing(element);
                                end;*hiter3;

                            element=vname(&element2);
                            %tagoff(element);
                            call missing(element);
                            end;*hiter2;

                    element=vname(&element1);
                    %tagoff(element);
                    call missing(element);
                    end;*hiter1;

stop;
run;

*================================================================================================
        PART-2:     CREATION OF COMPLEX XML
                        WITH HASH-OF-HASHES
================================================================================================;

                    %let elem0=GTP;
                    %let elem1=BaseBattery;
                    %let elem2=BaseTest;
                    %let elem3=PerformingLab;
                    %let elem4=LabTest;
                    %let elem5=LOINCTestCode;
                    %let elem6=BaseResult;
                    %let elem7=SingleResult;
                    %let elem8=TextResult;
                    %let elem9=NumericResult;
                    %let elem10=ResultReferenceRange;
                    %let elem11=ResultUnits;


data GTPAttributes;
        infile datalines dsd;
        input seq ModelVersion $ CreationDateTime : $ 30.;
        datalines;
1,01-0-01,2012-10-08T16:50:09:20-05:00
2,01-0-01,2012-10-08T16:50:09:20-05:00
3,01-0-01,2012-10-08T16:50:09:20-05:00
;
run;

data BaseBatteryAttributes;
        infile datalines missover;
        input seq ID : $12. Name : $12.;
        datalines;
1 RC3266 CHEMISTRY
2 HM197 HEMATOLOGY
3 CM792 URINALYSIS
;
run;
 data BaseTestAttributes;
        infile datalines missover;
```

```
        input seq Status $ TestType $;
        datalines;
1 D S
2 D S
3 D S
;
run;

data PerformingLabAttributes;
        infile datalines dsd;
        input seq ID $ Name : & $ 30.;
        datalines;
1,L1234,Central Lab ABC-Chicago NA
2,L1234,Central Lab ABC-Chicago NA
3,L1234,Central Lab ABC-Chicago NA
;
run;

data LabTestAttributes;
        infile datalines missover;
        input seq ID $ Name : & $ 20.;
        datalines;
1 RCT15 SerumSodium
2 HMT13 Platelets
3 CMT49 Urine Protein (3+)
;
run;

data LOINCTestCodeAttributes;
        infile datalines dsd;
        input seq Value $ CodeListID : $ 12.;
        datalines;
1, 2951-2, LOINC V3.7
2, 26515-7, LOINC V3.7
3, 2887-8, LOINC V3.7
;
run;

data BaseResultAttributes;
        infile datalines dsd;
        input seq ReportedResultStatus $ ReportedDateTime : $ 25.;
        datalines;
1,F,2001-05-10T04:19:32-05:00
2,F,2001-06-23T11:38:05-05:00
3,F,2001-07-20T15:22:49-05:00
;
run;

data SingleResultAttributes;
        infile datalines missover;
        input seq ResultClass $ ResultType $;
        datalines;
1 R N
2 C N
3 S T
;
run;
        data TextResultAttributes;
        infile datalines missover;
        input seq Value $;
        datalines;
1 143
2 348
3 Neg
;
```

```
run;
        data NumericResultAttributes;
        infile datalines missover;
        input seq Value $ Precision $;
        datalines;
1 143
2 348
3
;
run;

data ResultReferenceRangeAttributes;
        infile datalines missover;
        input seq ResultRangeLow $ ResultRangeHigh $;
        datalines;
1 132 147
2 140 400
3
;
run;
data ResultUnitsAttributes;
        infile datalines dsd missover;
        input seq Value $ CodeListID : & $30.;
        datalines;
1, mmol/L, ISO1000
2, X10^3/uL, Lab ABC Conventional Units
3,,
;
run;



data _null_;
        option lrecl=1024;
        length elemtext e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 name1 name2 $30 val1 val2 $70;
        length      seq nx val0 8
                    GTP
                    BaseBattery
                    BaseTest
                    PerformingLab
                    LabTest
                    LOINCTestCode
                    BaseResult
                    SingleResult
                    TextResult
                    NumericResult
                    ResultReferenceRange
                    ResultUnits $100;
        call missing( of _all_);

        declare hash att(ordered:"a");
        att.defineKey("val0");
        att.defineData("val0","&elem0","&elem1","&elem2","&elem3","&elem4","&elem5",
         "&elem6","&elem7","&elem8","&elem9","&elem10","&elem11");
        att.defineDone();
        declare hiter hiatt("att");

        e0="&elem0.";e1="&elem1.";e2="&elem2.";e3="&elem3.";e4="&elem4.";e5="&elem5.";
  e6="&elem6.";e7="&elem7.";e8="&elem8."; e9="&elem9.";e10="&elem10.";e11="&elem11.";
        array hx {12}$100 _temporary_;
        do nx=1 to 3;
        k=0;
                do ex=e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11 while (nx le 3);
                        k=k+1;
                        elemtext=strip(ex)||strip("Attributes");
```

```
                dsid=open(elemtext);
                cols=attrn(dsid,"nvars");
                rc1=fetchobs(dsid,nx);
                val0=getvarn(dsid,1);
                name1=varname(dsid,2);
                val1=getvarc(dsid,2);
                if (cols=3)then do;
                name2=varname(dsid,3);
                val2=getvarc(dsid,3);
        end;
        else do;
                name2="";
                val2="";
        end;
        attribtext1=strip(name1)||"="||'"'||strip(val1)||'"';
        attribtext2=strip(name2)||'='||'"'||strip(val2)||'"';
        if(name1="") and (val1="") then attribtext1="";
        if(name2="") and (val2="") then attribtext2="";

        hx(k)=strip(attribtext1)||' '||strip(attribtext2);

        rca=att.replace(key:val0,data:val0,data:hx(1),data:hx(2),data:hx(3),data:hx(4),data:hx(5),
    data:hx(6),data:hx(7),data:hx(8),data:hx(9),data:hx(10),data:hx(11),data:hx(12));
        rcx=close(dsid);
        end;*(do ex);
        call missing(hx(1),hx(2),hx(3),hx(4),hx(5),hx(6),hx(7),hx(8),hx(9),hx(10),hx(11),hx(12));
        end;*(do nx);
        att.output(dataset:"attribdata");

        k=0;
*-------------------------------------------------------------------------------------------------------------------------
SECTION-2B:     C R E A T I O N  O F  M A I N  H A S H  O B J E C T
                                                        (initial hash container)
-------------------------------------------------------------------------------------------------------------------------;

        declare hash gt(ordered:"a");
                gt.defineKey("GTP");
                gt.defineData("GTP","bat","hibat");
                gt.defineDone();
        declare hiter higt("gt");

*-------------------------------------------------------------------------------------------------------------------------
SECTION-2C:     D Y N A M I C  C R E A T I O N  O F  S U B - H A S H E S
-------------------------------------------------------------------------------------------------------------------------;

******** Step-1: OBJECT DECLARATION ***************************************************************************;

        declare hash bat;
        declare hiter hibat;
        declare hash tes;
        declare hiter hites;
        declare hash res;
        declare hiter hires;
        declare hash sin;
        declare hiter hisin;

******** Step-2: HIERARCHICAL CREATION OF HASHES WITHIN HASHES ****************************;

do while (hiatt.next() eq 0);

rcgt=gt.find();
if rcgt ne 0 then
        do;
                bat=_new_ hash();
                bat.defineKey("BaseBattery");
```

```
                    bat.defineData("BaseBattery","tes","hites");
                    bat.defineDone();
                    hibat=_new_ hiter("bat");
                    rcgt=gt.replace();
            end;

if bat.find() ne 0 then
            do;
                    tes=_new_ hash();
                    tes.defineKey("BaseTest");
                    tes.defineData("BaseTest","PerformingLab", "LabTest", "LOINCTestCode", "res","hires");
                    tes.defineDone();
                    hites=_new_ hiter("tes");
                    rcbat=bat.replace();
            end;

if tes.find() ne 0 then
            do;
                    res=_new_ hash();
                    res.defineKey("BaseResult");
                    res.defineData("BaseResult", "sin","hisin");
                    res.defineDone();
                    hires=_new_ hiter("res");
                    rctes=tes.replace();
            end;

if res.find() ne 0 then
            do;
                    sin=_new_ hash();
                    sin.defineKey("SingleResult");
                    sin.defineData("SingleResult","TextResult","NumericResult","ResultReferenceRange","ResultUnits");
                    sin.defineDone();
                    hisin=_new_ hiter("sin");
                    rcres=res.replace();
            end;
                    rsin=sin.replace();

end;*[set labdata...];
call missing(of _ALL_);

*-------------------------------------------------------------------------------------------------------------------------------------------------------
SECTION-2D:         F O R M A T I O N   O F   C O M P L E X   X M L   D O C U M E N T
-------------------------------------------------------------------------------------------------------------------------------------------------------;

length ELEMENT ELEMENT1 ELEMENT2 ELEMENT3 ELEMENT4 $20 VAL VAL1 VAL2 VAL3 VAL4 $70;
options nonumber nodate nocenter; title ;

********** Step-1: XML OUTPUT DESTINATION ***************************************************************************;
filename xmlout "&xmlfilelocation2";
file xmlout;

********** Step-2: MACRO DEFINITIONS FOR CREATING XML SYNTAX ************************************************;
%let tag1="<";
%let tag2=">";
%let tag0="/";

%macro tagon (ex, vx);
xtext=%sysfunc(strip(%sysfunc(left(&tag1.)))) | | %sysfunc(strip(%sysfunc(left(&ex.)))) | |
" "| | %sysfunc(strip(%sysfunc(left(&vx)))) | | %sysfunc(strip(%sysfunc(left(&tag2.))));
put xtext;
%mend tagon;

%macro atton (ex, vx);
xtext=%sysfunc(strip(%sysfunc(left(&tag1.)))) | | %sysfunc(strip(%sysfunc(left(&ex.)))) | |
" "| | %sysfunc(strip(%sysfunc(left(&vx))));
```

```
wtext="/>";
put xtext;
put wtext;
%mend atton;

%macro tagoff(ex);
ztext=%sysfunc(strip(%sysfunc(left(&tag1.)))||%sysfunc(strip(%sysfunc(left(&tag0.)))||
strip(left(&ex.))||%sysfunc(strip(%sysfunc(left(&tag2.))));
put ztext;
%mend tagoff;

********** Step-3: READ-OUT OF HASH-OF-HASHES TO POPULATE XML DOCUMENT ****************************;
put '<?xml version="1.0" encoding="ISO-8859-1" ?>';

rc=higt.first();
do until (higt.next() ne 0);
val=GTP;
element=vname(GTP);
%tagon(element, val);
call missing(element, val);

do while (hibat.next() eq 0);
        val=BaseBattery;
        element=vname(BaseBattery);
        %tagon(element, val);
        call missing(element,val);

        do while (hites.next() eq 0);
                val1=BaseTest;
                val2=PerformingLab;
                val3=LabTest;
                val4=LOINCTestCode;
                element1=vname(BaseTest);
                %tagon(element1, val1);
                element2=vname(PerformingLab);
                %atton(element2, val2);
                element3=vname(LabTest);
                %atton(element3, val3);
                element4=vname(LOINCTestCode);
                %atton(element4, val4);
                call missing(of element:,of val:);

                do while (hires.next() eq 0);
                        val=BaseResult;
                        element=vname(BaseResult);
                        %tagon(element, val);
                        call missing(element,val);

                        do while (hisin.next() eq 0);
                                val1=TextResult;
                                val2=NumericResult;
                                val3=ResultReferenceRange;
                                val4=ResultUnits;
                                val=SingleResult;
                                element=vname(SingleResult);
                                %tagon(element, val);
                                call missing(element,val);

                                element1=vname(TextResult);
                                %atton(element1, val1);
                                element2=vname(NumericResult);
                                %atton(element2, val2);
                                element3=vname(ResultReferenceRange);
                                %atton(element3, val3);
                                element4=vname(ResultUnits);
```

```
                                        %atton(element4, val4);
                                        call missing(of element:,of val:);

                                element=vname(SingleResult);
                                %tagoff(element);
                                call missing(element);
                                end;*hisin;

                        element=vname(BaseResult);
                        %tagoff(element);
                        call missing(element);
                        end;*hires;

                element1=vname(BaseTest);
                %tagoff(element1);
                call missing(element1);
                end;*hites;

        element=vname(BaseBattery);
        %tagoff(element);
        call missing(element);
        end;*hibat;

        element=vname(GTP);
        %tagoff(element);
        call missing(element);
        end;*higt;

stop;
run;
```