Processing of hierarchical data with hash objects

Part 1 – Creation of XML documents

Joseph Hinson

Princeton, NJ, USA

The introduction of hash objects with SAS® version 9 brought about new opportunities for novel programming techniques. The recent discovery that hash tables can contain even other hash objects, also known as the 'hash of hashes', opens the door to their application to hierarchical data structures. This is because hierarchies, like the XML structure, can be considered 'containers within containers', and with the hash-of-hashes technique, tables can contain tables, thereby becoming compatible with hierarchical formats. This has the potential of being useful for processing clinical data in XML format. The advent of Clinical Data Interchange Standards Consortium (CDISC) and its increasing reliance on XML technology for handling clinical trial data implies that clinical SAS® programmers now have to get used to hierarchical data structures. Clinical trial data set submissions now have to be accompanied by metadata such as define.xml. The traditional SAS® data sets are tabular and relational, whereas the XML structure is hierarchical. SAS® Institute has embraced this new development with a set of XML tools like the XML libname engine, XML mapper, and the CDISC Procedure. One untapped resource is the hash object, which has otherwise proven very useful for fast table look-ups and sortless merges. In this first series of articles, the conversion of a SAS® data set to an XML structure using the hash-of-hashes technique is demonstrated. The illustration is based on a simplified CDISC laboratory data model.

Keywords: Hash objects, Hash-of-hashes, XML, Laboratory data model, CDISC

Introduction

Hierarchical data structures are ubiquitous, and with the current proliferation of XML-based documents, programmers might need to come up with novel tools for processing such structures. The Clinical Data Interchange Standards Consortium (CDISC) has been systematically developing several XML-based models for the electronic acquisition, submission, and archival of clinical trial data. Models such as the Operational Data Model, the Study Design Model, 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 increasingly adopting CDISC standards as the preferred way of submitting drug application data and information. One dramatic move by the FDA is the proposed replacement of the SAS® version 5 transport file format with an XML format. Also required is the presentation of data set metadata as an XML entity 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 SAS® programming thought process, which is based on the data set, a relational and tabular structure of observations and columns (variables). Other data types exist that do not follow such regular structure of rows and columns as seen especially in the SAS® data set and 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 hierarchical, 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

Correspondence to: J Hinson, 100E Leigh Avenue, Princeton, NJ 08542, USA. Email: joseph.hinson@mail.mcgill.ca

Published by Maney
DOI 10.1179/1757092112Z.0000000003

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 such as XML. As will be shown in this paper, this is possible because of the unique ability of hash objects to contain other hash objects.

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.

Hash objects are RAM memory-resident tables and are called 'objects' because they possess their own methods and attributes. Each record in a hash table is made up of distinct lookup key and its associated data. Hash objects (also called 'associative arrays') permit the quick retrieval of data based on the lookup key. Each lookup key is passed through a 'hash' function which maps the key to a record in the hash table.

The hash iterator is a sort of a 'hash table navigator', which allows data items in a hash table to be retrieved without reliance on lookup keys. It achieves this by accessing data in a step-by-step sequential manner, and was used in this paper to consolidate all the various hash objects into a final single XML listing.

As earlier mentioned, hash objects have a variety of methods associated with a particular object. In this paper, the hash methods object.find() and object.replace() were widely used to look for data and to place new data in a hash table. Also, all the XML listing generated by the program in this paper used the object.add() and the object.output() methods. The hash iterator object methods, object.first() and object.next(), were used for the hierarchical accessing of data for XML creation.

The hash-of-hashes technique

Ever since the introduction of hash objects programming, the technique has been popular in 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 Richard DeVenezia made the discovery that 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 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' (Fig. 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.

The phrase 'hash of hashes' refers to the multilevel nesting of hash objects, based on unique hash keys. For instance, to make a hash object, ***geo***, containing another hash object ***state***, 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, as shown in Fig. 1B below, each different country would have its own table of states kept in the `state` object:

The hash-of-hashes thus can segregate data into distinct hierarchies according to the hash key.

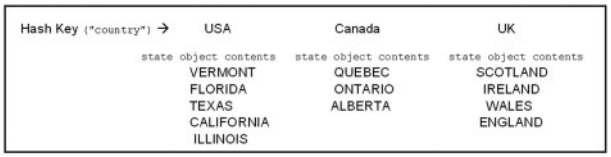

Figure 1 Hash-of-hashes as containers within containers

In the above example, the individual states could also be hash objects containing distinct set of cities if they have 'state' as the key, and a 'city' hash as data. Such that key=ENGLAND would have a hash object containing *London*, *Liverpool*, and *Bristol*, whereas key=SCOTLAND would have an object containing *Edinburgh*, *Glasgow*, and *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, ultimately creating a hierarchy of nested groups.

This is also the basis of data set splitting,[2] where such hash groupings can be output as new distinct data sets, using the *object.output (dataset:'name')* method.

Creating a hash-of-hashes hierarchy

The actual process of creating a hash-of-hashes begins with reading an observation from an input data set, then assigning key and data values to an object based on the desired hierarchical structure. For instance, a data set containing continents, cities, populations, countries, and states, would be read and hash objects assigned such that a continent hash would contain a country hash, which would also contain a state hash, filled with a city hash carrying population numbers. When the observation is read, the hash picks up the proper value for the KEY, then checks whether a hash object already exists for that particular KEY. If not, it creates a new hash object for that key using the *_new_* operator. This way, a hash object gets created for a specific KEY value, with the *_new_* operator allowing the same hash name to be reused.

Creating an object

Normally, one would create a hash object by first declaring and then instantiating the object.

This can be done with a single statement (METHOD 1):

```
declare hash name();
```

or with two separate statements (METHOD 2):

```
declare hash name;
name = _new_ hash;
```

To create a hash object2 inside another hash object1, First, object1 is created using METHOD 1:

```
declare hash object1();
object1.defineKey('variable1');
object1.defineData('variable1','object2');
object1.defineDone();
```

Then, METHOD 2 is used to declare object2 (*no* instantiation). A test is then made to see if for that particular KEY ('variable1'), object1 already contains object2. If not, object2 is instantiated. Then object1 is updated (with 'object1.replace()') to reflect that new content:

```
declare hash object2;
if object1.find() ne 0 then
do;
      object2 = _new_ hash;
      object2.defineKey('variable2');
      object2.defineData('variable2',
      'object3');
      object2.defineDone();

      object1.replace();
end;
```

This process is continued for object3, object4, object5, etc., till the whole hierarchy of objects within objects have been created.

Reading from the hash of hashes to create a hierarchical listing

Once the hash-of-hashes has been created, the *hash iterator object* (SAS® abbreviation: 'hiter') is used to read information from the nested hash objects without the use of a KEY. Information is read row-by-row, starting from the first row and using the method 'iterator.next()' (other available methods are 'iterator.first()', 'iterator.prev()', and 'iterator.-last()'). By using iterator objects, contents of all the nested tables become available for reading in the proper sequence according to their positions in the hierarchy.

The creation of iterator objects

Hash iterator objects are also created in the same manner as hash objects, except that their arguments are the specific hash objects on which they would operate:

```
declare hiter name('object1');
```

or in two separate steps:

```
declare hiter name;'
name=_new_ hiter('object1');
```

The iterator objects are inserted into hash objects just like their hash counterparts:

```
declare hash object1();
        object1.defineKey('variabl-
        e1');
        object1.defineData('variabl-
        e1','object2', 'hiter2');
        object1.defineDone();

declare hash object2;
if object1.find() ne 0 then
do;
        object2 = _new_ hash;
        object2.defineKey('variabl-
        e2');
        object2.defineData('variabl-
        e2', 'object3', 'hiter3');
        object2.defineDone();
        hiter2 = _new_ hiter;

        object1.replace();
end;
```

The hash objects nested within other objects are read object-by-object according to their positions in the hierarchy, systematically creating an XML document (a new hash object 'xml' is first created for building the XML document).

```
rc=hiter1.first();
    do until (hiter1.next() ne 0); ←---
    return code =0 if next() is possible
    element1='<Variable1>'; ←----------
    for the start tag
    rc=xml
    .add(); ←--------------------add the
    retrieved info to new object 'xml'
    element1=Variable1; ←---------------
    for the actual data
    rc=xml.add();

       do while (hiter2.next() eq 0);
       element2='<Variable2>';
       rc=xml.add();
       element2=Variable2;
       rc=xml.add();

              do while (hiter3.next()
              eq 0);
              element3='<Variabl
              e3>';
              rc=xml.add();
              element3=Variable3;
              rc=xml.add();

                     etc.

                     call missin-
                     g(of hash:);
             end;*hiterN;
                    -    -
                    -    -
                    -    -
      end;*hiter2;
end;*hiter1;
```

(NOTE: ***labsumm*** *must be created beforehand; alternatively, an explicit OUTPUT statement can be used to create a table directly, instead of the hash object* ***labsumm****).*

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,

<**subjectid**> **100098** <**/subjectid**>

This element can also be written longitudinally as:

<**subjectid**>

100098
</subjectid>

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>
<country>
    Canada
      <state>
         Ontario
            <city>
               Toronto
                  <population>
                     4 million
                  </population>
              </city>
         </state>
     </country>
</continent>
```

demonstrating the hierarchical presentation of elements: *the population of cities within states within countries within a continent.* So in a way, one could consider the XML structure as 'containers within containers', with the base element CONTINENT 'containing' the COUNTRY element, also 'containing' the STATE element, and so on. In a similar fashion, one could conceive of a continent hash object containing a country object filled with a state object which itself contains a city object carrying population data per city. This is the basis of the compatibility between the hash-of-hashes and hierarchical data structures such as XML.

Programming Hashes-of-hashes to Create an XML Structure from a SAS Data Set

The hash of hashes algorithm

This simply is a two-step strategy:

1. isolate distinct groups from the data and place them in hash objects in a hierarchical manner;
2. use the hash iterator to traverse each hash object, starting from the outer 'container' all the way to the innermost container, issuing PUT commands to lay out XML elements.

Step 1: creating hashes within hashes from input data set

General rule

If we read in *N* variables, var1, var2, …, varN, where we want to create a hierarchy of hash1 containing hash2 containing hash3 ……. containing varN, the procedure is:

```
declare hash hash1 ();
        hash1.defineKey('var1');
        hash1.defineData('var1','hash2','hite-
        r2');
        hash1.defineDone();
declare hiter hiter1('hash1');
```

Please note: All the hash and hiter objects (hash2 to hashN, hiter2 to hiterN) must be declared at this point before proceeding.

```
rc1=hash1.find(); ←-----rc1 is a return code. '0' means the
search was successful
if rc1 ne 0 then
       do;
              hash2 = _new_ hash();
              hash2.defineKey('var2');
              hash2.defineData('var2','hash3',
              'hiter3');
              hash2.defineDone();
              hiter2 = _new_ hiter('hash2');
              rc=hash1.replace();
       end;
       if hash2.find() ne 0 then
              do;
                       hash3 = _new_ hash();
                       hash3.defineKey('var3');
                       hash3.defineData('var3',
                       'hash4','hiter4');
                       hash3.defineDone();
                       hiter3 = _new_ hiter
                       ('hash3');
                       hash2.replace();
              end;
                              -   -
                              -   -
                              -   -
                              -   -
              if hashN-1.find() ne 0 then
                       do;
                              hashN = _new_
                              hash();
                              hashN.defineKey
                              ('varN');
                              hashN.defineData
                              ('varN',
                              'varN+1');
                              hashN.defineDone();
                              hiterN = _new_
                              hiter('hashN');
                              hashN-1.replace();
                       end;
                              hashN.replace();
                              ←----only for the very last
                              hash;
```

Step 2: using the hash iterator ('hiter') to retrieve data from hashes within hashes to create a hierarchical listing (Fig. 4)

General rule

```
rc=hiter1.first(); ←----------only
for the very first hash
```

Figure 2 A segment of the XML schema for the CDISC LAB model (with several elements and attributes removed for clarity)

```
do until (hiter1.next() ne 0);
       hash1=var1;
       rc1=labsumm.add();
       row=row+1;
       call missing(of hash:);

       do while (hiter2.next() eq 0);
              hash2=var2;
              rc=labsumm.add();
              row=row+1;
              call missing(of hash:);
                           _    _
                           _    _
                           _    _
                     do while
                     (hiterN.next()
                     eq 0);
                     hashN=varN;
                     rc=labsum-
                     m.add();
                     row=row+1;
```

Application to XML-based clinical data — I

A simplified xml structure of the CDISC LAB

The XML-based LAB model was developed by CDISC for the transfer of clinical lab data from vendors to sponsors. Like all XML documents, a schema document exists for the lab data model, as shown in Fig. 2 (several elements and attributes removed for clarity). A short segment of the CDISC LAB model BaseSampleData.xml is shown in Fig. 3.

The *root element* for the LAB XML document is named 'Good Transmission Practice'.

Good Transmission Practice data provides information about either the transmission as a whole or a particular record within it. In this section, focus would be on just a few LAB model elements: ***Study***, ***Site***, ***Subject***, ***Visit***, ***BaseBattery***, ***BaseTest***, and ***SingleResult***.

Figure 3 A segment of a CDISC LAB model BaseSampleData.xml

Transforming a laboratory data set into a LAB model xml structure

1. a conventional data set, LABDATA, is first created from raw data with seven items as shown in Fig. 4;
2. the hash-of-hashes algorithm is then applied, leading first to the table structure shown in Fig. 5.

Step 3: generate xml tags and elements

Then, as the contents of the hashes within hashes were retrieved with the hiter object, data_null_/put method was used to create a text file:

Output specification:

```
filename xmlout 'C:\Documents and
Settings\hinsonj\Desktop\XMLfiles\
lab.xml';
file xmlout;
```

Macros for inserting tags:

```
%let tag1='<';
%let tag2='>';
%let tag0='/';

%macro tagon (ex, vx);
xtext=strip(left(&tag1.))||stri-
p(left(&ex.))||strip(left(&tag2.));
```

```
vtext=&vx.;
put xtext;
put vtext;
%macro tagon;

%macro tagoff(ex);
ztext=strip(left(&tag1.))||stri-
p(left(&tag0.))||strip(lef-
t(&ex.))||strip(left(&tag2.));
put ztext;
%mend tagoff;
```

XML header (abbreviated):

```
put '<?xml version=' 1.0'
encoding=' ISO-8859-1' ?>' ;
```

General algorithm:

```
****READ HASH OF HASHES and SEND TO XML DOCUMENT BEING CREATED **************;
 rc=hiter1.first();
 do until (hiter1.next() ne 0);
        value=Variable1;
        element=vname(Variable1);
        call missing(element,value);

        do while (hiter2.next() eq 0);
               value=Variable2;
               element=vname(Variable2);
               put catx(element,close);
               put strip(value);
               call missing(element,value);
                              - - -
                              - - -
                              - - -
                                                  do while (hiterN.next() eq 0);
                                                       value=VariableN;
                                                  element=vname(VariableN);
                                                  put catx(element,close);
                                                  put strip(value);

                                                  *call missing(element,value);
                                        element=vname(VariableN);
                                        put catx(open,end,element,close);
                                        call missing(element);
                                        end;*hiterN;

                                element=vname(VariableN-1);
                                put catx(open,end,element,close);
                                call missing(element);
                                end;*hiterN-1;

                                - - -
                                - - -
                                - - -
 element=vname(Variable1);
 put catx(open,end,element,close);
 call missing(element);
 end;*hiter1;
```

XML output

Figure 6A–D are segments of the LAB model XML output generated by the hash-of-hashes algorithm:

Figure 4 A typical laboratory raw data

Application to XML-based Clinical Data — II

A complex XML structure of the CDISC LAB model (with both elements and attributes)

Clinical data in XML format are far more intricate than the examples shown above 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 model has 12 key elements:

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'>
```

Figure 5 A segment of the output data set LabHashOfHashes showing nested table items

```
-<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'. They have their own attributes and have a different end tag: '/>'.

Thus programming a full-blown LAB model is not trivial. Yet, it is quite feasible to program a 12-level nested hash-of-hashes to process such a complex XML data. The challenge would be to incorporate the attributes associated with the elements and sub-elements. An effective code has to discriminate between different types of XML elements, identifying attributes, and constructing these various forms using the appropriate tags.

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).

Figure 6 (A) Study AB123, Site 11, Subject 8222, Visit 1; (B) Subject 8577, Visit 1; (C) Site 36, Subject 9904, Visit 1; (D) Site 36, Subject 9904, Visit 3 (Final)

Figure 7 Continued

Figure 7 Programming steps for generating an XML lab document using the hash-of-hashes technique

In fact, the hash-of-hashes tables are usually a mixture 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 is the preferred approach as it would also afford the opportunity to validate 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 first approach is already in development. By using the lab schema itself and putting the schema into a hash look-up table, the main input data can easily be differentiated into elements and attributes using regular expressions to parse and compare with information in the schema hash object. In the current paper, the third approach has been used below just to demonstrate the concepts involved in processing elements with attributes. The input data were created by extracting information from a Lab XML document available on the CDISC website.[5] This was done on purpose to facilitate validation of the code. 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

Figure 7 Continued

and attributes. The hash keys and data determine which elements are key ones and the order in which they occur. For the sake of simplicity, only the BaseBattery segment of the LAB model is being presented here.

As shown below, the input data (part of the program) has elements as variables and entire attributes as data. Where a sub-element lacks a particular attribute, a null observation is presented (in future, such input data can be laid out as worksheets in an Excel workbook, with each worksheet representing a main element). The output is shown in Fig. 8

Program

```
data labtex;
option lrecl=1024;
infile datalines dsd ;
```

Figure 7 Continued

```
length  BaseBattery
        BaseTest
        PerformingLab
        LabTest
        LOINCTestCode
```

Figure 8 BaseBattery segment of LAB model XML showing attributes and sub-elements generated with the hash-of-hashes technique

```
          BaseResult
          SingleResult
          TextResult
          NumericResult
          ResultReferenceRange
          ResultUnits $70;

input     BaseBattery : & $70./
          BaseTest : & $70./
          PerformingLab : & $70./
          LabTest : & $70./
          LOINCTestCode : & $70./
          BaseResult : & $70./
          SingleResult : & $70./
          TextResult : & $70./
          NumericResult : & $70./
        ResultReferenceRange : & $70./
        ResultUnits : & $70.
        ;
datalines;
ID=' RC3266' Name=' CHEMISTRY',
Status=' D' TestType=' S',
ID=' L1234' Name=' Central Lab ABC-Chicago',
ID=' RCT1' Name=' Total Bilirubin',
Value=' 14631-6' CodelistID=' LOINC V3.7',
ReportedResultStatus=' F' ReportedDataTime=' 2001-05-10T04:58:10-05:00',
ResultClass=' R' ResultType=' N',
Value=' 9',
Value=' 9' Precision='',
ReferenceRangeLow=' 3' ReferenceRangeHigh=' 21',
```

```
Value=' umol/L' CodeListID=' ISO1000'
ID=' RC3266' Name=' CHEMISTRY',
Status=' D' TestType=' S',
ID=' L1234' Name=' Central Lab ABC-Chicago',
ID=' RCT1' Name=' Total Bilirubin',
Value=' 14631-6' CodelistID=' LOINC
V3.7',
ReportedResultStatus=' F' ReportedDataTime=' 2001-05-10T04:58:10-05:00',
ResultClass=' C' ResultType=' N',
,
Value=' 0.5' Precision='',
,
Value=' mg/dL' CodeListID=' Lab ABC'
ID=' RC3266' Name=' CHEMISTRY', …….
Status=' D' TestType=' S',
ID=' L1234' Name=' Central Lab ABC-Chicago',
ID=' RCT1' Name=' Total Bilirubin',
Value=' 14631-6' CodelistID=' LOINC V3.7',
ReportedResultStatus=' F' ReportedDataTime=' 2001-05-10T04:58:10-05:00',
ResultClass=' S' ResultType=' N',
,
Value=' 9' Precision=' ',
,
Value=' umol/L' CodeListID=' ISO1000'
;
run;

data _null_;
      k=0;
*-------------------------------------------------------------------------------------------------------------------------------
[1]CREATION OF FIXED HASH OBJECT(parent hash container)
-------------------------------------------------------------------------------------------------------------------------------;
if (1=2) then set labtex; *<----(This is just a trick for getting
                                variables into PDV for access
                                by hash objects. The statement
                                is false so is NOT executed !);
declare hash bat();
       bat.defineKey('BaseBattery');
       bat.defineData('BaseBatter-
       y','tes','hites');
       bat.defineDone();
declare hiter hibat('bat');
*-------------------------------------------------------------------------------------------------------------------------------
[2]DECLARATION OF HASH OBJECTS FOR DYNAMIC INSTANTIATION
--------------------------------------------------------------------------------;
declare hash tes;
declare hiter hites;
declare hash res;
declare hiter hires;
declare hash sin;
declare hiter hisin;
*-------------------------------------------------------------------------------------------------------------------------------
[3]ISOLATION OF DATA INTO HASHH IERARCHIES(Hash-Of-Hashes)
-------------------------------------------------------------------------------------------------------------------------------
                                        ;
do until(done);
       set labtex end=done;
       rcbat=bat.find();
       if rcbat 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');
                            tes.replace();
                    end;

               if res.find() ne 0 then
                      do;
                            sin=_new_ hash();
                            sin.defineKey('SingleResult');
                            sin.defineData('SingleResult','TextResult','NumericRes-
                            ult','ResultReferenceRange','ResultUnits');
                            sin.defineDone();
                            hisin=_new_ hiter('sin');
                            res.replace();
                      end;
                            rsin=sin.replace();
 end;*[ set labdata…];
 call missing(of _ALL_);
*-------------------------------------------------------------------------------------------------------------------------
(4) X M L O U T P U T
-------------------------------------------------------------------------------------;
 length ELEMENT ELEMENT1 ELEMENT2 ELEMENT3 ELEMENT4 $15 VAL VAL1 VAL2 VAL3 VAL4 $70;
 options nonumber nodate nocenter; title ;

 filename xmlout 'C:\Documents and Settings\hinsonj\Desktop\XMLfiles\lab.xml';
 file xmlout;
 %let tag1='<';
 %let tag2='>';
 %let tag0='/';
     **************MACRO DEFINITIONS FOR CREATING XML SYNTAX**************;
 %macro tagon (ex, vx);
 xtext=%sysfunc(strip(%sysfunc(left(&tag1.))))||%sysfunc(strip(%sysfunc(lef-
 t(&ex.))))||' '||%sysfunc(strip(%sysfunc(left(&vx))))||%sysfunc(strip(%sysfun-
 c(left(&tag2.))));
 put xtext;
   %mend tagon;

   %macro atton (ex, vx);
   xtext=%sysfunc(strip(%sysfunc(left(&tag1.))))||%sysfunc(strip(%sysfunc(lef-
   t(&ex.))))||' '||%sysfunc(strip(%sysfunc(left(&vx))));
   wtext='/>';
   put xtext;
   put wtext;
   %mend atton;

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

   put '<?xml version='1.0' encoding='ISO-8859-1' ?>';

   ****READ HASH OF HASHES and SEND TEXT FILE **************;

   rc=hibat.first();
   do until (hibat.next() ne 0);
        val=BaseBattery;
```

```
        element=vname(BaseBattery);
        %tagon(element, val);
        call missing(element,va-
l);(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;

stop;
run;
```

Output

Future enhancements

As earlier mentioned, regular expressions can be used to parse attribute data, making it unnecessary to present the entire attribute statement as an observation in input data. Similarly, the different data attributes can be consolidated into their own data sets: for instance, an ID data set, a NAME data sets, etc. Logic can then be developed, using the schema as look-up table, to construct an attribute text by reading in information from the various data sets.

Conclusions

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 two-dimensional SAS® data set can readily be converted to a hierarchical data model through the hash-of-hashes processing.

The technique has the potential of being useful for clinical hierarchical data processing. This has been fully demonstrated 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 technique can further be extended with extra coding to handle multiple input data sets and multiple output tables within the same hierarchy. A planned sequel to this paper will demonstrate how to use this hash-of-hashes technique to process complete XML documents containing all elements and attributes. 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 as the processing of define.xml. Hopefully, in this first paper of the series, the capability of the hash-of-hashes technique to process nested information has been exposed.

Acknowledgements

The author is 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.

Appendix

The full code (Fig. 7)

The LABDATA data set should first be generated, as shown in Fig. 4.

```
data _null_;
    k=0;
    *-------------------------------------------------------------
    [1] C R E A T I O N   O F   F I X E D   H A S H   O B J E C T S
        (initial hash containers)
    -------------------------------------------------------------;
    if (1=2) then set labdata; *<----(This is just a trick for getting
                        variables into PDV for access
                        by hash objects. The statement
                        is false so is NOT executed !);
    ****FOR STUDY PROCESSING;
    declare hash std();
        std.defineKey("STUDY");
        std.defineData("STUDY","sit","hisit");
        std.defineDone();
    declare hiter histd("std");

    *-------------------------------------------------------------
    [2] C R E A T I O N   O F   R E U S A B L E   H A S H   O B J E C T S
        (for making hashes within hashes)
    -------------------------------------------------------------;
    declare hash sit;
    declare hiter hisit;
    declare hash sub;
    declare hiter hisub;
    declare hash vis;
    declare hiter hivis;
    declare hash bat;
    declare hiter hibat;
    declare hash tes;
    declare hiter hites;
    declare hash res;
    declare hiter hires;
```

(a)

```
*-------------------------------------------------------------
[3] I S O L A T I O N   O F   G R O U P S
    B Y   H A S H - W I T H I N - H A S H E S
-------------------------------------------------------------;
do until(done);
    set labdata end=done;

    rcstd=std.find();
    if rcstd ne 0 then
        do;
            sit=_new_ hash();
            sit.defineKey("SITE");
            sit.defineData("SITE","sub","hisub");
            sit.defineDone();
            hisit=_new_ hiter("sit");
            rcstd=std.replace();
        end;

        if sit.find() ne 0 then
            do;
                sub=_new_ hash();
                sub.defineKey("SUBJECT");
                sub.defineData("SUBJECT", "vis","hivis");
                sub.defineDone();
                hisub=_new_ hiter("sub");
                sit.replace();
            end;

            if sub.find() ne 0 then
                do;
                    vis=_new_ hash();
                    vis.defineKey("VISIT");
                    vis.defineData("VISIT","bat","hibat");
                    vis.defineDone();
                    hivis=_new_ hiter("vis");
                    sub.replace();
(b)             end;


                if vis.find() ne 0 then
                    do;
                        bat=_new_ hash();
                        bat.defineKey("BASEBATTERY");
                    bat.defineData("BASEBATTERY","tes","hites");
                        bat.defineDone();
                        hibat=_new_ hiter("bat");
                        rcvis=vis.replace();
                    end;

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

                        if tes.find() ne 0 then
                            do;
                                res=_new_ hash();
                            res.defineKey("SINGLERESULT");
                            res.defineData("SINGLERESULT");
                                    res.defineDone();
                            hires=_new_ hiter("res");
                                tes.replace();
                            end;
                            rcres=res.replace();

    end;*[set labdata...];
(c) call missing(STUDY,SITE,SUBJECT,VISIT,BASEBATTERY,BASETEST,SINGLERESULT);
```

```
*-------------------------------------------------------------
   [4] C O N S O L I D A T I O N   O F   H A S H   T A B L E S
I N T O   O N E   H I E R A R C H I C A L     L I S T I N G (LabHash0fHashes)
-------------------------------------------------------------;

length row 3 hash1 $8 hash2 $8 hash3 $10 hash4 $8 hash5 $15 hash6 $15 hash7 $15;

declare hash labsumm(ordered:"a");
labsumm.defineKey("row","hash1","hash2","hash3","hash4","hash5","hash6","hash7");
labsumm.defineData("row","hash1","hash2","hash3","hash4","hash5","hash6","hash7");
labsumm.defineDone();
declare hiter hisumm("labsumm");
call missing(of hash:);

    row=0;
    ****READ HASH OF HASHES INTO A DATASET(LabHash0fHashes)****;
    rc=histd.first();
    do until (histd.next() ne 0);
        hash1=STUDY;
        rc1=labsumm.add();
        row=row+1;
        call missing(of hash:);

        do while (hisit.next() eq 0);
            hash2=SITE;
            rc2=labsumm.add();
            row=row+1;
            call missing(of hash:);

            do while (hisub.next() eq 0);
                hash3=SUBJECT;
                rc3=labsumm.add();
                row=row+1;
                call missing(of hash:);
                                                        (d)

                  do while (hivis.next() eq 0);
                      hash4=VISIT;
                      rc4=labsumm.add();
                      row=row+1;
                      call missing(of hash:);

                      do while (hibat.next() eq 0);
                          hash5=BASEBATTERY;
                          rc5=labsumm.add();
                          row=row+1;
                          call missing(of hash:);

                          do while (hites.next() eq 0);
                              hash6=BASETEST;
                              rc6=labsumm.add();
                              row=row+1;
                              call missing(of hash:);

                              do while (hires.next() eq 0);
                                  hash7=SINGLERESULT;
                                  rc7=labsumm.add();
                                  row=row+1;
                                  call missing(of hash:);
                              end;*hires;
                          end;*hites;
                      end;*hibat;
                  end;*hivis;
              end;*hisub;
          end;*hisit;
end;*histd;

rcs=labsumm.output(dataset:"LabHash0fHashes");      (e)
```

```
*-------------------------------------------------------------------------------
(5)                    XML   OUTPUT   USING
                  DATA_NULL_AND PUT  STATEMENTS
-------------------------------------------------------------------------------;

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

filename xmlout "C:\Documents and Settings\hinsonj\Desktop\XMLfiles\lab.xml";
file xmlout;
%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;
```

(f)

```
put '<?xml version="1.0" encoding="ISO-8859-1" ?>';


****READ HASH OF HASHES and SEND TEXT FILE **************;

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

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

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

            do while (hivis.next() eq 0);
                val=VISIT;
                element=vname(VISIT);
                %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);
                        val=BASETEST;
                        element=vname(BASETEST);
                        %tagon(element, val);
                        call missing(element,val);
```

(g)

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

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

                element=vname(BASETEST);
                %tagoff(element);
                call missing(element);
                end;*hites;

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

        element=vname(VISIT);
        %tagoff(element);
        call missing(element);
        end;*hivis;

    element=vname(SUBJECT);
    %tagoff(element);
    call missing(element);
    end;*hisub;

element=vname(SITE);
%tagoff(element);
call missing(element);
end;*hisit;

element=vname(STUDY);
%tagoff(element);
call missing(element);
end;*histd;

stop;
run;
```

(h)

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. Cary (NC): SAS® Users Group; 2012. Paper PO14.

2 Dorfman P, Vyverman K. The SAS® Hash Object in action. In: SGF 2009: Proceedings of the SAS® Global Forum; 2009 Mar 22–25; Washington, DC, USA. Paper 153-2009.

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 Sep 11–14; Portland, ME, USA. Paper FI 03.

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 Apr 4–7; Las Vegas, NV, USA. Cary (NC): SAS Institute; 2011. Paper 255-2011.

5 CDISC. Laboratory Data Model [document on the Internet]. Cary (NC): SAS Institute. Available from: http://www.cdisc.org/content1058 [published April 2004; cited August 2012].