Paper 301-2009

Test-Driving the Improvements to the INFOMAPS Procedure and LIBNAME Engine

Bill McNeill, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® information maps provide a layer of metadata that describes your physical data structures in terms that business users can understand. The INFOMAPS procedure and LIBNAME engine, introduced in SAS 9.1, allow from Base SAS for the creation, modification, and deletion of information maps along with querying of the data described by the information map. For SAS 9.2, these products now include enhanced functionality. This paper is written for the customers with the role of utilizing the power of information maps in Base SAS. It will test drive the new features of the procedure and the engine. Common changes to both INFOMAPS products include support on all SAS BI platforms and the use of trusted peer connections. The procedure side includes support for stored processes, enhanced INSERT and OPEN statements, and new MOVE and UPDATE statements. On the engine side are additional filter Boolean operators, surfacing of aggregate data, and new SQL dictionary tables.

INTRODUCTION

Information maps provide a layer of metadata that describes your physical data structures in terms that business users can understand. They have been used by various SAS tools such as Web Report Studio for some time. Beginning with SAS 9.1.3, information maps became available for use within Base SAS via the INFOMAPS procedure and LIBNAME engine. This paper will use portions of demonstration code and the resulting output to highlight new functionality that has been introduced to the INFOMAPS procedure and LIBNAME engine in SAS 9.2. The full demonstration code and output are available online. See the information at the end of this paper. The set up of the metadata that is needed to define the information map is not addressed by this paper. Instructions on how the metadata was set up for use in the demonstration are included in the demonstration code.

CONNECTION PARAMETERS

There are three ways to specify parameters to establish a connection to the metadata server:

1. The INFOMAPS procedure and LIBNAME engine statement
2. The SAS system options
3. A trusted peer connection

But first, there is one parameter that both the INFOMAPS procedure and LIBNAME engine require, the MAPPATH option. This option specifies the path to the information maps within the metadata repository. Throughout the demonstration, the value for this option is specified using a macro variable that is defined by using syntax similar to this:

```
%LET infomap_path=/this/is/the/path/to/the/maps;
```

Specifying the connection parameters in each invocation of the INFOMAPS procedure or LIBNAME engine looks like this:

```
PROC INFOMAPS METAUSER="CARYNT\user1"
              METAPASS=super_secret_password
              METASERVER=server.na.sas.com
              METAPORT=8561
              MAPPATH="&infomap_path";

LIBNAME emp INFOMAPS METAUSER="CARYNT\user1"
                     METAPASS=super_secret_password
                     METASERVER=server.na.sas.com
                     METAPORT=8561
                     MAPPATH="&infomap_path";
```

Specifying the connection parameters (using the group of SAS system options that begin with META) looks like this:

```
OPTIONS METAUSER="CARYNT\user1"
        METAPASS=super_secret_password
        METASERVER=server.na.sas.com
        METAPORT=8561;
```

With those SAS system options in place, the INFOMAPS procedure and LIBNAME engine statements can now be reduced to:

```
PROC INFOMAPS MAPPATH="&infomap_path";
LIBNAME emp INFOMAPS MAPPATH="&infomap_path";
```

If the METAUSER and METAPASS options are not specified in SAS system options and are not specified in the procedure or LIBNAME statement, a trusted peer connection is used. In these cases, the process ID of your current user account is used to authenticate the connection and allow access to the metadata server.

CREATION OF THE INFORMATION MAP

Let's start with defining the information map with one data source, the “EMPINFO” data set. This is done using the INFOMAPS procedure.

```
PROC INFOMAPS MAPPATH="&infomap_path";
NEW INFOMAP "Employee Info" AUTO_REPLACE=YES;

INSERT DATASOURCE SASSERVER="SASMain"
                  TABLE="HR".empinfo
                  _ALL_
                  ID="Empinfo";

SAVE;
RUN;
```

Now given the information map with data items created from the physical columns of the “EMPINFO” data source table, let's compare the columns in the “EMPINFO” data source to the corresponding data items in the information map. We'll start by looking at the CONTENTS procedure output of the “EMPINFO” data set.

```
LIBNAME HR BASE "C:\Program Files\SAS\SASFoundation\9.2\core\sample";
PROC CONTENTS DATA=hr.empinfo;
RUN;
```

Output:

```
                                The CONTENTS Procedure

      Data Set Name        HR.EMPINFO                     Observations          309
      Member Type          DATA                           Variables             17
      Engine               BASE                           Indexes               0
      Created              Wed, Oct 01, 2008 11:22:43 AM  Observation Length    216
      Last Modified        Wed, Oct 01, 2008 11:22:43 AM  Deleted Observations  0
      Protection                                          Compressed            NO
      Data Set Type                                       Sorted                NO
      Label
      Data Representation  WINDOWS_32
      Encoding             us-ascii  ASCII (ANSI)


                           Engine/Host Dependent Information

Data Set Page Size          16384
Number of Data Set Pages    5
First Data Page             1
Max Obs per Page            75
Obs in First Data Page      62
Number of Data Set Repairs  0
Filename                    C:\Program Files\SAS\SASFoundation\9.2\core\sample\empinfo.sas7bdat
```

```
Release Created             9.0202B0
Host Created                XP_PRO


                    Alphabetic List of Variables and Attributes

 #    Variable    Type    Len    Format    Informat    Label

 3    ADDR1       Char     32                         ADDRESS
 4    ADDR2       Char     32                         ADDRESS
14    BIRTHDAY    Num       8    DATE9.    DATE9.      Birth date
 1    DEPTCODE    Char      3
16    DIVCODE     Char      3
 6    DIVISION    Char     40
13    EDLEV       Num       8    8.        8.          Education level
10    EMPNO       Char      6    $6.       $6.         Employee number
11    GENDER      Char      1                          Employee Gender
15    HDATE       Num       8    DATE9.    DATE9.      Hire date
 5    IDNUM       Num       8    SSN11.    F11.        Identification Number
17    JOBCODE     Char      8
 7    LOCATION    Char      8
 2    NAME        Char     32    $32.                  NAME
 8    PHONE       Char      8    $8.                   extension number
 9    ROOM        Char      8    $8.                   Office Location
12    STATUS      Char      1                          Status
```

Now assign a libref to the location of the information map. This is done using the Information Maps LIBNAME engine, INFOMAPS. The LIBNAME engine provides a read-only way to access data generated from an information map and to bring it into a SAS session.

```
LIBNAME emp INFOMAPS MAPPATH="&infomap_path";
PROC CONTENTS DATA=emp.'Employee Info'n;
RUN;
```

Output:

```
                              The CONTENTS Procedure

Data Set Name        EMP.'Employee Info'n    Observations          .
Member Type          DATA                    Variables             17
Engine               INFOMAPS                Indexes               0
Created              .                       Observation Length    0
Last Modified        .                       Deleted Observations  0
Protection                                   Compressed            NO
Data Set Type                                Sorted                NO
Label
Data Representation  Default
Encoding             Default


                    Alphabetic List of Variables and Attributes

#    Variable                Type    Len    Format    Label

1    Address                 Char     32              Physical column ADDR1
2    Address_2               Char     32              Physical column ADDR2
3    Birth_Date              Num       8    DATE9.     Physical column BIRTHDAY
4    Deptcode                Char      3              Physical column DEPTCODE
5    Divcode                 Char      3              Physical column DIVCODE
6    Division                Char     40              Physical column DIVISION
7    Education_Level         Num       8    8.        Physical column EDLEV
```

8	Employee_Gender	Char	1		Physical column GENDER
9	Employee_Number	Char	6	$6.	Physical column EMPNO
10	Extension_Number	Char	8	$8.	Physical column PHONE
11	Hire_Date	Num	8	DATE9.	Physical column HDATE
12	Identification_Number	Num	8	SSN11.	Physical column IDNUM
13	Jobcode	Char	8		Physical column JOBCODE
14	Location	Char	8		Physical column LOCATION
15	Name	Char	32	$32.	Physical column NAME
16	Office_Location	Char	8	$8.	Physical column ROOM
17	Status	Char	1		Physical column STATUS

The first noticeable difference is the change in the variable names. The default behavior of the INFOMAPS procedure is to use the label assigned to the variable in the data set as the data item name in the information map. In the case of the data set variable “ADDR1”, its label is "ADDRESS". So, in the information map, that data item should be named "ADDRESS". It would follow that the variable in the data set surfaced by the INFOMAPS LIBNAME engine should also be named "ADDRESS". But, as the CONTENTS procedure output shows, the variable is named "Address". Something has happened here. That something is found in the NEW INFOMAPS statement of the INFOMAPS procedure. The option INIT_CAP defaults to YES. This option specifies whether to capitalize the first letter of each word in the data item name. Therefore, the data set variable "ADDR1" with the label "ADDRESS" becomes the data item named "Address" in the information map.

Also note that now the label for the "Address" variable uses the variable name from the data set, “ADDR1” in the label "Physical column ADDR1". Where the original data set variables had no assigned label, the information map data item name is the same as the data set variable name. This can be seen in the data set variables “DEPTCODE” and “DIVCODE”. The variable names from the data set are the same names for the data items in the information map.

The next noticeable difference is some of the "Employee Info" data set variable names are slightly different from the data set variable labels of the “EMPINFO” data set. Let's start with the first 2 "Employee Info" data set variables, "Address" and Address_2". Note that in the “EMPINFO” data set, the labels for these two variables were of the same value, "ADDRESS". Based on the default behavior for the procedure mentioned above, these two variables should be named the same in the "Employee Info" data set. In fact, if you look at this information map via Information Map Studio, you will see that these two data item names (that form the basis of the SAS variables names) are the same. You will also notice in Information Map Studio when you open the “Employee Info” information map you receive a warning that these two data items have the same name. The warning encourages you to change one of the names so that both can have unique data items names within the information map. The information map will work with the two data items having the same name, but the repetition will be confusing.

The INFOMAPS LIBNAME engine does not allow this condition. Notice that the variables listed in the CONTENTS procedure output all have unique names. When the information map is created, each data item is assigned a unique identifier. Normally, this identifier is based on the data item name. In the case of the two "Address" data items, the second occurrence of the same data item name is given a unique identifier. So, the first "Address" data item is named "Address" and has an ID of "Address". The second "Address" data item is named "Address" but it has an ID of "Address_2". These unique IDs are what allows Information Map Studio to display multiple data items with the same name yet still keep proper track of the data items as they are used in Information Map Studio. This is why the INFOMAPS LIBNAME engine uses the data item's unique ID as the basis of the variable name and not the data item name. This ensures that the SAS variables will always be unique and this is also why the second "Address" has a SAS variable name of "Address_2".

The last noticeable difference in the variable names is that the “EMPINFO” data set labels contain embedded spaces. When these labels were made into data item names, the embedded spaces were retained for the data item name and unique ID. But, the "Employee Info" data set variable names must correspond to SAS variable naming rules. Therefore, embedded spaces in the data item's unique ID have been converted to underscore characters. The same substitution would have happened to any other character in the name that would be illegal in a SAS variable name. This is why you should always use the CONTENTS procedure to determine the name of the SAS variables when using the INFOMAPS LIBNAME engine. The embedded spaces and other illegal characters can be retained by using the SAS system option VALIDVARNAME. By setting this option to ANY, normally illegal characters in SAS variable names are retained. But, this also means that these variable names need to be specified as SAS name literals if they contain embedded spaces or other illegal characters. Here is an example of the CONTENTS procedure output after setting the VALIDVARNAME option to ANY:

```
OPTION VALIDVARNAME=ANY;
PROC CONTENTS DATA=emp.’Employee Info’n;
RUN;
```

Output (showing only the CONTENTS variable listing):

The CONTENTS Procedure

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	Address	Char	32		Physical column ADDR1
2	Address_2	Char	32		Physical column ADDR2
3	Birth Date	Num	8	DATE9.	Physical column BIRTHDAY
4	Deptcode	Char	3		Physical column DEPTCODE
5	Divcode	Char	3		Physical column DIVCODE
6	Division	Char	40		Physical column DIVISION
7	Education Level	Num	8	8.	Physical column EDLEV
8	Employee Gender	Char	1		Physical column GENDER
9	Employee Number	Char	6	$6.	Physical column EMPNO
10	Extension Number	Char	8	$8.	Physical column PHONE
11	Hire Date	Num	8	DATE9.	Physical column HDATE
12	Identification Number	Num	8	SSN11.	Physical column IDNUM
13	Jobcode	Char	8		Physical column JOBCODE
14	Location	Char	8		Physical column LOCATION
15	Name	Char	32	$32.	Physical column NAME
16	Office Location	Char	8	$8.	Physical column ROOM
17	Status	Char	1		Physical column STATUS

UPDATE AN EXISTING DATA ITEM

To make the information map easier for all to use requires a modification to make all the data item names unique. This is done with the following code:

```
PROC INFOMAPS MAPPATH="&infomap_path";
     UPDATE INFOMAP "Employee Info";
     UPDATE DATAITEM "Address_2" NAME="Address 2"
                                 DESCRIPTION="Address line #2";
SAVE;
RUN;

PROC CONTENTS DATA=emp.'Employee Info'n;
RUN;
```

Output (showing only the CONTENTS "Address" variables):

The CONTENTS Procedure

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
1	Address	Char	32		Physical column ADDR1
2	Address_2	Char	32		Address line #2

Notice that the SAS variable name for the second address variable has not changed. Updating a data item's name will not change the unique ID associated with the data item. So, even though the VALIDVARNAME option is still set to ANY, the variable "Address_2" still contains an underscore as that is how the unique ID for that data item is defined in the information map by the INFOMAPS procedure. Also note that the data item description (which shows up as a label for the variable) has been updated.

ADDING DATA SOURCES

Now two more data sources will be added to the information map. The first new data source is the "Jobcodes" data set. The only column we want to display from this data source is the "Title" column. Therefore, the COLUMNS option is used so that just that column is made into a data item in the information map. When the relationship

between the “Jobcodes” and “EMPINFO” data sources is defined, the data sources are joined by the “Jobcode” column. Since this column is already contained in the “EMPINFO” data source, it would be redundant to include it in the “Jobcodes” data source as well.

The second data source is the “Salary” data set. All the columns in this data source will be made into data items within the information map. Here is the code to add the data source to the information map:

```
PROC INFOMAPS MAPPATH="&infomap_path";
UPDATE INFOMAP "Employee Info";

INSERT DATASOURCE SASSERVER="SASMain"
       TABLE="HR".jobcodes
       COLUMNS=( "TITLE" )
       ID="Jobcodes" /* force case of ID name; default is JOBCODES */
       DESC="Job Code Information";

INSERT RELATIONSHIP LEFT_TABLE="Empinfo"
                    RIGHT_TABLE="Jobcodes"
                    JOIN=INNER
                    CONDITION="(<<Empinfo.JOBCODE>>=<<Jobcodes.JOBCODE>>)";

INSERT DATASOURCE SASSERVER="SASMain"
                  TABLE="HR".salary
                  ID="Salary";

INSERT RELATIONSHIP LEFT_TABLE="Empinfo"
                    RIGHT_TABLE="Salary"
                    JOIN=INNER
                    CONDITION="(<<Empinfo.IDNUM>>=<<Salary.IDNUM>>)"
                    DESC="Employee Salary Information";

SAVE;
RUN;
```

FOLDERS IN METADATA

In the previous code, the “SALARY” data source has been inserted into the information map but there are no data items defined. Now the data items will be defined. But first, it would be nice to keep the data items associated with this data source separate within the information map. This does not affect the usage of the information map as it pertains to access within a SAS program. When the information map is accessed via Information Map Studio, it can help to have the data items residing in folders based on the data source. In this case, the data items derived from the “SALARY” data source will be placed in the “Salary” Info folder. Here is the code that creates the folder:

```
PROC INFOMAPS MAPPATH="&infomap_path";
UPDATE INFOMAP "Employee Info";
INSERT FOLDER "Salary Info";

INSERT DATAITEM COLUMN="Salary".salary
                NAME="Annual Salary"
                FOLDER="Salary Info";

INSERT DATAITEM COLUMN="Salary".enddate
                FOLDER="Salary Info";

SAVE;
RUN;
```

Figure 1 shows what the currently defined information map looks like in Information Map Studio. Note that the folders have been expanded to show the data items contained in the folders.

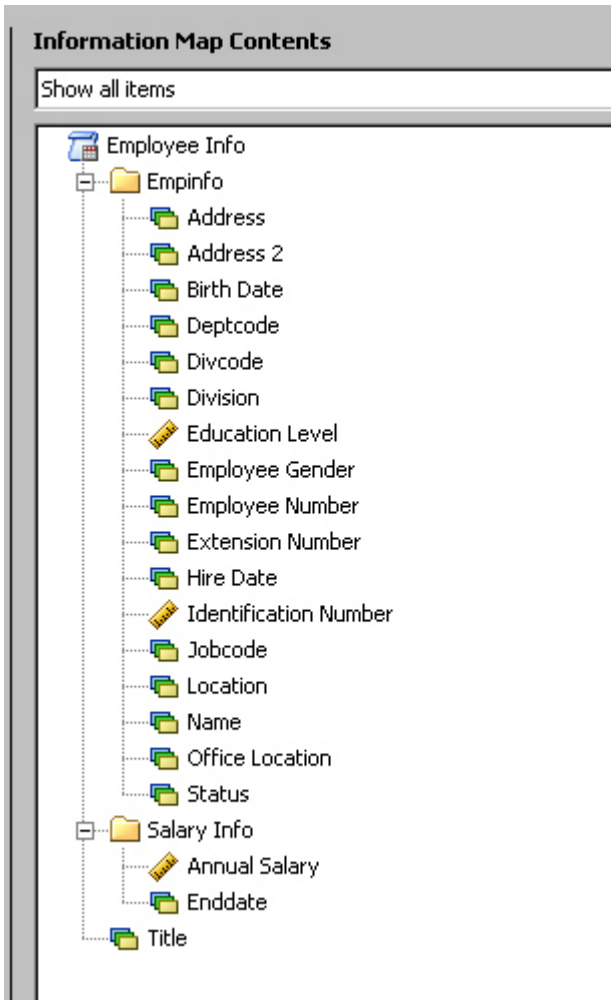

Figure 1: Folders with Data Items

The "Salary Info" folder has been created, but what created the "Empinfo" folder? When the “EMPINFO” data source was added earlier, the _ALL_ option was specified. When this option is specified, the data items created in the information map are placed under a folder named for the data source.

Now notice the difference when viewing the CONTENTS procedure output of the "Employee Info" data set. This is how the "Employee Info" information map looks when surfaced in SAS. The folder structure is ignored and the data items all appear as variables on the same level.

```
PROC CONTENTS DATA=emp.’Employee Info’n;
RUN;
```

Output:

The CONTENTS Procedure

Data Set Name	EMP.'Employee Info'n	Observations	.
Member Type	DATA	Variables	20
Engine	INFOMAPS	Indexes	0
Created	.	Observation Length	0
Last Modified	.	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO

```
Label
Data Representation  Default
Encoding             Default


                    Alphabetic List of Variables and Attributes

 #    Variable                Type    Len    Format       Label

 1    Address                 Char     32                 Physical column ADDR1
 2    Address_2               Char     32                 Address line #2
19    Annual Salary           Num       8    DOLLAR12.     Physical column SALARY
 3    Birth Date              Num       8    DATE9.        Physical column BIRTHDAY
 4    Deptcode                Char      3                  Physical column DEPTCODE
 5    Divcode                 Char      3                  Physical column DIVCODE
 6    Division                Char     40                  Physical column DIVISION
 7    Education Level         Num       8    8.            Physical column EDLEV
 8    Employee Gender         Char      1                  Physical column GENDER
 9    Employee Number         Char      6    $6.           Physical column EMPNO
20    Enddate                 Num       8    DATE9.        Physical column ENDDATE
10    Extension Number        Char      8    $8.           Physical column PHONE
11    Hire Date               Num       8    DATE9.        Physical column HDATE
12    Identification Number   Num       8    SSN11.        Physical column IDNUM
13    Jobcode                 Char      8                  Physical column JOBCODE
14    Location                Char      8                  Physical column LOCATION
15    Name                    Char     32    $32.          Physical column NAME
16    Office Location         Char      8    $8.           Physical column ROOM
17    Status                  Char      1                  Physical column STATUS
18    Title                   Char     20    $F20.         Physical column TITLE
```

ADDING FILTERS

With all the data items defined, attention can be turned to creating filters for the information map. A filter is comparable to the WHERE statement in SAS. Its purpose is to restrict the data being retrieved. To retrieve the data, an SQL procedure query is generated and executed behind the scenes. Filters that are applied to the data set that is based on the information map show up in the SQL procedure query as WHERE (or, in some cases HAVING) statements. You can see the resultant SQL query from Information Map Studio.

The difference between filters and the SAS WHERE statement is that filters are applied when the query is obtaining the data from the data source. This has the potential to reduce the number of observations read into the SAS data set. The SAS WHERE statement acts on the observations in SAS and therefore does not reduce the number of observations in the data set. The SAS WHERE statement can keep only certain observations in the data set from being used. The following code creates four filters in the information map. The first filter created is placed in the "Salary Info" folder. The second filter, "Education and Publications Departments", contains the SUBSTRN function. This function takes the first three characters of the job code and tries to match them to the values "EDU" or "PUB". The CONTENTS procedure then shows the available filters.

```
PROC INFOMAPS MAPPATH="&infomap_path";
UPDATE INFOMAP "Employee Info";

INSERT FILTER "Status is Current"
       CONDITION="<<root.Enddate>> IS NULL" folder="Salary Info"
       DESC="Currently employed";

INSERT FILTER "Education and Publications Departments"
       CONDITION='SUBSTRN(<<root.Jobcode>>, 1, 3) IN ("EDU","PUB")'
       DESC="Employees in Education and Publications";

INSERT FILTER "Host Systems Development"
       CONDITION='<<root.Division>>="HOST SYSTEMS DEVELOPMENT" '
       DESC="Employees in Host Systems Development";
```

```
INSERT FILTER "Cary HQ"
       CONDITION='<<root.Location>>="Cary" '
       DESC="Located in Cary, North Carolina HQ";

SAVE;
RUN;
PROC CONTENTS DATA=emp.'Employee Info'n;
RUN;
```

Output (suppressing the CONTENTS variable list):

```
                              The CONTENTS Procedure

          Data Set Name        EMP.'Employee Info'n    Observations          .
          Member Type          DATA                    Variables             20
          Engine               INFOMAPS                Indexes               0
          Created              .                       Observation Length    0
          Last Modified        .                       Deleted Observations  0
          Protection                                   Compressed            NO
          Data Set Type                                Sorted                NO
          Label                                        Filters               4
          Data Representation  Default
          Encoding             Default

                                   Information Maps

   FilterName                      FilterType  FilterDesc

   Status is Current               Unp         Currently employed
   Education and Publications Depar Unp        Employees in Education and Publications
   Host Systems Development        Unp         Employees in Host Systems Development
   Cary HQ                         Unp         Located in Cary, North Carolina HQ
```

In the header, a "Filters" line has been added to note the total number of filters that are available to the INFOMAPS LIBNAME engine. Note that the filter name "Education and Publications Departments" has been shortened. In SAS, filter names follow the same rules as SAS variables and are therefore limited to 32-byte lengths. So when specifying the filter "Education and Publications Departments", the shortened version, "Education and Publications Depar" must be used.

MOVING OBJECTS WITHIN THE INFORMATION MAP

For moving objects around in the information map, the MOVE statement was added to the INFOMAPS procedure. The MOVE statement can be used to better organize objects within the information map. The filters were created at the top level of the information map. It might be better to have the filters grouped in their own folder. In the code below, a folder is created and the listed filters are then moved into the newly created folder. This is followed by code that moves the "Title" data item into the existing "Empinfo" folder.

```
PROC INFOMAPS MAPPATH="&infomap_path";
UPDATE INFOMAP "Employee Info";

MOVE FILTER ID_LIST=("Education and Publications Departments"
                     "Host Systems Development"
                     "Cary HQ")
            NEW_LOCATION="Filters"/CREATE;

MOVE DATAITEM "Title" NEW_LOCATION="Empinfo";

SAVE;
RUN;
```

Figure 2 shows how the moved filters and data item now look in Information Map Studio:

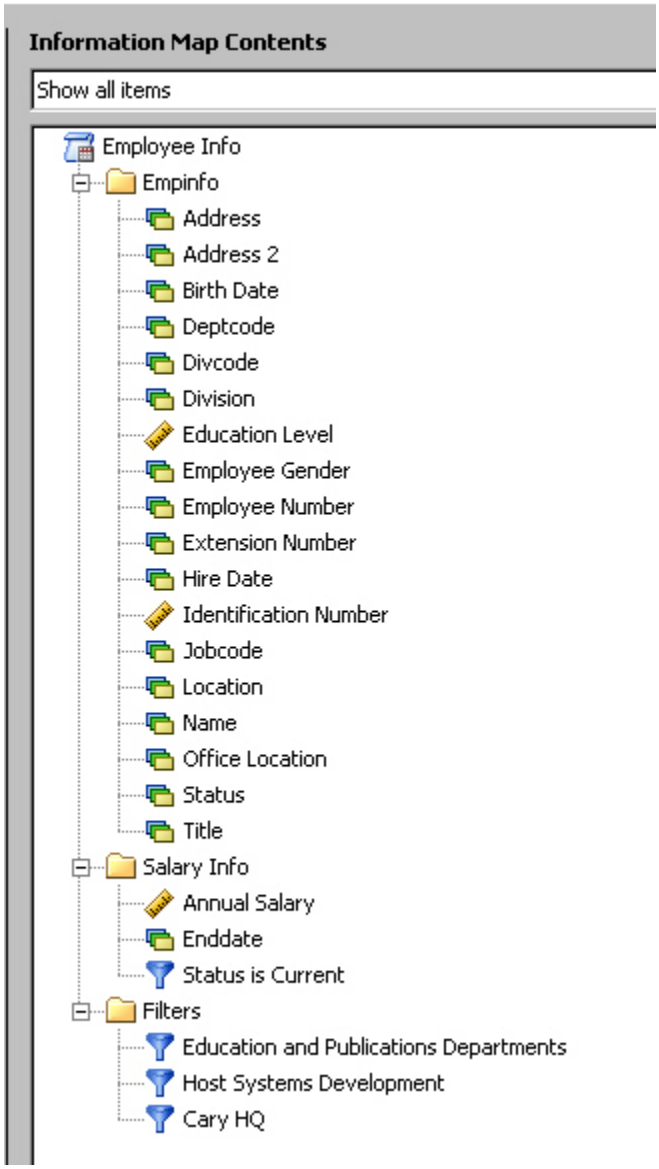

Figure 2: Folders, Data Items, and Filters

Remember that moving objects within the information map has no impact on the data set that the INFOMAPS LIBNAME engine surfaces. The INFOMAPS LIBNAME engine ignores the folder structure within the information map.

EXAMPLE OF WHERE STATEMENT AND FILTER OPTION

As noted earlier, there are differences between the SAS WHERE statement and the filter option in the INFOMAPS LIBNAME engine. This example will show those differences. Consider that an ad hoc report was requested showing all employees who have a job code of "HSD007" in the Host Systems Development Division. The filter for restricting the data to just the Host Systems Development Division employees already exists in the information map. The restriction for the job code does not exist. While a filter can be created for this restriction, recall that this is an ad-hoc report that will be run only once. Therefore, a SAS WHERE statement can be used. Here is the code:

```
TITLE "Job Code HSD007 in the Host Systems Development Division";
PROC PRINT DATA=emp.'Employee Info'n(FILTER='Host Systems Development'n);
           WHERE jobcode="HSD007";
RUN;
```

Output:

Job Code HSD007 in the Host Systems Development Division

Obs	Address	Address_2	Birth Date	Deptcode	Divcode
8	670 ATLANTIC ROAD	CARY, NC 27513	27DEC1952	HSD	HSD
9	28 SUNRAY STREET	RALEIGH, NC 27610	12FEB1947	HRO	HRO
17	1688 KINGDOM DRIVE	APEX, NC 27502	31MAY1956	HRO	HRO
21	6918 CHATHAM STREET	DURHAM, NC 27713	14MAR1952	HSD	HSD

Obs	Division	Education Level	Employee Gender	Employee Number	Extension Number	Hire Date
8	HOST SYSTEMS DEVELOPMENT	16	F	000963	1441	30JUN1991
9	HOST SYSTEMS DEVELOPMENT	18	F	000226	37	07OCT1992
17	HOST SYSTEMS DEVELOPMENT	16	F	000239	1323	10MAY1982
21	HOST SYSTEMS DEVELOPMENT	20	M	000069	1400	13SEP1991

Obs	Identification Number	Jobcode	Location	Name	Office Location
8	373-67-1896	HSD007	Cary	Eagle, Susan K.	3117
9	736-90-6683	HSD007	Cary	Fiorentino, Megan L.	7869
17	736-60-6336	HSD007	Cary	Quinones, Patricia R.	9198
21	380-57-6510	HSD007	Cary	Weber, Phil H.	9312

Obs	Status	Title	Annual Salary	Enddate
8	N	PRODECT MGR	$60,000	.
9	N	PRODECT MGR	$57,000	.
17	N	PRODECT MGR	$79,000	.
21	N	PRODECT MGR	$67,000	.

The "Obs" column notes that at least 21 observations are in the data set, but the SAS WHERE statement permits only four observations to be displayed by the PRINT procedure. In fact, there are 21 observations in the data set. These are the Host Systems Development Division records. Without any filters on the query, there would be 317 observations in the data set. Using the "Host Systems Development" filter allows for only the 21 matching observations to be read into the SAS session. Of those 21 observations, the SAS WHERE statement allows only 4 of the observations from the data set to be displayed by the PRINT procedure. The FILTER statement restricted the number of observations in the data set. The SAS WHERE statement allowed the PRINT procedure to display only 4 of the 21 observations in the data set.

Hatcher (2004) gives details about information maps and ad hoc reporting.

DATA ITEMS WITH AGGREGATE FUNCTIONS OR EXPRESSIONS

New to the INFOMAPS LIBNAME engine for SAS 9.2 is the ability to display data items that use aggregate functions or are defined with expressions. The following code updates the "Jobcode" data item to use a default aggregate function and adds the "Monthly Salary" data item to the information map. The "Jobcode" data item has the aggregate function COUNT DISTINCT added as the default aggregation. The "Monthly Salary" data item is defined with an expression. Therefore, it is a calculated data item. It does not appear in the original data source, but is created from a physical column in the original data source. The calculated "Monthly Salary" data item being created here is 1/12 of the yearly salary. Here is the code (note that AGGREGATE=YES needs to be specified as a data set option In the CONTENTS procedure statement):

```
PROC INFOMAPS MAPPATH="&infomap_path";
UPDATE INFOMAP "Employee Info";
```

```
INSERT DATAITEM EXPRESSION="<<Salary.SALARY>>/12"
                TYPE=NUMERIC
                NAME="Monthly Salary"
                FOLDER="Salary Info"
                DESC="Monthly salary computed from the yearly salary"
                CLASSIFICATION=MEASURE
                FORMAT="DOLLAR12.2";

UPDATE DATAITEM "Jobcode" NAME="Jobcode"
                           AGGREGATION=COUNTDISTINCT
                          CLASSIFICATION=MEASURE;

SAVE;
RUN;

PROC CONTENTS DATA=emp.'Employee Info'n(AGGREGATE=YES);
RUN;
```

Output (from the CONTENTS procedure except the filter listing):

The CONTENTS Procedure

Data Set Name	EMP.'Employee Info'n	Observations	.
Member Type	DATA	Variables	21
Engine	INFOMAPS	Indexes	0
Created	.	Observation Length	0
Last Modified	.	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label		Filters	4
Data Representation	Default	Aggregate Variables	5
Encoding	Default		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Default Aggregation	Label
1	Address	Char	32			Physical column ADDR1
2	Address_2	Char	32			Address line #2
19	Annual Salary	Num	8	DOLLAR12.	SUM	Physical column SALARY
3	Birth Date	Num	8	DATE9.		Physical column BIRTHDAY
4	Deptcode	Char	3			Physical column DEPTCODE
5	Divcode	Char	3			Physical column DIVCODE
6	Division	Char	40			Physical column DIVISION
7	Education Level	Num	8	8.	SUM	Physical column EDLEV
8	Employee Gender	Char	1			Physical column GENDER
9	Employee Number	Char	6	$6.		Physical column EMPNO
20	Enddate	Num	8	DATE9.		Physical column ENDDATE
10	Extension Number	Char	8	$8.		Physical column PHONE
11	Hire Date	Num	8	DATE9.		Physical column HDATE
12	Identification Number	Num	8	SSN11.	SUM	Physical column IDNUM
13	Jobcode	Num	8		COUNT {Distinct}	Physical column JOBCODE
14	Location	Char	8			Physical column LOCATION
21	Monthly Salary	Num	8	dollar12.2	SUM	Monthly salary computed from the yearly salary
15	Name	Char	32	$32.		Physical column NAME
16	Office Location	Char	8	$8.		Physical column ROOM

```
17 Status            Char   1                          Physical column STATUS
18 Title             Char  20 $F20.                    Physical column TITLE
```

With the AGGREGATE=YES option set, the output from the CONTENTS procedure has been changed. In the header, an "Aggregate Variables" line has been added to note the total number of variables that contain default aggregate functions. Also, the "Default Aggregation" column has been added to the list of variables section. This column will display the default aggregation function for each aggregated variable. The "Monthly Salary" variable has been added as the 21st variable in the data set. The "Jobcode" variable now displays the default aggregation function, "COUNT{Distinct}". Sharp eyes will also note that the "Jobcode" variable's type has changed from character to numeric. What happened? Since the AGGREGATE option is enabled, this variable no longer returns the job code (which is a character value). This variable now returns the distinct count (a numeric value) of the job codes assigned to the employees. This is demonstrated with the code:

```
TITLE "Number of unique job codes assigned to employees";
PROC PRINT DATA=emp.'Employee Info'n(AGGREGATE=YES KEEP=jobcode);
RUN;
```

Output:

Number of unique job codes assigned to employees

Obs	Jobcode
1	192

The change in variable type appears only because the AGGREGATE option is set. If the option is not set, the "Jobcode" variable would then return the actual job codes. The "Monthly Salary" variable values, as noted previously, will be 1/12 of the Salary values. To display those values:

```
TITLE "Show the Salary and Monthly Salary values";
PROC PRINT DATA=emp.'Employee Info'n(OBS=5 KEEP='Annual Salary'n
                                                'Monthly Salary'n);
RUN;
```

Output:

Show the Salary and Monthly Salary values

Obs	Annual Salary	Monthly Salary
1	$183,000	$15,250.00
2	$38,000	$3,166.67
3	$85,000	$7,083.33
4	$69,000	$5,750.00
5	$100,000	$8,333.33

COMPLEX FILTER USAGE

Before SAS 9.2, the INFOMAPS LIBNAME engine supported multiple filter usage only with the Boolean AND operator. Complex filter usage can now be created with the Boolean OR operator and the NOT operator. Also, clauses of the FILTER option can now be grouped by using parentheses. In this next example, three filters are used to create a complex filter clause. In this case the filters are restricting the data to only current employees who do not work in host systems development or do not work at headquarters. Note that due to the use of the parentheses, the NOT operator contains two filters: "Host Systems Development" and "Cary HQ". The results of these two filters are joined together by the OR operator. Then the NOT operator is applied. With the AGGREGATE option turned on, the results will show the total annual salary for each division outside host systems development and Cary headquarters. Here is the output using the filters:

```
TITLE "Total Annual Salary for Divisions outside Host Systems and HQ";
PROC PRINT DATA=emp.'Employee Info'n
               (KEEP='Annual Salary'n Division
                AGGREGATE=YES
                FILTER=(     'Status is Current'n
                         AND ( NOT(    'Host Systems Development'n
                                    OR 'Cary HQ'n))) );
RUN;
```

Output:

Total Annual Salary for Divisions outside Host Systems and HQ

Obs	Division	Annual Salary
1	CALIFORNIA REGIONAL	$96,500
2	EDUCATION	$193,000
3	SALES & MARKETING	$197,000
4	TEXAS REGIONAL	$918,000

STORED PROCESS

The INFOMAPS procedure enables you to set a stored process for an information map. A stored process is a SAS program that is stored on a server and can be executed as required by requesting applications. Once the stored process is set for the information map, it is automatically invoked when the INFOMAPS LIBNAME engine accesses the information map. Therefore, no options need to be set on the LIBNAME engine statement to use the stored process. Here is an example of setting a stored process:

```
PROC INFOMAPS MAPPATH="&infomap_path";
UPDATE INFOMAP "Employee Info";
SET STORED_PROCESS NAME="infomap_revert"
                   LOCATION="/BIP Tree/StoredProcess" ;
SAVE;
RUN;
```

SURFACING INFORMATION MAP DETAILS VIA SQL DICTIONARY TABLES

There is much information available on the information maps. The DATASETS and even the enhanced CONTENTS procedures can surface only so much. To surface the additional information, new tables were created for the INFOMAPS LIBNAME engine in the SQL procedure's dictionary tables. The new INFOMAPS LIBNAME engine-specific tables are: INFOMAPS, DATAITEMS, and FILTERS.

INFOMAPS DICTIONARY TABLE

The INFOMAPS table contains details about the information maps available via the INFOMAPS LIBNAME engine. The following code is used to see the data in the INFOMAPS dictionary table. Be careful to note that the values contained in quotes in the WHERE clause are case sensitive. The LIBNAME value always has to be specified in uppercase letters. The MEMNAME value is in mixed case. This is the default for the INFOMAPS LIBNAME engine. When in doubt on how to specify the values in the SQL procedure WHERE clause while using the dictionary tables, note how the library name, member name, variable name, and/or filter name are specified in the CONTENTS procedure output. The WHERE clause will want the same value specified within the quotes.

```
LIBNAME SQLDICT infomaps MAPPATH="&infomap_path"
                         AGGREGATE=YES;

PROC SQL;
TITLE "SQL Dictionary Table INFOMAPS";
SELECT * FROM dictionary.infomaps WHERE LIBNAME="SQLDICT"
                                    AND MEMNAME="Employee Info";
```

Output:

SQL Dictionary Table INFOMAPS

Library Name	Member Name	Information Map Name	Information Map Repository	Information Map Path	Information Map Prompt ID	Information Map Description
SQLDICT	Employee Info	Employee Info		/BIP Tree/Users/ sasbim/demo		

The output shows the details available for the information map that is surfaced as the "Employee Info" data set:

- the SAS library name, "SQLDICT"
- the SAS member name, "Employee Info"
- the information map name, "Employee Info"
- the path to the information map, "/BIP Tree/Users/sasbim/demo".

The rest of the information map information values do not contain data.

DATAITEMS DICTIONARY TABLE

The DATAITEMS table contains details about the data items contained in the data sets surfaced by the INFOMAPS LIBNAME engine. The following code is used to see the data in the DATAITEMS dictionary table:

```
TITLE "SQL Dictionary Table DATAITEMS";
SELECT * FROM dictionary.dataitems WHERE LIBNAME="SQLDICT"
                                    AND MEMNAME="Employee Info"
                                    AND NAME="Jobcode";
```

Output:

SQL Dictionary Table DATAITEMS

Library Name	Member Name	Column Name	Data Item Name	Data Item ID	Data Item Path	Data Item Classification	Data Item is Calculated?	Data Item is Usable?	Data Item Default Aggregation	Data Item Prompt ID	Data Item Description
SQLDICT	Employee Info	Jobcode	Jobcode	Jobcode	/Empinfo	MEASURE	NO	YES	COUNT {Distinct}		Physical column JOBCODE

The output shows the details available for the data items in the "Employee Info" data set:

- the SAS Library Name, "SQLDICT"
- the SAS Member Name, "Employee Info"
- the SAS Column Name, "Jobcode"
- the Data Item Name, " Jobcode "
- the Data Item ID, " Jobcode "
- the Data Item Path, "/Empinfo"
- the Data Item Classification, "MEASURE"
- the Data Item Default Aggregation, “COUNT {Distinct}”

- if the Data Item is Calculated, "NO"
- if the Data Item is Usable, "YES"
- the Data Item Description, "Physical column JOBCODE"

The rest of the data item information values do not contain data.

FILTERS DICTIONARY TABLE

The FILTERS table contains details about the filters available to the data sets surfaced by the INFOMAPS LIBNAME engine. The following code is used to see the data in the FILTERS dictionary table:

```
TITLE "SQL Dictionary Table FILTERS";
SELECT * FROM dictionary.filters WHERE LIBNAME="SQLDICT"
                                   AND MEMNAME="Employee Info"
                                   AND NAME="Cary HQ";
```

Output:

SQL Dictionary Table FILTERS

Library Name	Member Name	Filter Prompt Usage	Prompt ID with Filter	SAS Name for Filter	Filter Name	Filter ID	Filter Path	Filter Description
SQLDICT	Employee Info			Cary HQ		Cary HQ	/Filters	Located in Cary, North Carolina HQ

The output shows the details for the filters available for use with the "Employee Info" data set:

- the SAS Library Name, "SQLDICT"
- the SAS Member Name, "Employee Info"
- the SAS Name for Filter, "Cary HQ"
- the Filter Name, "Cary HQ"
- the Filter ID, "Cary HQ"
- the Filter Path, "/Filters"
- the Filter Description, "Located in Cary, North Carolina HQ"

The rest of the filter information values do not contain data.

CONCLUSION

Information maps are a powerful tool. The use of information maps in Base SAS is new as of 9.1.3 and 9.2, and it brings many modifications. Therefore, now that you know how to use this tool in Base SAS, it will enhance THE POWER TO KNOW®

REFERENCES

Hatcher, Diane. 2004. “Designing Information Maps for Ad Hoc Reporting.” *Proceedings of the Twenty-Nineth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available at http://www2.sas.com/proceedings/sugi29/104-29.pdf

SAS Institute Inc. 2009. *Base SAS 9.2 Guide to Information Maps* Cary, NC: SAS Institute Inc.

RESOURCES

You can download both this *SAS Global Forum 2009* paper and the program that it uses from http://support.sas.com/saspresents

ACKNOWLEDGMENTS

I am indebted to the INFOMAPS team for their invaluable help putting together this paper:

Sue Her and Helen Wolfson (INFOMAPS procedure)

Carlotta Hicks and Giselle Smith (INFOMAPS LIBNAME engine)

CONTACT INFORMATION

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

Bill McNeill
SAS Campus Drive
SAS Institute Inc.
E-mail: Bill.McNeill@sas.com