ROUND PEGS INTO SQUARE HOLES:
DATA WAREHOUSES FOR THE HARDWARE IMPAIRED

Michael Davis, Bassett Consulting Services, Inc., North Haven, Connecticut

Abstract

One of the dirty secrets of most data warehouse projects is that they require vast amounts of disk drive space and robust computers for successful implementation. Unfortunately, not every organization that lusts for a comprehensive data warehouse can finance the tab for the hardware that may be required to create the warehouse of their vision.

Bassett Consulting Services has helped two clients make better use of limited computing resources through a two step approach. First, an active data dictionary permits "point and click" selection, renaming, joining, and sub-setting of data, even when some of the elements reside on tape cartridges or in hierarchical files.

Then "code engines" create SAS code on the fly to extract the requested data from their actual physical locations and transform the data into SAS data sets. These data sets even contain variable labels and formats, courtesy of the data dictionary, and are ready for further analytical processing.

Introduction

When Bassett first began creating SAS/AF® FRAME applications, our custom was to incorporate SAS data step program code through SCL SUBMIT blocks, which were compiled along with the rest of the FRAME catalog entries. We became dissatisfied with this approach for several reasons.

First, our clients were asking for "point and click" selection of variables. Within the programming structures that we used, it was not possible to comply with their requests without custom programming for each new data source. Compounding the problem was the realization that the data steps incorporated into our applications involved much repetitive code. Bassett wanted to employ an approach that would facilitate the adoption of object-orient programming techniques.

Second, it seemed that the SAS System's best features for developing "point and click" interfaces required that the data to be in a SAS data set or relational database management system (RDBMS) and stored on disk. Unfortunately, our clients could not afford to devote the disk space necessary to permanently store their data warehouses on disk.

Third, looking towards the future, Bassett wanted to position the applications it developed so that when Version 7 is released, SAS variable names up to 32 characters long can be accommodated without re-coding and recompiling. Similarly, when it was necessary to create time-series data sets, we wanted to be able to look-up a five character variable prefix easily. The data dictionary approach seemed a good solution.

Fourth, it was clear that no matter how well any data warehouse might be designed, there would always be some essential information not catalogued in the data warehouse. Thus a data consolidation structure should be created that would allow our clients to integrate data from SAS data sets, RDBMSs, and flat-files stored on disk or tape into tables for subsequent analysis and reporting.

This was the impetus for developing the Meta-Master© utilities and the Data Integrator© interfaces. The Meta-Master utilities consist of a data dictionary and SAS/AF applications designed to populate the dictionary, maintain the dictionary, and to define objects called variable lists.

Figure 1 on the following page shows the relationship between Data Integrator and Meta-Master. Underneath the user interface layer of Data Integrator are the Meta-Master data dictionary, the Meta-Master utilities, and the Data Integrator code engines. The data dictionary is where the details for each data element are stored. The Meta-Master utilities are used to automatically populate the data dictionary and to maintain it.

The core of Data Integrator is composed of "code engines". These engines write SAS code to extract, sort, combine, and manipulate data as specified by lists called "job streams". These job streams are specified to through Data Integrator's "point and click" interface, made possible by the Meta-Master data dictionary.

To better understand how Meta-Master and Data Integrator work, let's examine how variable lists and job streams are defined.

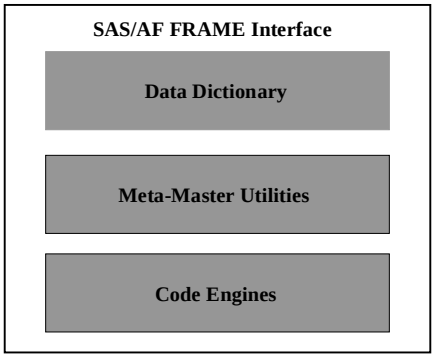

Figure 1

Variable Lists

For the Meta-Master data dictionary and utilities to properly function, each data element must have a unique identification. Meta-Master uses the SAS System two-level convention (<libref> **.**<member>) to specify to which table data element (variable) belongs. This is illustrated in Figure 2. Since the way in which Meta-Master employs this form of categorization has some unusual features. Let us look at these features.

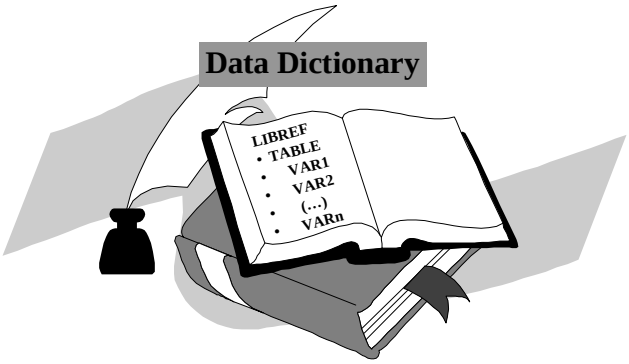

Figure 2

SAS users may be already familiar with the term libref. A libref is an alias which tells the SAS System the physical location of the library where it can find a specified data set member. However, since the Data Integrator system is designed to reference flat-files in a manner that is largely transparent to its users, Meta-Master extends the definition of libref to include a single flat-file or related series of flat-files.

As an example, let us suppose that the Meta-Master routines were to be employed by a health insurer. We might have a series of patient claims data sets, one for each month. Rather than having to redundantly define the data elements in the various claims tables for each month, we can use a single libref (and table) to represent a typical month. Then, we merely have to tell Data Integrator for which month (and year) we desire our data.

Our use of term “table” is similar to the SAS data set member. We decided to refer to data set members as tables for two reasons. First, our definition of tables should be viewed as logical or symbolic, while SAS data sets members are but one form of physical representation. Second, we wanted to reserve the possibility that we might extend the Meta-Master utilities in the future to allow multiple tables to point back to a single SAS data set, as is currently permitted with flat-files.

The term “variable” refers to a single data element or SAS variable. The combination of libref, table, and variable points to a unique data element in our system of categorization. However, during testing of the Meta-Master prototype on a rather large table with several thousand variables, it was decided to provide two additional means to sub-categorize variables within a table.

A variable “category” classifies similar variables for ease of selection. Where variables are distinguished only by the time period each variable represents or its position in a series, an “index value” is assigned to it in the data dictionary. It is important to note that the use of categories and index values in the Meta-Master system is optional and that categories and index values are only used to subset the number of variables shown for a table to speed selections.

Variable List Structure

The heart of the variable list is a list of the variables to be included in a particular job step. When a variable list is created, it is stored in a SAS catalog as an SLIST entry. When a variable list is created, the person creating the list is encouraged to supply a narrative description to be stored with the SLIST catalog entry. The structure of the variable list is illustrated in Figure 3.

Although all of the code engines created for the Data Integrator thus far handle only one libref and table per variable list, Meta-Master stores the libref and table for each variable in the event that it becomes desirable to relax this restriction in the future.

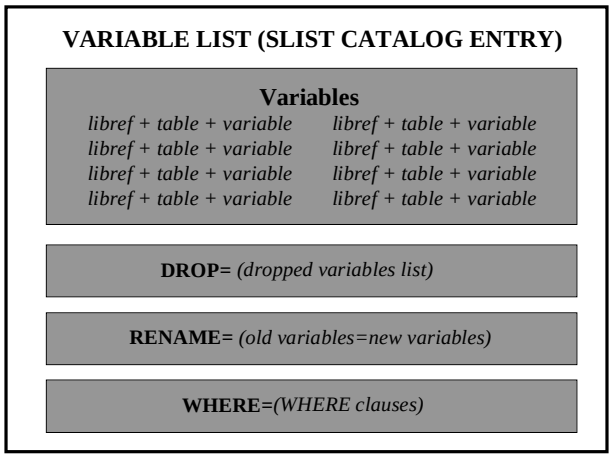

Figure 3

Other information is stored along with the list of variables for those job steps that may require that information. These pieces of information are used to set SAS data set options.

For example, if a job step requires that some incoming variables be renamed to prevent "collisions", a list of original and renamed variables can be stored in the variable list. Variables that are not required for the current step can be dropped. If WHERE processing is required, it can be applied to only the data sets that require it and before the current observation is brought into the job step.

Variable lists are managed by the Meta-Master utilities and routines. These list are referenced by the Data Integrator application, which invokes SAS/AF FRAME interfaces and code engines written in SCL (Screen Control Language). Data Integrator uses Job Stream lists to combine these elements and accomplish data extraction and manipulation.

Job Streams Lists

Each job stream list consists of at least one step and can have as many steps as desired. Job streams are stored as nested lists, in job step sequence, within a SLIST catalog entry. The structure of a typical job streams is illustrated in Figure 4.

Data Integrator processes a job stream one step at a time. The type of step is specified by the list name under which the job step was stored. At the time this paper was prepared, Bassett had created code engines to write SAS statements and other types of code for the following types of job steps:

- INFILE
- SET
- MERGE
- SORT
- TEMPLATE
- COPY
- DOWNLOAD (to spreadsheet)

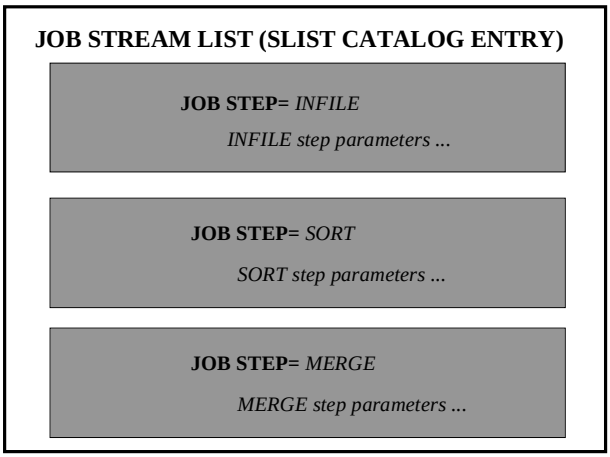

Figure 4

Some of the code engines execute on the desktop computer on which Data Integrator runs. Other code engines take advantage of the SAS System's ability to distribute program logic and submit code to be executed on a remote computer, such as an MVS mainframe.

What do the different engines do? The INFILE engine writes a data step to extract a list of variables from a disk or tape flat-file. The SET engine brings in one or more SAS data sets for sub-setting and transformation. The MERGE engine brings in two or more data sets and combines them using BY variables as the merge key.

The SORT engine does what one expects. It sorts a single SAS data set. However, one can use this engine to rename or drop variables and to subset the observations copied to the output data set. The output data set need not be the input data set.

The TEMPLATE engine was developed to accommodate those situations where complex data step logic was required. Rather than develop a convoluted menu structure to handle all potential situations, the user keys a program template. For some applications, the template is inserted exactly as keyed. However, the TEMPLATE engine shines when Data Integrator uses the data dictionary to "fill in the blanks". The user merely keys in the word INPUT on its own line and lists the variables to be

included in the data step. Data Integrator looks up the variable offsets and informats required to fill out the INPUT statement. Thus a piece of program code that looks like this:

```
INPUT
  name
  address
  city
  state
  zip
;
```

is automatically translated to:

```
LENGTH
  name    $ 20
  address $ 25
  city    $ 20
  state   $ 2
  zip     $ 9
  ;
INPUT
  @00001 name     $char20.
  @00021 address  $char25.
  @00046 city     $char20.
  @00066 state    $char2.
  @00068 zip      $char9.
  ;
FORMAT
  name    $20.
  address $25.
  city    $20.
  state   $ 2.
  zip     $ 9.
  ;
LABEL
  name=    "Customer Name"
  address= "Street Address"
  city=    "City"
  state=   "State"
  zip=     "Zip Code"
  ;
```

As shown in the preceding example, Data Integrator supplies LENGTH, FORMAT, and LABEL statements, using information stored in the data dictionary.

The COPY and DOWNLOAD engines were developed for a client which routinely stores needed data on an MVS mainframe. After using Data Integrator to create and remotely submit a MVS batch job to extract the data, the COPY engine submits a remote SAS program to copy the extract data from tape to disk.

Once the data has been copied to disk, the DOWNLOAD engine downloads the data to the desktop computer. Then the SAS/ACCESS® Interface to PC File Formats converts the downloaded data to a Microsoft Excel .XLS spreadsheet file.

Using the same technology, other data step processing or procedures can be accommodated. Further, jobs to be remotely submitted to any platform supported by the SAS System, including operating system and utility program statements, can be cloned from existing engines.

Additional Job Step Parameters

What kinds of additional information do the code engines require? It depends on the code engine. As an example, let us consider the SET code engine when the job is to be both created and run on the desktop platform. Five items must be specified:

- output libref
- output member
- drop list
- IF clauses
- variable lists

The output libref and output member are inserted into the DATA statement created by the SET code engine. The drop list allows variables drawn from input SAS data sets only for use by a sub-setting IF statement to be dropped at the completion of the data step.

A SET job step may contain more than one variable list since a SET statement can specify more than one SAS data set to be incorporated into a data step. Data Integrator loops through all the requested SAS data sets as it builds the SET statement. The data set options specified within each variable list are inserted in the SET statement after SAS each data set.

Data Dictionary Features

One feature of the Meta-Master Data Dictionary is the ability to store the data type for each element. Data Type can assume the following values:

- data - SAS data set
- view - SAS view
- flat - disk flat file
- tape - tape flat-file

Libref, Table and Variable Name have been discussed previously. Variable Type can be either character or numeric. Label is a forty character variable which contains the SAS variable label.

Transformation is a unique feature of the Meta-Master Data Dictionary. For example, a tape data set might have a transaction date, representing a Julian date, that is stored as a packed decimal. In our applications, we want to use it as a SAS date value. How can we resolve this conflict?

When the code engines used to extract data are run, they inspect the value of the Tranform(ation) data dictionary field. If it is not empty, Data Integrator creates an assignment statement which incorporates the transformation. In our example, the field on the tape might be read using the PD3. Informat. The appropriate transformation would be

DATEJUL(trandate)

and Data Integrator would build the following assignment statement in the created data step:

trandate= DATEJUL(trandata) ;

While transformations were incorporated into Meta-Master for the purpose just described, they can be used to solve other problems. These uses include summing variables across an observation and counting observations either conditionally or unconditionally.

To illustrate the creation of a true/false variable, if we had entered a transform for a numeric variable, *testco*, as

INPUT(UPCASE(company),'BASSETT')

where *company* is a character value containing the names of companies, testco would be true when company contained the word "BASSETT" .

The key thing to remember about transformations is that once they are entered into the data dictionary, Data Integrator's code engines become smart enough to remember the desired processing logic whenever that SAS variable is selected.

With respect to the other fields shown on the screen shown in Figure 6, the Informat, Format, and Length fields are familiar to SAS users and inherit their features. Offset is the starting position of the field to be read to create the SAS variable. It is required when reading data from a tape or disk flat-file.

Autoloader

The Meta-Master utilities would be far less useful if it were necessary to key in the dictionary information by hand. As observed in a preceding discussion, some of our clients routinely work with tables containing thousands of variables. Fortunately, one of the Meta-Master utilities is a FRAME application that automatically loads the data dictionary for a new table.

How is this done? For almost any table we might insert into the data dictionary, we have one of three items. We might have a copy of the data set or one with a similar structure. When this is not the case, we almost surely have a SAS program with an INPUT statement coded to read the raw data.

Last, if the raw data was created by a production program, we might have access to a COBOL record description for the raw data file. In this case, we employ the COB2SAS utilities available at no cost from SAS Institute's Technical Support department. COB2SAS is used to translate the record description into SAS INPUT and LABEL statements, which are fed to the Autoloader.

How does the Autoloader work? In all cases, Meta-Master reads the SAS System dictionary views to determine the table layout and to populate the Meta-Master data dictionary. In the case where a table is drawn from a SAS data set or view, all of the required information can be obtained from the SAS System dictionary views.

In the case of tape and disk flat-files, two additional items must be obtained. These two items are the variable offsets and informats. This information is obtained from parsing the INPUT statement.

To use the Autoloader to define a table created from a flat-file, the user enters the name of the file where the SAS program containing an INPUT statement for the file is located. An editor window is opened and the file is loaded. The user deletes the DATA and INPUT statements since the Autoloader

supplies them. Other statements, such as LABEL, FORMAT, INFORMAT, and LENGTH statements are left in the editor.

In addition to creating the SAS system dictionary views and parsing the INPUT statement for variable offsets and informats, the Autoloader identifies appropiate variable formats based on the informats and variable lengths.

Building Job Streams

The user clicks on a radio box station to select the next job step to be built. Clicking on the REVIEW job stream station prints a detailed list of the job stream under construction to the SAS log. If the wrong job step is selected, clicking on the UNDO last step button backs out the last step entered.

To keep users from wondering if the job step they just entered was accepted, a "stack" displays the job steps that have been previously entered.

Running Job Streams

One feature of this interface is the librefs and filerefs associated with the selected job stream are verified before the job stream is actually executed. If a libref or fileref is unassigned, a pop-up window allows the user to make the assignment or cancel execution before any time is wasted running the job stream.

Custom Interface for Remote MVS Batch Jobs

For one client, Bassett created a custom interface and code engines to meet their requirements. The data they wished to extract was stored in tape cartridges on an MVS mainframe. The data sets were very large, typically occupying 50 or more IBM 3480 tape cartridges. Their analysts needed the ability to extract data for a single client and download it for analysis with a Microsoft Excel spreadsheet.

When the client was shown the a prototype Data Integrator interface that allowed them to define and store job stream lists, their reaction was that the interface was needlessly complex and confusing. To satisfy their concern, we modified the Data Integrator interface to build a disposable single step job stream list. Thus users at this site only have to create and manage variable lists.

The INFILE code engine was modified to submit the extraction program in batch mode, along with the required MVS JCL. To minimize both contention for the tape data sets and execution time, the code engine also inserts a SyncSort® extraction program ahead of the SAS data step.

As part of this customized version of Data Integrator, the client was furnished with a utility FRAME application that catalogs which clients are stored on each tape cartridge of selected tape flat-files. This information is used by the SyncSort extraction program so that only the cartridges containing the selected clients information are mounted and read.

Conclusion

The Data Integrator and Meta-Master represent an object-oriented approach using SAS System software to rapidly develop custom data extraction and manipulation applications. These applications may be used to populate data warehouses or for other purposes.

Since the analysis data does not need to be retained on disk storage, this approach should be attractive to SAS System users who may be working under hardware or budget constraints yet do not want to sacrifice the "point and click" interfaces possible using SAS Institute software.

An ancillary benefit of this approach is that a data dictionary is created. This dictionary is a good tool for managing a data warehouse. Because the dictionary is stored as a SAS data set, it is relatively simple to interface it to other SAS applications.

Acknowledgments

The author may be contacted at:

Bassett Consulting Services, Inc.
10 Pleasant Drive
North Haven CT 06473-3712
Telephone: (203) 562-0640
Fax: (203) 498-1414
Internet: 0002395748@mcimail.com