You’ve just bought a Data Warehouse. Now what?
Stanley Fogleman, Harvard Clinical Research Institute, Boston, MA

ABSTRACT
One of the ways to get involved in implementing Data Warehouse technology (DI Studio) is to pick a pilot project, in this case, some SAS® datasets supplied by an outside vendor which needed to be mapped to an existing Access database. At first, it appeared to be an example of using a sledgehammer to kill a fly. As the project progressed however, we realized that a number of peculiarities of Access databases (no native support for formats, peculiar representation of dates and representing true/false/no answer) made DI Studio an ideal tool for identifying and resolving those issues.

INTRODUCTION
As usual in the data processing world, we discovered on short notice that some fairly detailed analysis needed to be done on a subset of data provided by an outside vendor. The problem was that the columns needed were spread out among 20 or more SAS datasets and needed to be mapped to an existing Microsoft Access database. Needless to say, neither the designer of the SAS datasets nor the Access database had any notion that any such thing would ever need to occur. The SAS datasets provided by the outside vendor were almost a complete mystery. Once the existing variable names and structure were discovered, we had the added surprise that the vendor had renamed, resized and redefined variables without notice. Above all else, this was a mapping problem (in most instances one variable on the SAS dataset needed to be mapped to a corresponding variable on an ACCESS table). One lesson I retained from the DI Studio class I had taken earlier was that mapping was something the tool was exceptionally good at.

USER REQUIREMENTS
Since in essence we were mapping variables from many tables (the SAS datasets) into a few tables (the access tables), we needed some kind of mapping tool outside of DI Studio to produce a spreadsheet of SAS variables to access variables. Note: Of course, the reverse could have been true. Again, my instructor in the DI Studio warned us against “flying without a map”. This process was done from both the table to table mapping and variable to variable mapping. Otherwise, it was all too easy to lose the forest for the trees during setup of the process flows in DI Studio. The “table to table” mapping was done first. The Data Dictionary of the SAS datasets and the Access tables were loaded into a custom Access database that allowed the user to associate specific variable names by their SAS label values. The variable names were not self-documenting, so this helped in assigning mapping values. After selecting a SAS dataset name, only the variables for that dataset would appear. This was also true for the Access table names. Once the variables were “narrowed down” the “from” variable and the “to” variable could be selected and a row would be added to the access mapping table which would be the result of this endeavor. This produced a mapping document that made it easy to determine 1) which tables would need to appear in a particular process flow and 2) which variables needed to be mapped. In addition the mapping document would list whether the sending field was of the same data type as the receiving field.

A WORD ABOUT SAS VS. ACCESS VARIABLE DIFFERENCES
Formats do not exist as such in Access (i.e., there is no built-in functionality to associate a variable value to a “decoded” value). In a sense, then, it is pointless to spend much time on trying to bring the decoded values over. Since the majority of the time the data was intended for use in statistical analyses, we decided to keep the raw values and provide the end users with a dictionary of decoded values. In SAS, we are used to a numeric value of zero meaning “false” and any other value resolving to “true”. In the Access world, there are no such associations, and any meaning must be derived programmatically. So in some instances we would need to map “true”, “false” and “no answer” values to their corresponding programmatically expected values in the Access world. The Data Validation transform was particularly useful for that. Dates presented another challenge. In the access world, there are only datetime values, so it was necessary to append the time value for midnight to any date field where the time was not specified so the values would transfer correctly. Another challenge was whether the Access database was set up to accept null values in particular fields.
DI STUDIO LIMITATIONS (SELF-IMPOSED)

We intentionally used a limited toolset (data transformations) for DI Studio in part because of our unfamiliarity with the application and in part to keep debugging as simple as possible. The only Data Transforms used were SQL Join, Data Validation and Loader. We needed to be able to isolate and resolve problems quickly and I was afraid as a developer of getting in over my head. Hindsight being 20/20, it might have been prudent to take advantage of the reporting options which the tool offered, particularly in the “data exception” area (to catch missing dates or blank responses, for example.)

DI STUDIO LIMITATIONS (OF THE TOOL ITSELF)

We originally had the idea to use the tool as an interactive development facility, but it is more correct to say that the tool was designed to produce job flows to run in batch mode. Since we had no guarantee that the vendor files would have the same skeletal structure over time (and in fact, variables were added and renamed in the course of the study!), it was unrealistic to assume that we would be able to use the “write once – run many” analogy for which the tool was obviously designed. Another limitation we ran into was for the support of user-defined formats. The tool was designed with one of two options: 1) to use a system defined library for a format catalog 2) to use a search path to search through several catalogs. Neither of these options was particularly appealing. The first option presumed that there is some kind of “master format catalog” in use at the site. The second option assumes unique naming of formats, something on which I would not wager on. SAS Tech support was nice enough to provide us with a custom transformation to add a libname “on the fly” to the job in question. I found it confusing to use the “Source Designer” for Target datasets. SAS Tech Support clarified this somewhat by pointing out that the Target designer was designed for “newly minted” SAS datasets. Since the ACCESS table was very much in existence at the time the project started, the “Source Designer” needed to be used in its place. Confused? So was I.

BUILDING THE DATA MART

Once the values were mapped and spreadsheets produced showing the source dataset name and variable to the access table and variable, it was possible to start building data marts. A data mart is defined as a set of dimensional tables supporting a business process¹. To keep it from becoming overly complicated, it was decided to create flows by their ultimate target. For example, one area of interest was records pertaining to hospitalization. All the SAS datasets which fit this category were placed in a job process. Also, for the release of DI Studio we were using (3.3), SAS Tech Support discouraged us from making the job flows too dense because of the amount of metadata which needed to be moved from one repository to another during the “check in” and “check out” process. I understand this has been improved considerably in more recent releases of DI Studio.

An analogy that might be helpful is a single page of a flowchart equating to a job process. (Of course, job processes can have hundreds of files and job processes, but this is a much simpler example.)

A small amount of housekeeping: First the libraries have to be created and defined using management console. The analogy is to using a libname statement in a SAS program. The metadata for the files needs to be imported after the libraries have been defined. Then, also in Management Console, a custom or project repository should be defined. A useful way to think of the repository is as a large container for all of your job flows. I used a project repository because I wanted to have the control capabilities afforded by a project repository. Since we have both an Application Server as well as a Metadata Server, I used Relative naming to refer to the location of the folders containing the Project Metadata Repository. The temptation (in Windows, at least) is to refer to drives by their Letter designations. This will cause you nothing but grief, as the server which accesses this data may have no letter drive mapped to it. Also, I wanted to be able to “check in” the process flows at the end of the project. (Note: SAS Tech Support advises that future releases of DI Studio may not have a custom repository). Then a new metadata profile pointing to the project repository was defined. Also, we discovered that the vendor supplied datasets did not have a unique identifier that we wished to use as a standard key (it was only available on one of the sas datasets and available through a pair of columns to all of the subsidiary sas datasets). We could have chosen to do a join for each of the tables, but for simplicity’s sake, we elected to create a sas dataset view outside of DI Studio to get the one variables we needed on all of the source sas datasets. This did much to make the flow simple and straightforward. As it turns out, the views are treated as dataset objects, so there were no additional headaches once we went down this path. (Other than remembering to build them each time the new batch of sas datasets from the vendor arrived.)
In DI Studio, I would then log in using the newly created metadata profile and create “empty” process flows using the “Process Designer” tool. After this point, a series of repetitive steps occurred. The ACCESS dataset icon (representing an ACCESS table) was dragged into the process flow window. (An alternate way of doing this would be to use the “Source Designer”. This automatically generates a “loader” icon pointing to the table that was dropped and an empty container from which the corresponding SAS dataset is “dragged and dropped” from their source library onto a project flow. The loader icon is clicked on and a dialog box with several tabs appears. Select the “Mapping” tab. This will display the two tables side by side. (Note: If there are any variables in common which are of the same type and the “auto-mapping” feature is turned on, mapping will occur automatically). Using the spreadsheets created by the ACCESS application mentioned in the previous section, all the “SAS” variables that needed to be “carried forward” were mapped to their “ACCESS” counterparts. Mapping can be done in the tool in a myriad of ways, but the most straightforward were to click and highlight the source column and click and highlight the target column and select “new mapping”. Another equally valid way is to highlight a source column and then drag an arrow from the source column to its target. At the time the variables were mapped, if the two data set types or lengths are inconsistent, a “data expression wizard” will appear from which you will have the opportunity to enter a SAS function. For example, I transformed a one character string into a number using the function:

\[
\text{INPUT}(\text{PUT(MY\_SAS\_CHAR\_VAR},1.),1.)
\]

Also frequently used was the function: to turn a SAS date into an access datetime:

\[
\text{DHMS(MY\_SAS\_DATE\_DT,0,0,0)}
\]

Some of the fields (containing comments) on either end were much wider than necessary to accommodate the data they contained. Substringing the variables worked on the sending end to “shorten” the field to avoid a mapping warning. I also made use of the “Data Validation” process as a way to deal with peculiar situations. For instance, a yes/no/no answer field might be character on the sending (SAS) end and a blank might represent “no answer”. It was being mapped to a numeric field on the Access end, so it was necessary to substitute a numeric value (zero, for example) for a space to avoid a load error when the process flow ran. For this I used the “custom validation” tab. Clicking on the “new” button offers you the range of variables on the particular dataset in question. There is also a “condition” field to specify the test. There is a rudimentary expression builder, but I found it faster to code the test once and copy the condition into notepad and use “cut and paste”. Amusing note: At times, I was making heavier use of notepad than I was of DI Studio! One of the problems in using the expression builder is that the wealth of functions available to you makes it difficult to find the one you’re looking for! I found it better to have function selection “worked out beforehand” because it was cumbersome to have to stop and troll for the correct one when one was focused on mapping the data(not necessarily a complementary thing!) The search was somewhat complicated by the fact that the numeric fields seemed to have various amounts of padding added to them. (So the number which was supposed to be in the field was not necessarily left-justified, which is what I would have expected!) Using the TRIM and LEFT functions in combination made this a lot easier. For complex function calls, it was necessary to “eyeball” the resulting function because the wizard has no validation tool “built in.” It is a good idea to have a parallel SAS session running for quick validation of function calls. I ran into the situation all too often where I left a period or comma out and would fail on a syntax error when the flow would execute. Looking through the sas log generated by the execution could be a daunting task. Also, sometimes the SAS function would not work as I expected it to(i.e., would produce no result or a different result than the one I would have liked), so it was necessary to dummy up some data in a SAS session until the SAS function was working as expected. I am hopeful that future releases of DI Studio will have some coding intelligence built in for the expression builder. It is awkward, to say the least, to have a parallel sas session running at the same time as DI Studio. The tool allows for a discrete action depending on whether the condition is true or false. It also allows for you to report that the condition has been reached in an exception report. For simplicity’s sake, I only coded for a “true” condition (there were only three possible values for this field, anyway), and did not utilize the exception report. As each step was completed, I ran the flow inside DI Studio and debugged as necessary.
PECULIARITIES
The SAS datasets originated in Europe, so the dates had peculiar formatting. We noticed that the European date format carried over to the ACCESS database, even though we made no effort to do so! We also noticed that DI Studio attempted to build indexes on certain fields, even though we had not asked it to do so! ACCESS would complain when SAS tried to drop a table that was already empty – this was frustrating if we encountered this mid-stream and had to start over. We eventually decided to use “truncate” for the Access tables which eliminated all the rows prior to loading the data. For the load technique in DI Studio, the default option was to append, so we had to be very careful to check this for all of the tables to make sure that we were refreshing the tables instead.

CONCLUSIONS
Use of DI Studio imposed a certain discipline on the project that would have been found wanting otherwise. Since there were three people involved in the project (one from a different department), all with competing schedules and deadlines, it was important that tasks and responsibilities were allocated discretely and roles reviewed at a team meeting do that we were not interfering with each other and also spending time in a productive fashion. By having several “mock loads” of the database, we were able to narrow down our list of issues to a handful(e.g. missing data, unexpected data). This helped us focus our time on eliminating known problems and gave us a certain confidence that data was being loaded correctly to the access tables. DI Studio could be improved by providing a mapping facility similar to Information Map Studio and adding the ability to validate SAS expressions “on-the-fly”. One of the hardest things about learning to use the tool is the requirement to think at a higher level of abstraction than would be necessary if you were a single SAS coder “pushing out code”. I found that during the process of having two systems learn to talk to each other, I was forced to ask questions about the data that might have been ignored or passed over in a conventionally coded process. For example: Do we need to archive the access tables created each time the process runs. There is a natural tendency as a programmer to want to “get to the coding” as quickly as possible. It is more often the case than not in using DI Studio that one must spend more time in planning than in execution. I believe the real value of using a data warehousing tool is in getting an organization to behave differently which will not come about (sadly) until several projects have been completed or are underway. If Data Integration Studio had not been used for this project, we would have lacked a discipline that the software imposed on us. We would have been in danger of having the project collapse of its own weight due to ever-changing user requirements. As it was, we were able to “reign in” change requests from the users to those which would have a genuine benefit to the outcome.

REFERENCES:

ACKNOWLEDGMENTS
SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.
This paper is dedicated to the memory of Randy Scott Maloney (1957-2007), a colleague and lifelong friend.

CONTACT INFORMATION
In case a reader wants to get in touch with you, please put your contact information at the end of the paper.)
Your comments and questions are valued and encouraged. Contact the author at:

Stanley Fogleman
Harvard Clinical Research Institute
930 Commonwealth Ave West
Boston MA 02215

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *