Paper 126-2008

Business Process Intelligence

How to Manage Data Integration and Measure ETL Successes

Danny Grasse, ThotWave Technologies, Chapel Hill, NC

ABSTRACT

The more 'Enterprise' you want in your Enterprise Business Intelligence initiative, the more confidence you should have in your integrated data. Given a heterogeneous collection of data sources, it should be a given that the owners of a BI system 1) discover data problems as soon as they occur upstream, and 2) notify the right resources about those issues when they occur, then 3) restart the data integration process after corrective action has been taken.

This paper expects to show how SAS®' Data Integration Studio (DI Studio) can be used both natively, and with custom transforms around a macro library, to help a development team manage a data integration process rigorously - one that will give accurate results at best, and avoid propagation of spurious errors at worst.

We will also explore the utility of saving the information of each event in a database warehouse of tracked issues. Benefits of such a system could be to produce trending reports on the stability of the system, identify which sources might be less reliable with their data feeds, etc. This system would give you the ability to manage your own business processes intelligently.

INTRODUCTION

This paper's purpose is to introduce the concept of detecting and persisting information about data-movement processes. It is not meant to be a 'how to' guide, but a consideration of ideas to accomplish the goal of actively monitoring these processes.

More generally, it discusses how to maintain information about a business process, and how to use that information for reporting. Kimball refers to this information as Process Execution Metadata, and we will discuss both Run Results and Exception Handling (Kimball 2004). The goals of such feedback can be many: improve the processes, clearly define SLA achievements, provide ROI metrics, etc.

More specifically, it discusses a technique of capturing exactly where in an ETL process a show-stopping error occurs. The data captured are details like which program was running, what type of situation existed, where that data came from, when it happened, etc.

The focus of this paper is on ETL processes, specifically using DI Studio. Thus some level of familiarity with DI Studio, database modeling, and SQL are useful to follow the technical content, but not necessary to understand the business value described.

BUSINESS CASE DESCRIBED

Assume a company is responsible for accepting data feeds from various clients, adds value to this data through their proprietary data manipulation logic, warehouses this data, and provides regular reports back to the clients. This scenario is common in the financial world (think brokers sending continuous trading data), in the retail world (think nationwide point-of-sale information being sent to corporate), and others.

As a consultancy, we at ThotWave® have recently seen more interest by businesses in knowing exactly what is happening in their internal processes, as well as identifying where problems occur as early as possible.

- What is a bad feed (switched columns, not all required files arrived on time, unknown files arriving)?
- Was it a situation for which the ETL code was not adequately tested – your basic bug (new dimensional data included, textual data casing, dates out of range, etc.)?
- Was it some cascading problem whose root cause occurred earlier in the process (everyone's seen SAS logs with tons of ERRORs in them where the only real ERROR was one macro var not defined, or some expected dataset wasn't created)?

The scenarios described here can be watched with the well-known Business Activity Monitoring (BAM) techniques. While BAM looks at the actual data, what we are watching here is more correctly termed Technical Activity

Monitoring (TAM, if you will). TAM means proactively inspecting each step in the process for expected conditions, and responding appropriately for each failure; specifically, stopping the process and alerting interested parties of the situation. To achieve this, we will also persist the event for reporting.

Let's discuss which kinds of reports may be useful. Think of this as a requirements gathering exercise. Knowing what we want to report will drive what data points need to be collected. Furthermore let's use the example scenario described above: multiple clients feed data daily with a known number of required files; those are scheduled to be automatically picked up and ran through the system, ultimately into a data warehouse after a final approval check has been successfully completed.

One report may be a trending report that shows the number of processing errors that happened, broken down by the client, with each client name being a drill-down into client-specific errors. The purpose of such a report may be to see which clients have provided the best data over some time span. To support such a report we need to include in each record at least the following data points: date when error happened, source client, type of error, and a textual description of the problem.

Another one may be a report that shows how long each job, or even larger sections of the ETL process, has taken, with the number records it processed. An immediate benefit of such metrics is the ability for interested parties to determine if these jobs or sections take too long. Historical trending on such data could tip off IT admins that the data is growing in such a way that the processes no longer adequate, and that some action needs to be taken.

By 'larger sections of the ETL process', think in terms of 'E' or the 'T' or the 'L' of the entire stream. There may be multiple jobs to support each logical section, and you want to know in which section each job is located. Kimball refers to these as batches and sub-batches(1). This report would need at least: how long each job/(sub)batch took, the name of each job/(sub)batch, and even the context, in terms of the source, in which the event occurred (Client A's Extract took this long, etc.).

We will look at reporting below, but it should be stated here that BPI reports should be looked at as any other business report. For example, these two reports are easy targets for inclusion in a dashboard view. Adding a stoplight icon gives a quick overview of how that day's process went. Turning on the red light would be an indication that at least error occurred, while green means that everything that was tested succeeded. If more information is desired, you should be able to drill down into the details that support these summaries.

TECHNICAL ARCHITECTURE

Now that the business case is defined, and we know the end goals, let's discuss some of the technical architecture of such a system. Let's also look at what DI Studio gives us natively for supporting this kind of system.

DATA MODEL

FIGURE 1 shows an ER diagram of the target data model.

[1,1]

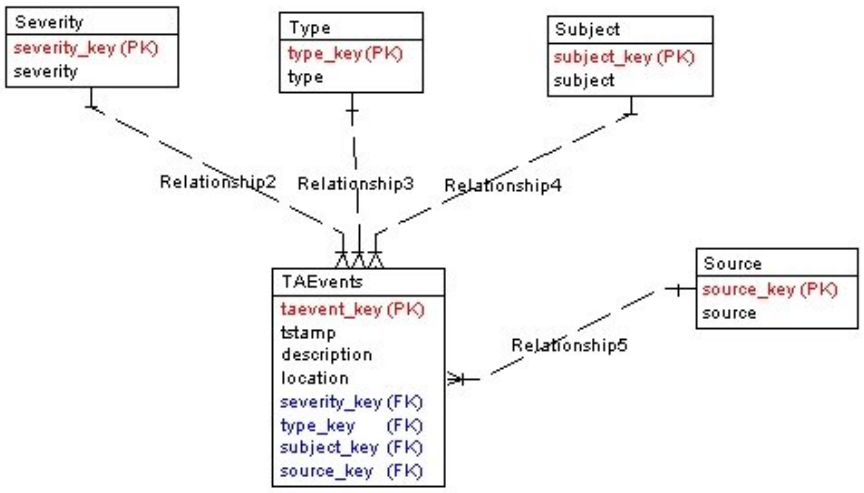

FIGURE 1

- SEVERITY: A dimension that describes how severe the tracked event is. Using the same types from SAS logs is reasonable: ERROR, WARNING, and NOTE.
- TYPE: A dimension that describes the type of event. In our ETL scenario, types would be MISSING_FILE (if an expected upload was never received), or NO_OBS (if a necessary dataset wasn't loaded thereby preventing the subsequent step from occurring).
- SUBJECT: A dimension describing where in the ETL process the event was trigged. Our values will reflect the batch-level value, 'Extract', 'Transform', or 'Load'. However, if sub-batches were defined, some hierchical construct may prove useful: 'Extract.Dimension', 'Extract.Fact', etc.
- SOURCE: A dimension that describes the originating source of the data. In our example, it would include client names.
- TAEVENTS: The fact table of Technical Activity Events being stored. It has foreign keys to the dimensional tables just mentioned, as well as its own analytical values.
    - TSTAMP: when the information was recorded
    - DESCRIPTION: textual explanation of what happened (*which* file was missing, *which* dataset was empty, etc.)
    - LOCATION: a low-level datum to track in which SAS program the error occurred

DI STUDIO (STATUS HANDLING AND CUSTOM TRANSFORMATIONS)

Process node of a DI Studio job in the Process Wizard has a 'Status Handling' tab in the Properties view. This gives simple support for doing desired actions for each of the exit codes for that step. You can also add this check onto the job itself by choosing the job's Properties view. The options include Send Email (you specify a recipient and the message), Send Entry to Text File (you specify the file and the message), Send Entry to Dataset (you specify `libref.dataset` and message), and two others (Custom and Send Event).

FIGURE 2 shows what options are available in the Status Handling tab.

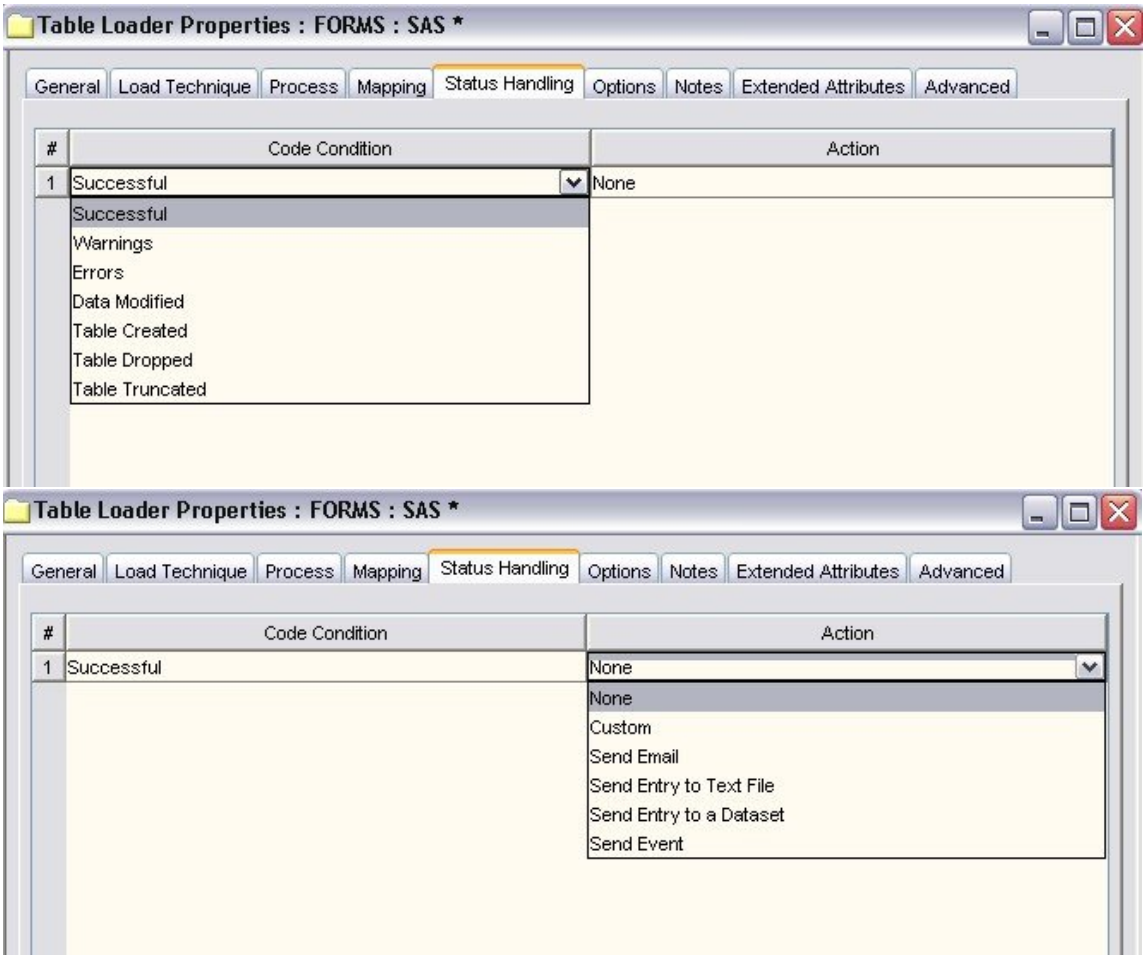

FIGURE 2

The 'Save Entry to a Dataset' has a predetermined set of values that it saves (Miller 2007):

- Label
- Job End Time
- Table Name
- Job Return Code
- Job Name
- Job Status
- Library
- No. Records After
- No. Records Before
- Job Start Time
- User ID

These suffice for pure TAM, but it is not the persistence or the notification we need to support true BPI. Enter Custom Transformations.

If you have worked with DI Studio, but not created your own transformations, you are missing out on a powerful feature. This allows you to centralize, share, and drop-in custom logic into your jobs. 'To centralize' means that they are registered as metadata and are correspondingly available. 'To share' means that anyone with DI Studio, and access to that metadata repository, can use it. Also, with the export and import feature, you can add them to other repositories easily. 'To drop-in' means that with the ease that you include pre-built transforms, you add in your home-grown ones. The resulting boxes in the process editor have the same look and feel as any others do.

To satisfy the BPI needs, you design your own library of transforms around an assertion library. This library of macro code will be much more granular than the Status Handling level of monitoring. DI Studio automatically includes macros from the `<config directory>Lev1/SASMain/SASEnvironment/SASMacros` directory in its autocall. So drop in your custom macros there, and they will be picked up. (Alternatively, you may edit the sasv9.cfg file and add your own folder to the autocall paths.)

One example of the need for using something other than the Status Handling feature is the Extract transform. It will report itself as successful even if the source tables are empty. One custom transform could assert that the source tables have observations. If not, an event with the table name, the job name, the process context, the timestamp, etc will all be inserted into our fact table. Another transform will inspect the data feeds and verify all required files are there; otherwise the client name, the missing file name, etc are inserted into the event data model.

This utility macro inserts a Technical Activity Event into the TAEvents table.

```
%macro insert_event(SEVERITY=,TYPE=,SUBJECT=,SOURCE=,
DESC=,LOCATION=,TSTAMP=%sysfunc(datetime(),8.));

proc sql;
    /* find the FKs from the dim tables */
    select severity_key into :SEVID
    from severity
    where severity="&SEVERITY";

    select type_key into :TYPEID
    from type
    where type="&TYPE";

    select subject_key into :SUBJID
    from subject
    where subject="&SUBJECT";

    select source_key into :SRCID
    from source
    where source="&SOURCE";

   select max(taevents_key)+1 into: TAKEY
   from taevents;

   %if &TAKEY = . %then %let TAKEY=0;

    /* insert the record into the fact table */
    insert into TAEvents
        (taevents_key,
         severity_key, type_key, subject_key, source_key,
         tstamp, description, location)
    values
        (&TAKEY,
         &SEVID, &TYPEID, &SUBJID, &SRCID,
```

```
          &TSTAMP, "&DESC", "&LOCATION");

    %if &SEVERITY=ERROR %then %do;
        FILENAME fileref EMAIL 'some@body.org'
SUBJECT="An error has occurred within &SUBJECT";

         %let TSTAMP_FMT=%sysfunc(putn(&TSTAMP,datetime16.));
        data _null_;
          put 'The following event has been recorded.\n\n';
                put "SOURCE: &SOURCE\n";
                put "TYPE: &TYPE\n";
                put "WHEN: &TSTAMP_FMT.\n";
                put "WHERE: &LOCATION\n";
                put "WHAT: &DESC";
        run;
    %end;
quit;
%mend insert_event;
```

Notice the extra notification logic embedded within. This logic is trivial in that it notifies a single recipient, and only for ERRORs. More complex logic could email interested parties based on some criteria: certain individuals get all ERROR events for a specific source, or maybe the ETL manager wants to be alerted when any WARNING or ERROR occurs. Nothing is stopping you from being creative with the active notification design to serve for your organization: attaching data, a 'short' message that is sent to known pager/cell-phone (SMS) email addresses, etc.

Another feature is to abort the program when the assertion fails - whenever a TA Event of ERROR severity is created. This will avoid those cascading ERRORs within a long program.

Here is the macro that checks if a specific dataset has any observations, and, if not, will call `%insert_event` with appropriate values.

```
%macro assert_obs (DS=,SUBJECT=,SOURCE=,LOCATION=,SEVERITY=ERROR);

 %let DSID = %sysfunc(open(&DS, I));
 %let DSOBS = %sysfunc(attrn(&DSID, NLOBS));
 %let DSRC = %sysfunc(close(&DSID));

 %if &DSOBS eq 0 %then %do;
     %insert_event(SEVERITY=&SEVERITY,
                   TYPE=NO_OBS,
                   SUBJECT=&SUBJECT,
                   SOURCE=&SOURCE,
                   DESC=Dataset &DS has no observations,
                   LOCATION=&LOCATION);
 %end;

%mend assert_obs;
```

Based off our own engagements, ThotWave Technologies® (http://www.thotwave.com) has created a freely downloadable, open source Framework for Unit Testing SAS (FUTS). This library of SAS code has several macros already defined for just these types of tests. To include any as metadata transformations is a simple task achieved through the Transform Generator wizard.

Note the use of &SYSLAST and &_OUTPUT in FIGURE 3 that makes this assertion immediately droppable between existing job steps.

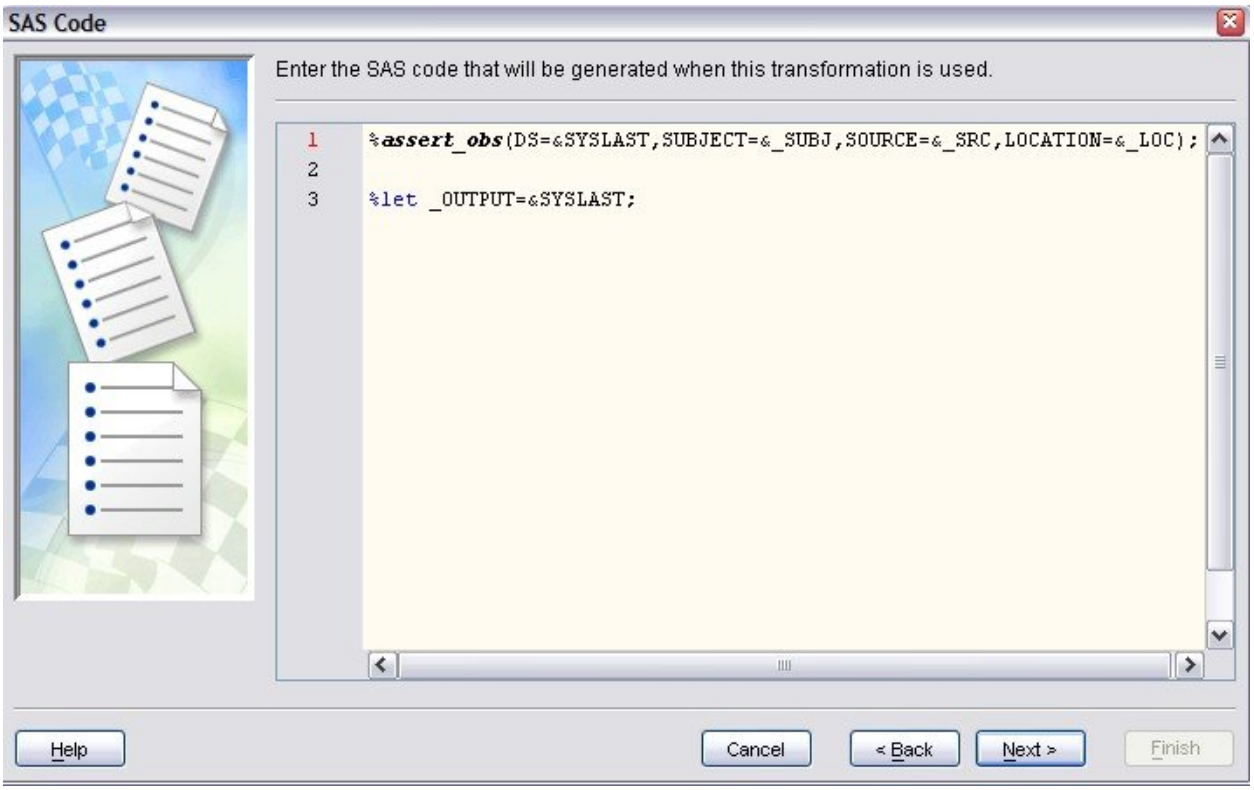

FIGURE 3

In FIGURE 4 we add in a static list of selectable choices. By adding a blank at the top, the user is forced to choose one, instead of relying on a default.

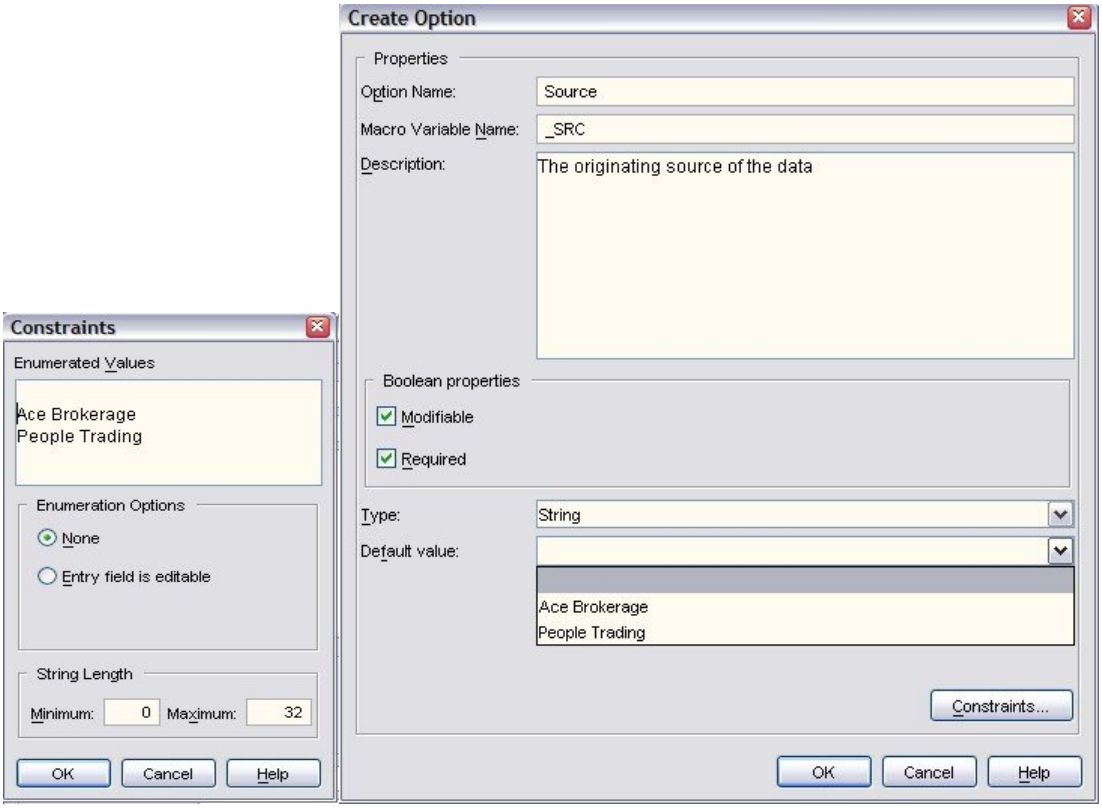

FIGURE 4

FIGURE 5 shows how to make the &SYSLAST and &_OUTPUT macro vars available to our transform.

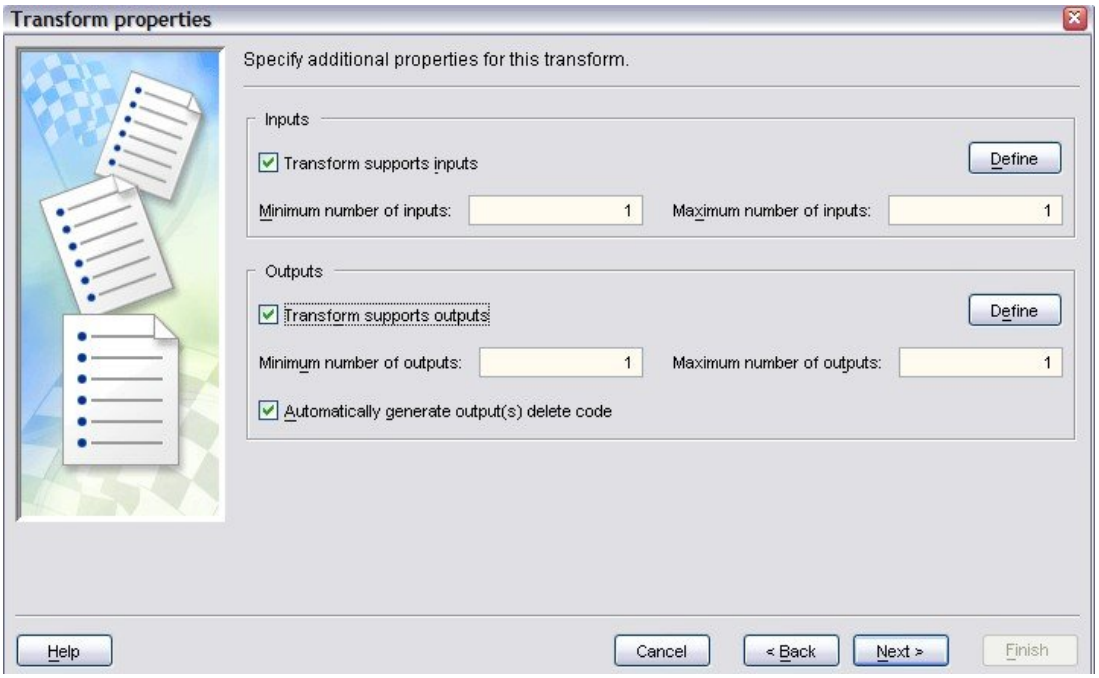

FIGURE 5

When dropped into the Process Wizard, the transform prompts for both previous and subsequent datasets (FIGURE 6).

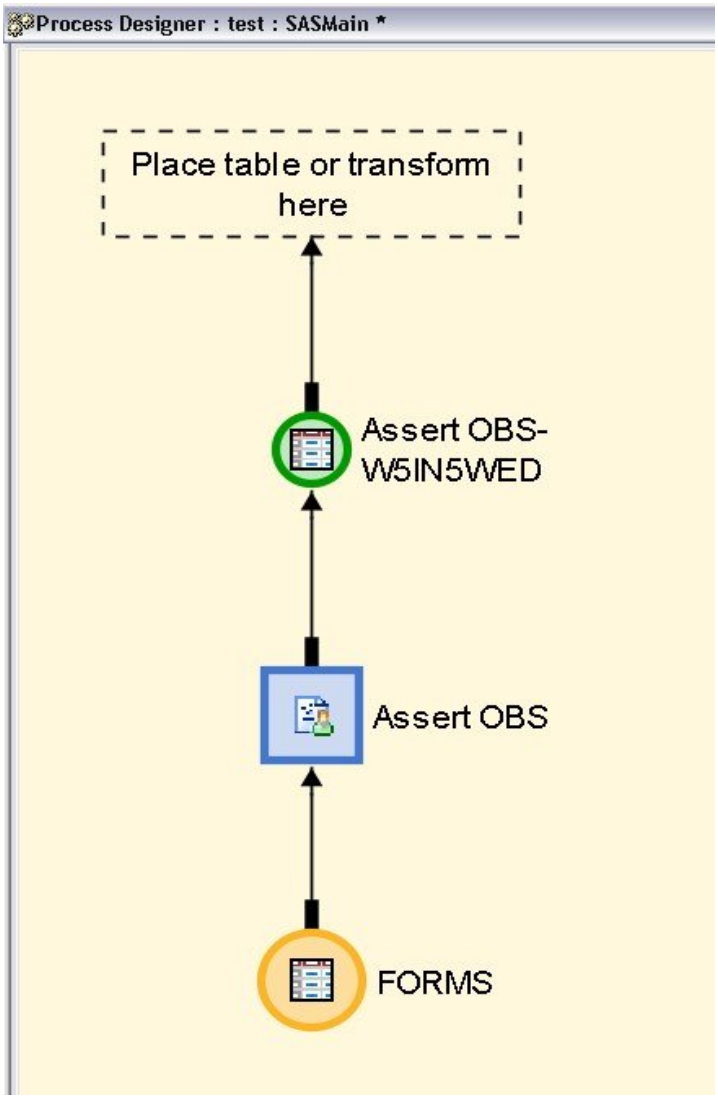

FIGURE 6

The resulting transform has your configured options available for the user, shown in FIGURE 7.

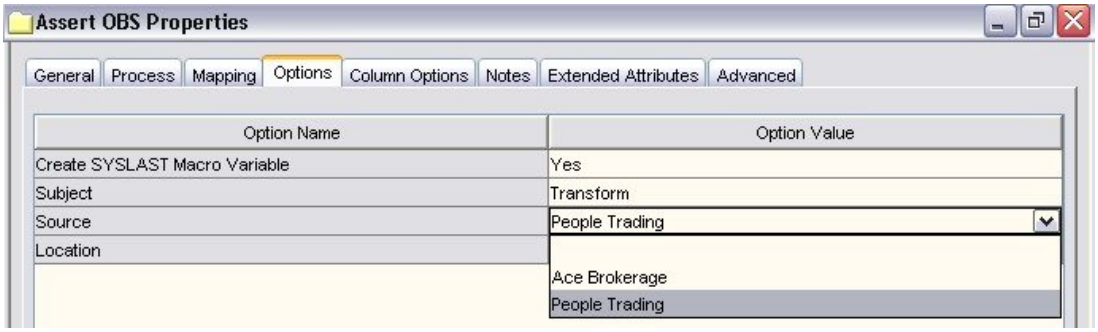

FIGURE 7

The generated code is shown in FIGURE 8.

```
60  /* Source table(s)/view(s)  */
61  %let _INPUT0 = foo.FORMS;
62
63  /* Target table/view(s)  */
64  %let _OUTPUT = work.W5IRRB4N;
65  %let  _OUTPUT0 = work.W5IRRB4N;
66
67  /* Delete target table(s)  */
68  proc datasets lib=work nolist nowarn memtype = (data view);
69     delete W5IRRB4N;
70  quit;
71
72
73  /* List of target columns to keep  */
74  %let keep = ;
75
76  %assert_obs(DS=&SYSLAST,SUBJECT=&_SUBJ,SOURCE=&_SRC,LOCATION=&_LOC);
77
78  %let _OUTPUT=&SYSLAST;
```

FIGURE 8

REPORT GENERATION

So far we have looked atto achieve some level of TAM in your SAS/DI Studio environment. The benefits of this effort are realized when the captured data is used as a driver for action.

As with any data mart, the data should be maintained for historical purposes (the trending reports earlier mentioned). Additionally, those reports that are deemed more time-sensitive can be pushed out to a distribution list. (We talked about attaching data to emails above.) You may always wish to have them available passively, also. Whatever dissemination method you choose will be your own decision. Treat it like any other project and do your due diligence for requirements.

Here are some simple reports created off the data model described above.

- FIGURE 9 shows how many TAEvents of type NO_OBS were created across time, broken out by batch subject.

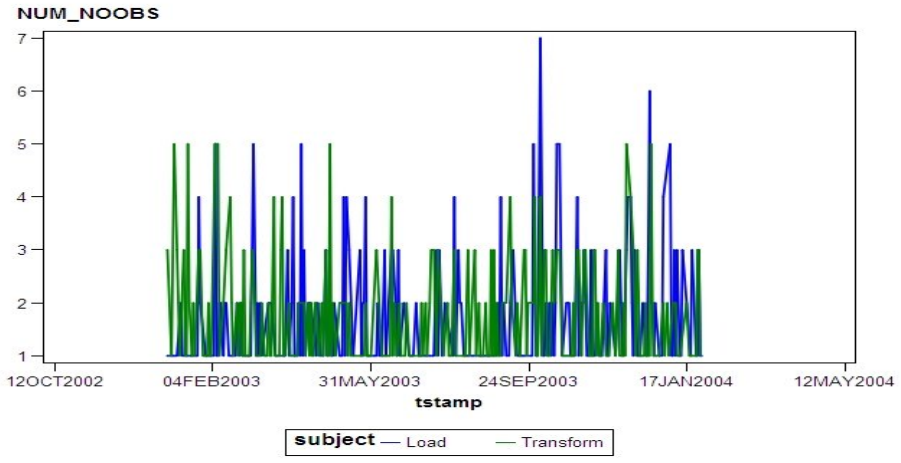

FIGURE 9

- FIGURE 10 show relative TAEvent frequency, first by severity broken out by source, and second by source broken out by severity.

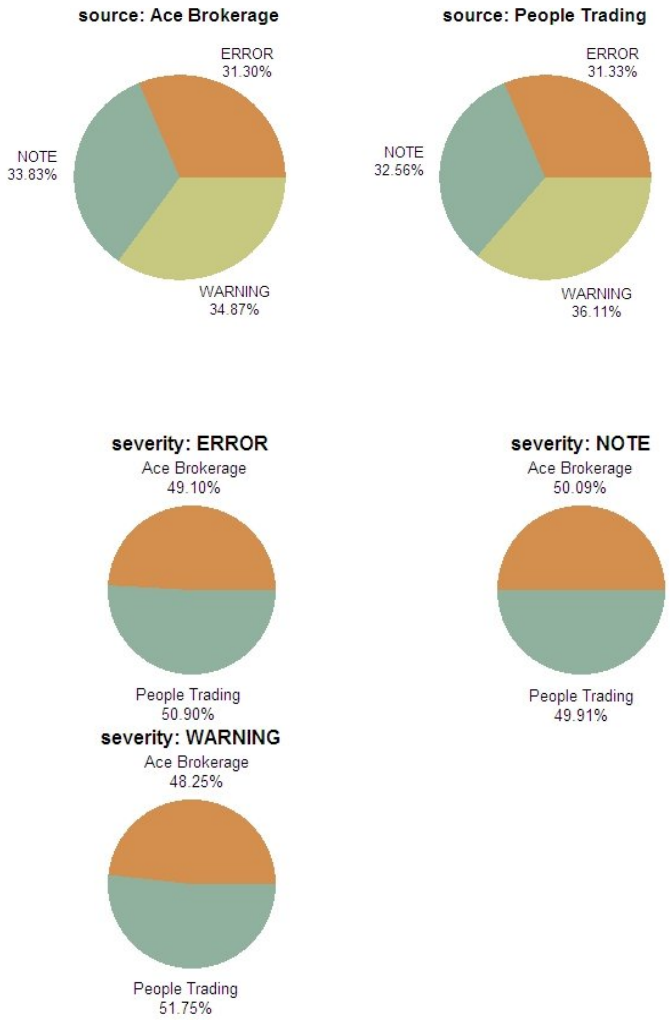

FIGURE 10

NON-TAM USES

We've been concentrating on the TAM aspect of BPI, but there's no reason that these same techniques cannot be used with BAM, and outside of the ETL world altogether and taken advantage of by more of your enterprise initiatives. Capturing this business-related metadata can give you the same insight as to what's being processed just as TAM gives you to how it's being processed.

Examples are:

1. Proactive inspection of data to identify potential fraud, as in credit card activity.
2. Verify incoming data meets expected analytical tolerances, as in regulatory compliance.
3. To monitor how users actually use your in-house applications - which reports to uses view most often, and are there any enhancements that could be made to serve their needs better?

Opening up the BPI warehouse to more sources does present its own set of issues though.

1. Our simple logic expects that all incoming types, sources, and subjects already be enumerated in the reference tables. You may need to allow arbitrary values to come through, and to handle them gracefully.

2. The more users to be notified, the more need there is to allow the users to configure which events they need to receive.

CONCLUSION

In order to achieve proactive responses to enterprise-wide processes, it is necessary to know as soon as possible when actionable situations occur. This paper has shown some ways to achieve such success, primarily within ETL data movement.

The scenarios described above are meant for illustration purposes, and may trigger thoughts of what quality support can be included in your own business processes. The end goal is to know your processes; not just when something goes wrong (which is critical, and does form the basis of this paper), but also how things are improving. Knowing your processes at this level is true Business Process Intelligence.

REFERENCES

1) Kimball, Ralph and Conserta, Joe. 2004. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons,.
2) Mike Miller. 2007. *SAS DI: Introduction to User Written Transforms and Status Handling*. SESUG 2007.

ACKNOWLEDGMENTS

Richard Phillips: for providing assistance with the SAS code and generated charts.
Jeff Wright: for providing direction and general support.

RECOMMENDED READING

Kimball, Ralph and Conserta, Joe. 2004 The *Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. John Wiley & Sons.

Greg Nelson, Danny Grasse, Jeff Wright. *Automated Testing and Real-time Event Management: An Enterprise Notification*. http://www.thotwave.com/Document/papers/SUGI_04/Automated_Testing/228_29_eventmngmt.pdf.

Jeff Wright. *Drawkcab Gnimmargorp: Test-Driven Development with FUTS*.
http://www.thotwave.com/Document/papers/SUGI_06/FUTS/BackwardProgramming.pdf

Mike Miller. 2007. *SAS DI: Introduction to User Written Transforms and Status Handling*. SESUG 2007.

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:

Danny Grasse
ThotWave Technologies
510 Meadowmont Village Circle #192
Chapel Hill, NC 27517
Work Phone: (919) 649-9066
Fax: (800) 584-2819
E-mail: dgrasse@thotwave.com
Web: www.thotwave.com