WAREHOUSE NOW!
BLASTING THROUGH TO LEGACY DATA WITH (ALMOST) NO PROGRAMMING

JAY A. JAFFE, PhD

Most data warehouse vendors sell lock-stock-and-proprietary systems requiring a centralized IS (Information Services) conversion commitment of much time and many dollars. But with the help of the SAS System, you can give your users a line to decision-support information without involving IS, through whatever data sources you've been creative enough to get at for your present SAS reporting.

Your "poor person's warehouse" allows you to use only Base SAS skills to easily create specialized reports with on-user-demand variability within a snappy user interface. It also provides a set slice'n'dice reporting templates for general use, to which set you may add your own. The user needs know nothing about SAS software or data structures, and the programmer doesn't have to do any AF programming at all to specify the user interface automatically generated for each report.

On this foundation may be built an "as-if" data warehouse, that allows the users direct access to legacy corporate data stores as mapped by the knowledgeable programmer, who in turn is freed for more complex and challenging tasks as the user breaks through to to the legacy on his own.

INTRODUCTION

Coming to Terms

Most often, we see "data warehouse" defined in terms of database: its structure, volatility), granularity, time-variance. We are told data warehouse is indeed a database, but one physically separate from operational data stores and organized for what we call decision support.

But it is only that last part (decision support), that accounts for why talking "data warehouse" generates so much hoopla outside purely technical circles. Data warehouse as a "better" MIS: that is why all the fanfare.

As SAS® users and programmers, we are used to being in the critical path to decision support. We have learned to find the data we need, from wherever it comes, and massage and present it to meet the sundry purposes of management information. We know its limits and its quirks, and try to step around these to get the correct answers. Few of us have the good fortune to work where a conscious, corporate-wide system of data organization, much less a full-featured warehouse, SAS-based or otherwise, has actually been implemented. No, we expect the Legacy and its effluence: all kinds of data, from all kinds of places, aggregated by operational or audit considerations rather than purpose design, and much of it useless to us.

You will see other SUGI presentations about using SAS® software to construct warehouse data stores. But here, let's discover uses for the SAS System as a vehicle for providing warehouse services, whatever and wherever the source data may be.

It's not that building data warehouse according to the structural definitions is wrong — quite the contrary! — it's just not what we are about in this paper. We will care less about how the shelves are stocked than how we can get product off them, and define:

data warehouse: a system that makes data appear to users as
if integrated into purposefully-oriented collections, and which helps turn these data into usable information upon user request.

Not necessarily integrated, but as if integrated. And responsive to information-hungry users.

Let our present mission be, Make it so.

Coming to Grips

What I will in reader-friendly fashion call "our" system is only one of many possible approaches one might take to this mission, but it is a pretty good one if I may say so. There is no way in these few pages to describe this complex and multi-leveled project in detail; some components will even remain completely untouched.

This paper is an attempt to offer the "big picture" in broad strokes, and the selections of detail (meant not as the tip of the iceberg, but as interesting chunks from here and from there) are meant to illustrate only some of the important conceptual and technical principles of the enterprise.

Our effort here unfolds in three evolutionary stages. At Stage 1, we give the SAS programmer a way of turning over report production directly to users, by allowing them to code parameterized programs such that "non-technical" users can specify choices in a user interface (UI) whose implementation need not concern the programmer at all. The programmer writes code that sets up the report, "parameterizing" it for user variation with the help of utility macros provided by the system, and constructs a brief, simple report class definition that is used by the system to generate a virtual SAS/AF® user screen for the report. Per well-thought-out report class, the user can generate many report variations without bugging the programmer for many "ad-hoc". The programmer can turn her attention to more complex or one-shot reporting tasks, and to environment and coding additional report clauses.

This set of capabilities — parameterized reporting via an automatic user interface — stands alone as enough for many practical purposes. I devote the lion's share of this paper to understanding the Stage 1 system not only for that, but because it is our solid ground on which to build data warehouse. On this ground, we erect the foundation: Stage 2, a generalized way to let users play with reporting at the variables level.

Stage 2 introduces template reporting, and this, too, is worthwhile on its own merits. A Stage 2 report, in a vague analogy to SAS/Assist® software, provides a "canned" parameter set to which SAS datasets can be linked as input and methods for turning these into particular output forms. The programmer writes the code to set up the input datasets based on user choices, but the user may have a wide variety of choices thereafter for...
what he can do with it, using variables as pick-and-choose elements for, say, a PRINT, a FREQ, a MEANS, even a REPORT. A single input dataset built by one report class code entry may call on many templates, leveraging the programmer's efforts that much more than did the Stage 1 system; she may even expand the system with additional templates, as these are coded very similarly to parametric reports.

At this point, all that remains to achieve data warehouse is to tackle the corporate data by mapping it for the user as if it were purposefully organized, at an elemental level, for decision support. The program code no longer constructs input datasets for templates; the user does. The Stage 3 system enables this by knowing (through its metadata) what the important elements are and how they may be combined for reporting, and where they come from. After that, the Stage 2 foundation on the Stage 1 ground provides all that is needed to let the user feel the power of transforming these data into meaningful reports. As for the programmer, she may now rise to a new challenge: using her knowledge true corporate data to provide the data mappings to business elements, and building supporting methods as needed. She may never get completely free from special-purpose reporting — after all, there is only so far any generalized reporting system can go — but day-to-day will never be the same. And, she may add to or expand business-valueable warehouse subject areas as only an expert in both the data and decision support can do.

Realizing this system at any of its Stages requires no Information Technology (IT) commitment whatsoever. With the help of the SAS System it is entirely possible to develop and release at the departmental level. It can join in to build the system on behalf of the departments, or it can go its own way and build a true, storage-defined data warehouse which we may lock forward to our system's coexisting with, adapting to, even obsoleting below into graceful retirement. Or, IT can choose to do nothing, or grudgingly want to do something but not get the approval (i.e., budget) for a data warehousing initiative.

But that's not our problem, is it? We will Warehouse NOW!

1. USER-DRIVEN REPORTING

The system as originally developed was, with some differences, was more or less what we are calling Stage 1 of our mission. Built to run under MVS/TSO SAS 6.08 for the Kaiser-Permanente Medical Care Program's Northern California region, its purpose was to help support decisions need-driven staffing and scheduling of outpatient healthcare providers based on client's needs, with reports about topics such as appointment patterns, scheduling activities, specialty referrals patterns, etc.

Because support was needed for character-based as well as vector graphic terminals, I decided to eschew Frame, except to display SAS/GRAPH output when necessary, and use Program entries for the SAS/AF user displays. Conversion to Frame -- or other UI schemes -- is a problem that can be faced on its own, without any change to background processing.

USER VIEW

Menus; General Screen Operations

The first AF screen the user sees (after any SPF panel we might use at front-end) is the menu shown in Display 1. In our user interface, screens in which the dominant action is a single choice to be made among fixed alternatives, of which a menu is a prime example, show choice codes (which may but need not be numeric sequence numbers) associated with each prompt label, and a field in which the user can enter the choice.

Providing several methods of accomplishing something on the screen never hurts. People are different, and supporting their differing habits helps garner votes for "user-friendly"-ness. For menus, our system allows for placing the cursor on the selection line to take the place of typing in the choice code. In both cases, the ENTER press triggers the selection, but cursor placement is particularly attractive where a pointing device may be used, as is becoming more and more the case as personal computers running terminal emulators replace "green screen" terminals.

Also to accommodate user preference, and because its implementation is but a simple variant of a necessary method for other screens, the TAB key may be used to move the cursor to a choice. (By use of a blank field next to the choice code, both code and label can be protected, avoiding the confusion often caused by the use of unprotected fields merely for TAB or ENTER.)

Command Actions

The layout of most system screens has both a menu bar (in our SAS System system, the first-level of a Pmenu) and a function key prompt line. We use the menubar typically for choices that pull down other subchoices, to which Pmenus are just the ticket (and which also yields a system message line without offer of a Command prompt). Function keys we tend to use for choices that have immediate action, the key prompts can be used by cursor-to-then-ENTER and have the same effect as a function key press.

Help on the Way

Context-sensitive help is implemented by placing the cursor on a field and then pressing F1: general help for the screens or system as a whole is obtained by placing the cursor anywhere else but a choice, or via the "Help" Pmenu item.

Screen Scrolling

After initialization, the system "knows" the size in lines of the user's display, and can decide when to allow screen scrolling based on the number of choices that require display (the same method is used in other system screens when there is more than can fit on the display at one time). When scrolling is enabled, the Fkeys prompts F7=Up and/or F8=Down appear.

Application Specifics

For the Stage 1 production, programmers have written a number of reports distributable into topic areas. The "Categories" menu in Display 1, which we use as the main menu, shows the major categories. Though the system can handle an arbitrary number of menu levels, there are generally only two or three in this application: main, perhaps a subcategory menu (not shown; virtually identical to the main menus in appearance), and a menu of report classes such as the example in Display 2. A quick-cut to the desired report screen is provided on each menu screen, for the user that already knows which one he wants; this is especially appreciated in sluggish mainframe-terminal sessions where each user interrupt can take annoyingly long.

Parameter Entry

Let's pick an example: Report REFO3. The purpose of this report is to examine patterns of referrals. Listing 1 shows a page of output from an instance of REFO3 so you can get the context.

Display 3 shows the "Choices" screen that appears when this report class is selected. "Options" are more or less standard. The normal default for Print is on, and Download and Graph off. If a report class does not have one of these even as an option (e.g., if there is no SAS/GRAPH potential, or a date range is not needed) it simply does not show up on the screen at all.

The remainder of the choices are entered on each line. There are two standard forms for each input line. In one of these, a prompt is associated with a short (1-char) field that takes one of a small set of values, as noted for user convenience in the remainder of the display line. The other standard form is for users to enter values directly. Utility fields to the left of each input line alert the user to something or other, for example, if an entry is required.
Besides specifying which choices will be available for the report, with respect to each one the report class designer or programmer may specify it to be optional or required, and (for free entry parameters) how many values are acceptable. She may also stipulate default values, and even show an unmodifiable value default that was preselected by the system but which, for one or another reason, the user is not allowed to change.

If multiple values are allowed, the user enters these separated by commas. If there's not enough room on the line, he enter terminal commas and lines to use (we'll see this later).

The user can, for convenience, save and retrieve parameter sets, either as most-recent-for-this-report, or as a named set.

Selection Lists

Lists of values may be defined for the user's convenience. By placing the cursor on one of the free-form entry fields and pressing the context-sensitive List key (F2), the user may bring up a list if one is available (if not, a message is given). Display 4 shows an example; the user has selected Status, which in this report means the status of the referral (made, booked, seen, etc.). For lengthy lists, the user can limit the display as noted in the screen's explanatory text. Depending on the background “nesting” relationships of the parameters, the system can be set up with pre-limits dependent on values of other items. In the present example, a list of Departments would show only those that exist at the chosen Facility(ies); all may have a department of Medicine, but not all will here.

The user makes selections by placing an “X” (actually, any non-blank) next to the item, for more preferable, especially on the mainframe, than using an ENTER-toggle per item; the user can just TAB along entering (or blanking out) items to satisfaction. At any rate, when he returns to the parameter-entry screen, the codes for these items automatically transfer there. This communication is two-way: If values are manually entered (or removed) and then F2 is pushed, appropriate Xs are initialized on the picklist display.

Inverse Selection

The F9 key can be used either from a selection list, or context-sensitive from a free parameter input line, to transform selections to exclusions. In our example, suppose the user wishes to select all but two or three of the facilities, or the departments, etc. Instead of typing all but two or three codes, or selecting all but two or three from the picklist, he selects or removes only two or three, presses F9, and the system understands he wants all but those, i.e., that all are to be selected except for the ones entered.

F9 is a toggle (i.e., for selection to exclusion and vice-versa). The user knows what is currently active by an “E” (for Exclude) in the utility field (or nothing, if a straight selection); on the picklist, the screen title and information prompt change wording to refer to Exclusions instead of Selections. If F9 is not applicable to a certain parameter, a message so informs the user.

Error Checks

When the user attempts to run a job, the system first looks over the set of parameters chosen and determines for each if there are errors. Errors may be encountered in the number of values entered, or in the content of one or more values.

Jobs and Results

All reports can run in the background (more precisely, in Batch via jobstream TSO SUBMITs). A reports designer can specify per report class whether reports can also run foreground, i.e. as OLAP within the DMS session; this should be reserved, of course, for jobs that do not place heavy demands on data processing resources. Results of printable background reports are placed in external files so the system can manage them without the user's dealing with 10F, SDSF, SAR etc., although user can by preference alter this scheme. When managed by use system, results can be viewed, printed, downloaded, copied etc. from a single LIFO screen display, as shown in Display 6.

Additional Features

The system allows the user to set gross output preferences such as titles and footnotes, page numbering and dating, etc. A report design can override the ability of the user to set any of these; in the case of titles and footnotes, a report may take control of one or more of the 10 available lines but leave the rest to the user.

The system provides a few peripheral niceties for the user. He can enter various display preferences, “shall out” to ZSF without terminating SAS (yes you can, since 6.08), etc. These ancillary capabilities go beyond the scope of this presentation.

Access and Audit

Also beyond scope here, except for brief mention, are several system capabilities not apparent to the user, the most important of which is system access usage auditing.

Access Control

Access control is implemented in two pieces. First, of course, we set up appropriate securities (RACF, ACF2, TSS etc.) with IT such that users can read and write necessary files. Once this is established, we work within it to take complete control of more granular access decisions, which can be implemented at the category, report, and parameter levels (and at Stage 3, at the subject and element levels). Each of these objects is assigned to one or more logical access groups, and each user logonID is assigned a set of permissions associated with various access groups. Access permissions can be intersected, such that more than one can apply at the same time.

Auditing

The system writes few files designed expressly for audit, because we can use byproducts of user processing. For example, we keep a list of reports output for display to the user; we can use that to study report-request activity. We can also poll the system catalog for evidence of new user files, and read these as necessary by audit programs. Since we will also keep (for troubleshooting purposes) a running SAS log for each user, we can write specific notes thereto and read the log back in for detail audits.

REPORTS PROGRAMMING

The reports designer must consider what sort of variability is appropriate to this or that report class; often, it is best to do this in consultation with user representatives. This variability will be expressed in the source code by use of utility macros provided by the system.4 The programmer need not know any SAS/AF, but must be comfortable at least to an intermediate degree with macros and macro variables.

Listing 2(a) shows a selection from the source code for the example SARUS report. Listing 2(b) shows the code for a couple of the utility macros that it uses.

We thus see how easy it is to write reports code to accept varying parameters. But we have begged some questions: How and where does ISEI get the correct parameters? How do these get transmitted from UI selection? And How does the UI know what choices are to be displayed in the first place?

DESIGN HIGHLIGHTS

To construct answers to these critical questions, we begin by partitioning our design into two fully-separable subsystems that communicate only by data, as depicted in Figure 1. By separating user interface from background processes, we will realize greatest productivity: UI can be developed and tested on a completely independent track from background operations. In addition, this architecture strategically positions us to take best advantage of
raw text
While the lay of the output is undefined in the case of special (non-template) reporting, most templates are constrained to a particular set of output forms. There’s only a few ways this or that printing procedure shows its stuff. Accordingly, we have two ways to turn in developing a template reports user interface. We can stay with our generic “Choices” interface, or we can develop a set of displays that, like SAS/Assist, show the user where stuff will go while asking him just what stuff should go there.

There is something to be said for both these approaches. Here, we opt for the simplicity of the zero-programming UI, and treat everything as a parameter. After all, we can always display layout forms in a Help screen, and the simpler UI buys us much more than some convenience in delivering the system: we can continue to assume the reports programmer doesn’t want to do SAS/AF, and yet we can easily allow her to expand the system with new templates (you guessed it; Templates will be specification within a simple event store).

We do, however, need to modify the user presentation a bit to allow for template reporting. The system must accommodate the facts that one report can be tagged to many templates, and that one template can be used to many reports. We accomplish this by a two-tiered parameter display. Displays 6 and 7 show what this means. The report class represented in Display 6 allows for FREQ and TABULATE, and a couple of other, specially designed templates (these go beyond our scope here). These are gathered under the “Output” section, where instead of a simple “Printout” choice, the template choices are named. For each of these choices to which the user answers Yes, we present a screen such as Display 7. Notice how the choices here correspond highly to the choices SAS/Assist would present, though in our simpler, general form. A selection list called by F2 presents the variables from a virtual input dataset (it hasn’t been created yet, but the report class knows what it contains) that may be used for the template parameter.

**REPORTS PROGRAMMING**

If you’ve used SAS/Assist, you will have observed that in most cases, almost exclusively so, Assistant objects ask the user to select an input SAS dataset (they call it the “Active” dataset). There are other parameters of course, but only after dataset selection can Assistant go and do its thing. Template reporting is the same, basically. To take advantage of templates, all the report code need have the code do construct one or more potential input dataset, and then apply template macros per user selections. Actually, the report code is responsible only for the input dataset(s); a macro somewhat analogous to USEL seeks out any templates that have been selected by the user and arranges the parameter values so they will be passed to the correct templates.

It is, of course, imperative that dataset constructed for template reporting contain such variables and otherwise be in such form that the templates will work with it. But although the programmer/designer must have an idea of what templates might be appropriate to the purpose of the report, she need not identify precisely which these are. Instead, she considers the set of parameters they entail, and ensures the input dataset will support these.

To program a template is to construct its driver macro, and write its template class definition. The latter is virtually the same as a report class definition file as shown earlier, and is actually simpler in most cases. The names, requirements, and numbers allowed of the selectable parameters are coded the same way. Templates, just like special reports, may and often do share common parameters.

**Template Specialization**

Templates need not be direct transformations of a SAS procedure. As mentioned earlier, one can write DATA _NULL_ templates. It is occasionally useful to define a template over a couple of SAS job steps, for example, a PROG:SUMMARY with subsequent printed reporting based on the SUMMARY output.

**DESIGN HIGHLIGHTS**

Our Stage 1 system does, of course, allow for several forms of output per report: in our example the “print/download” choice does exactly that. But this choice was an end point: It is left entirely to the report code to decide what to do with it. We can surely imagine a report with more than one printing choice. For example, report code might be written to produce several different printouts, the user being allowed to choose which of these will indeed be produced by a particular instance. But the report code still has to deal with the consequences.

Once again, our architectural solution is to allow components to communicate via events. In this case, the communication is between the virtual input dataset variables and various parameters that are used by reports templates.

**Report Class: MAP section**

In the case of templates, the report passes parameters to code that is not its own and over which it takes no control. This turns out to be much of a problem, so long as we take a careful assessment of the input variables that may be used for this or that parameter. We will write a MAP section, that declares for the UI the input dataset(s) the report code will construct and their variables, provides a prompt for each, and then goes on to provide the variable-to-parameters mapping. Consider a dataset that contains five variables, as follows:

- NAME Respondent Name
- AGE Age at Interview
- SEX Sex
- INCOME Family Income 1995
- SIZE Family Size (living in household)

In this simplified example, let’s limit ourselves to one input dataset, and our universe of parameters to only three: VARI (variables list, e.g. for a VAR statement), STAT (may be used for a statistic, e.g. in a PROG:MEANS template), CLASS (appropriate for a classification variable, e.g. in a CLASS statement). Figure 2, then, illustrates a mapping analysis.

Note that NAME is not to be mapped; this means “Detail Only: Do not aggregate”. In other reports, NAME might very well be a CLASS variable. Remember that the mapping of variables to parameters is decided by analysis of the the report’s input data and its intended purposes. We use the result of this analysis to construct the MAP section for this input datasets; see Listing 8.

**3. DATA WAREHOUSE**

So far, our system isn’t purely self-service with respect to corporate data. True, the user has event-driven reporting at command, but it is up to an intermediary developer to identify the report data and implement the logic in code. Even with template reporting, the input datasets must still be explicitly created or referenced by the report code. We have not yet given the user choices for template reporting that do not require a programmer to create template input from the Legacy.

We have not, in other words, given users direct access to the corporate data at the variable level, subject-organized. This we now do, and when we are done our system will be very much the data warehouse indeed. The challenges of Stage 3 are to:

1. Build metadata about legacy sources, that will yield answers to the question: What are these business data, anyway (or stated more bluntly, Why do we care about it)?
2. How may the data be used in template parameters; How can reports get at it (where it lives, processing hoops to
User View

There's nothing remarkable in going from the Stage 2 system to data warehouse, at least as far as template reporting goes. Instead of variables mapping to templates, we will map elements (defined below) to templates, under subject headings. Elements and subjects are discussed below, under “Metadata.”

Ad-Hoc Query

In order for the warehouse to be even more complete, a query capability may augment template reporting at Stage 3. Similar to query windows found in other software (including later versions of SAS/Assist), this provides the user with an alternative approach to constructing reports. Details are beyond scope here, and anyway as SUGI approaches I'm still waiting to decide for myself.

What are these business data, anyway?

Don't you like that juxtaposition? Let's extrapolate to something more entertaining, to gather to mind all the various sources you have access to. Great care must be taken, however, with each subject.

Advisors - “data about data”. When refined into a data store, sometimes called “The Data Dictionary”, but I wish to leave aside that construction brief. This provides the user with an alternative approach to constructing reports. Details are beyond scope here, and anyway as SUGI approaches I'm still waiting to decide for myself.

METADATA

Metadata — “data about data”. When refined into a data store, sometimes called “The Data Dictionary”, but I wish to leave aside that construction brief. This provides the user with an alternative approach to constructing reports. Details are beyond scope here, and anyway as SUGI approaches I'm still waiting to decide for myself.

What It Is

The good news is that you don't have to build the data warehouse all at once. You can start with a small part of a single subject area, then add to it, and add new business subjects, at a pace that fits your need. Great care must be taken, however, with each subject. What are these business data, anyway? Why do we - or rather, our decision-making users - care about it?

This is sort of a trick question, because decision support doesn't care about the data per se, but about what it means. In our Stage 1 and 2 systems, we were interested in variables, i.e. in the “data itself”. This is, after all, what we use when writing reports (or template) code. As SAS programmers, we make the implicit leap from what it is to what it means, i.e. we translate the analytic requirements for meaningful information to sources and fields, and then datasets and variables. But now that we will let users have at it by self-service warehousing, we cannot do this; the relation must be made explicit. The user doesn't know, much less care, about fields and variables. He cares only about the meaning of the data. My American Heritage dictionary (the best) defines "subject":

"n. ... 3. A course or area of study. 4. A basis for action; cause." Don't you like that juxtaposition? Let's extrapolate to something we can deal with in the data warehouse:

"element: an item of quantifiable or classifiable fact of direct or indirect potential use in decision support reporting; subject: a named conceptual grouping of elements that seem frequently, where business decisions are concerned, to be thought about in combinations.

There are two approaches we can take from sources to subjects, and from subjects to sources. We will often find ourselves taking both approaches as we build, over time, the ad-hoc warehouse.

Sources to Subjects

When building the data warehouse, it is often convenient, even entertaining, to gather to mind all the various sources you have used in the past to produce all those ad-hoc, or "what if" systems. Which of these do you use on frequent occasions? These may be good candidates with which to start. First, separate the wheat from the chaff. Let's take an example: the "Registrations" database (DB2) used in many of the reports of our Stage 1 example that study medical office visits (registrations). Some of the fields in one of its tables:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPT_ACCESS_NO</td>
<td>Appointment access number</td>
</tr>
<tr>
<td>APPT_TP</td>
<td>Appointment type</td>
</tr>
<tr>
<td>BILLING_EXCP_TP_CD</td>
<td>Billing exception code</td>
</tr>
<tr>
<td>BILLING_METHOD_CODE</td>
<td>Billing method code</td>
</tr>
<tr>
<td>CLASS_ATTEND_CD</td>
<td>Class attend code</td>
</tr>
<tr>
<td>COVERAGE_CD</td>
<td>Coverage code</td>
</tr>
<tr>
<td>DATA_CLASS_CD</td>
<td>Data class code</td>
</tr>
<tr>
<td>DATA_SOURCE_CD</td>
<td>Data source code</td>
</tr>
<tr>
<td>DRB_COV_IND</td>
<td>DRB coverage indicator</td>
</tr>
<tr>
<td>DEPT_CD</td>
<td>Department code</td>
</tr>
<tr>
<td>DEPT_SUG_AREA</td>
<td>Department sug area</td>
</tr>
<tr>
<td>FAC_ID</td>
<td>Facility ID</td>
</tr>
<tr>
<td>GROUP_ID</td>
<td>Group ID</td>
</tr>
<tr>
<td>HP_EXCP_CD</td>
<td>High pressure exception code</td>
</tr>
<tr>
<td>MEMB_IND</td>
<td>Member indicator</td>
</tr>
<tr>
<td>NVMMachine</td>
<td>NVM machine</td>
</tr>
<tr>
<td>PATIENT_ID</td>
<td>Patient ID</td>
</tr>
<tr>
<td>PATIENT_FIRST_NAME</td>
<td>Patient first name</td>
</tr>
<tr>
<td>PATIENT_LAST_NAME</td>
<td>Patient last name</td>
</tr>
<tr>
<td>PATIENT_MIDDLE_NAME</td>
<td>Patient middle name</td>
</tr>
<tr>
<td>PATIENT_BIRTH_DATE</td>
<td>Patient birth date</td>
</tr>
<tr>
<td>PATIENT_GENDER</td>
<td>Patient gender</td>
</tr>
<tr>
<td>PATIENT_RACE</td>
<td>Patient race</td>
</tr>
<tr>
<td>PATIENT_ADDRESS</td>
<td>Patient address</td>
</tr>
<tr>
<td>PATIENT_CITY</td>
<td>Patient city</td>
</tr>
<tr>
<td>PATIENT_ZIP</td>
<td>Patient zip code</td>
</tr>
<tr>
<td>PATIENT_PHONE</td>
<td>Patient phone number</td>
</tr>
<tr>
<td>PATIENT_EMAIL</td>
<td>Patient email</td>
</tr>
<tr>
<td>PATIENT_MOBILE</td>
<td>Patient mobile number</td>
</tr>
<tr>
<td>PATIENT_FAX</td>
<td>Patient fax number</td>
</tr>
<tr>
<td>PATIENT_LINE_TYPE</td>
<td>Patient line type</td>
</tr>
<tr>
<td>PATIENT_LINE_NUM</td>
<td>Patient line number</td>
</tr>
<tr>
<td>PATIENT_LINE_NAME</td>
<td>Patient line name</td>
</tr>
</tbody>
</table>

I might care about terms of insurance coverage at a later date, but not now. I might forget COVERAGE_CD, DRB_COV_IND, MEMB_IND, etc. I don't even know what some stuff is (e.g. KJ_OBJ_IND). I do know that I often use MRN (the patient's unique ID number), FAC_ID (the facility - medical office location - where the visit took place), APPT_TP (what kind of visit it was, i.e. what it was for), MONTH_LACE (days of patient visit), etc.

It is this latter set - the stuff that I do use - that becomes my concern for warehousing: If I use it, especially if I use it frequently, well that means that the users I support likely will want to find it when the warehouse opens its doors.

Some of these items are directly meaningful to the user, though they could stand a little re-naming: FAC_ID to "Facility", APPT_TP to "Appointment Type." This means that there's nothing to do with them: the elements correspond directly to the data. Some could stand a bit of transformation: Does anybody care about age months? Let's FLOOR(MONTH_AGE/12) and create an AGE element in common language. (Actually, pediatricians might care about age in months, so we might have two elements from one field; no reason not.)

The point is, we can look at the various files available to us, and certainly find many user-interesting elements. We are not constrained to consider these all one subject area. If I do ever become interested in the insurance angle, I can get back to the coverage codes and indicators, and group them into a separate subject area. More importantly, we are not constrained to create a subject area from one data source. If I have other files of whatever storage type - that contain information pertinent to office visit registrations (and I do), elements from many of these might be treated analytically under common subject headings.

Subjects to Sources

Which brings us to the second method, which we only need mention briefly. When a SAS programmer embarks on an analysis, she will construct some report or reports that are relevant to a subject area. In so doing, she identifies as a matter of course the appropriate data sources which the reports will use, because their organization and field content meets reporting need.

Embarking on the analysis step of a Stage 3 system is similar, only we attend that much more consciously to the relationship between data and business information, and to the fact that decision support analyses concern business subjects. So, what have been the purposes of your recent reporting efforts? Sales analysis? Operations research? Payroll? Surveys? Fine, but what? What sales? What operations? What about the payroll? Surveys about what? What is it about your reporting that you or your clients deal with to make the business analysis? And how do you find the data sources you need to approach these subjects?

You know the answers to these questions; now you just discipline it into an explicit analysis.

Computation Types of Elements

Remember that elements are not the same as data fields or
It is now time to take a hard look, once again, at the data sources from whence our elements may come. Determine redundancies – that is, duplicate sources, for each element. It is important even to create redundancies, in the sense that one source might already have field that represents a summary, while another has a field that could be summarized up to the same thing (i.e., source A contains a field usable as a primitive element, yet the same can in theory can be derived as a summary element of source B).

We also take the very important step here of analyzing the key structure (implicit or explicit) in the sources, and see how they relate. If the source is a database table in 2nd-normal form or better, this information may be readily available. If not, you have to figure it be wary: Each source should at least be a relation, i.e. have some set of "ID" fields that uniquely identify each "row". Hierarchical structure is OK, so long as the hierarchy is completely represented in data fields. Back away from sources that are not organized with some semblance of unique row-ID rules; these will only get you into trouble. Find another source.

Replications

If we can't find another source for an element we simply must have, we'll have to create, after all, a physical data file that the warehouse system will keep up. We may also decide to create physical data files for performance or storage reasons, or for some other reason. Remember, it is perfectly fine to build a physical data warehouse, or parts thereof. If you can't wrest the Legacy by as-if methods as discussed below, or it is too costly or complex to do so, create replicated data, perhaps pre-transformed or pre-summarized, to make the job easier or simpler. You will want to put jobs into production to keep things up, and allow for storage.

DESIGN HIGHLIGHTS

As Proceedings deadline (1/19) looms, I have been drastically rethinking the original design work for the Stage 3 system, i.e., the structure of metadata stores and a couple of the ways the UI works with them. I have decided to pull this, and hope to have a SUGI handout outlining the (revised) Stage 3 design. Once again, if you don't get my handout at SUGI, please write and ask.

CONCLUSION

The main drive in this whole data warehouse movement is to take the data we've already got, and try to figure out what it means to us and how we're going to use it. ... I don't think the data warehouse is a place; it's a process. People don't build data warehouses; they do data warehousing.

Robert Theilheimer (1995)

If you haven't guessed by now, I agree with the META Group's VP for Open Computing and Server Strategies. What's important is how we get to the information and how we get it out; less important is what we go through to do so, including how the data exists in physical form.

The mainframe is not obsolete. It will be at least 15-20 years before massive distributed processing will be practical for large corporations. And the new breath of life in Open MVS and its supporting software architectures, as we will see explode over the next two to three years, will make heads turn once again toward the mainframe as a reasonable way to deliver to users directly. The Legacy is with us in many ramifications and will stay with us for some time. The system we have constructed is equally applicable to any hardware platform, so long as SAS 6.07 or higher is installed. The user interface, separable from everything else, may be replaced as we desire – even ported to another platform, e.g., taking advantage of SAS/Connect® library services, or even via mechanisms not dependent at all on a
running SAS at the PC or workstation.

Whether you build the system or have it built for you (modesty forbids mention of my own services...), you don’t need to go through macro-organizational hoops to do it. You don’t need I.S. for anything except data security. You can give your users the power to attack the Legacy, through your own code, templates and input, as a full-fledged elements-and-subjects system.

You can warehouse NOW!

REFERENCES


TRADEMARKS: SAS, SAS/AF, SAS/Assist, SAS/Graph, SAS/Connect, indeed any nouns or noun phrases herein having “SAS” in them, are registered trademarks ® denotes US registration) or trademarks of SAS Institute, Inc.

This paper is dedicated, with love, to Susan

Contact: Jay A. Jaffe, Ph.D
PO Box 23563
Pleasant Hill CA 94523
JJAFFE @ AIMNET.COM

NOTE: IF REQUESTING SUGI BAGOUTS, PLEASE ENCLOSE $10

1 For our purposes, a “report” does not necessarily mean printed output. I use the term to designate any kind of output results. This may include files for export, print images for download, etc. The term “report” is more convenient as it is clearly singular; we can more easily speak of one “report” than one “output”, and multiple “reports” than “outputs”.

2 and also because Frame development under MVS is a royal pain even if you limit yourself to text widgets. Now that 6.11 is released, I am reconsidering. 6.08 on Windows (and OS/2) suffered a serious incompatibility problem transporting AF catalog to MVS, but 6.11 claims to have fixed that. As SUGI approaches, I’m starting to experiment with the feasibility of developing the UI on the PC and porting the object catalog to MVS (unfortunately, the excellent 6.11 enhancements cannot port to 6.08); after that, we need to see if Frame performance, even if limited to text widgets, is too sluggish. Stay in touch, and I’ll let you know what I learn on both fronts.

3 Under the Windows 3.1 operating environment, for example, one often sees a command bar which may be mouse-clicked. But most Windows applications allow the use of the ALT key pressed with another key to do precisely the same thing. Sometimes, function keys are also assigned to popular actions, generally actions that would otherwise require a second-level menu pulldown (again via mouse or ALT combination).

4 Some people seem to consider SAS Macro Language an invention of the devil. However, it is extremely useful in a wide variety of circumstances. Certainly, it is easy to write incomprehensible macros. But it is as easy to write incomprehensible code in, say, C or assembler. The solution is the same: internally document, and excessively. Assembly code is frequently documented by comment at each and every statement. You don’t need to do quite that, but don’t be stingy with your comments.

5 An event class, as we define it, has no methods. Since, however, it can be represented in an SCL list, an event may in principle be used as instance data for an object subclass (or perhaps, in a 6.11 port, an SCL list model subclass) in the usual O-O sense, to provide general methods for reading, writing, displaying events as some of the context of a variant UI. The example system for this paper does not add an O-O layer, as it is neither the inconvenience outweighs the utility, we don’t care to build components for

multiple applications. We nonetheless observe that our events are object-based, and that they readily share a small set of methods.

6 But in addition, for use within the UI, parameter classes also encapsulate their picklist and error-checking methods (though not in the event store per se). Implementation of parameter methods is beyond scope here.

7 In other words, parameters are inherited by report classes, though the inheritance isn’t full-blown O-O in this system, where a parameter is either specific to the report class, or global for all report classes.

8 Here exactly would be the place to “plug in” an Assist-like screen if we chose to take that approach. Instead of another (third tier) choose screen, we would display the screen designed for the template.

9 Event data is different than metadata, as I use the concepts. It differs from metadata in three fundamental respects. In the first place, it is not about business data, but about what is wanted from these data; event data describes an intended usage. Second, event data is removed from the informative business data: you can change report or template definitions all you want, without affecting the data about which you will report. It’s a one-way relationship; if the business data changes in form, you may have to rethink your reporting, but not vice versa. Finally, event data is about events: What a report looks like now (i.e., until you change it), what the user wants the report to show now (i.e., in a report instance via parameter values). Metadata is in a sense more context; if constructed correctly, it need not change unless and until the underlying forms of the business data change.

10 Actually, the results of your careful analysis likely will turn out such that a user who attempts to combine elements in a more bizarre way will by that very action show they don’t know what the heck they are doing, and when their program bombs or gives ridiculous output, be kind and offer graciously to do an ad-hoc for them.
SELECTED SCREEN DISPLAYS (EXAMPLE STAGE 1 SYSTEM)

Note: To conserve space, I've culled out repetitive screen lines. Where this is done, the border bars are replaced with tildes (~).

+OUTPATIENT REPORTING SYSTEM: Main Menu--------------------------------------+

 HELP QUIT SYSTEM PROFILE
 F1=Help  F4=Output
 Choose an item from below :  or enter Report ID : 

 1 : Schedule Activity
 2 : Appointment Supply
 3 : Booklist
 4 : Referrals
 5 : Panel Management
 6 : Utility Reports

+----------------------------------------------------------------------------+

Display 1: Main Categories Menu

+OPRS: Referrals -------------------------------------------------------------+

 HELP QUIT SYSTEM PROFILE
 F1=Help  F2=List  F3=Return  F4=Output  F10=Submit
 Choose an item from below :  or enter Report ID : 

 1 : REF01 Open Interfacility Referrals Audit
 2 : REF02 Open Intrafacility Referrals Audit
 3 : REF03 Referrals Entered
 4 : REF04 Referrals Entered by Provider
 5 : REF05 Referrals Received
 6 : REF06 Referrals Received by Provider

+----------------------------------------------------------------------------+

Display 2: Reports Menu for Category 4

+REF03: Referrals Entered ----------------------------------------------------+

 HELP QUIT SYSTEM PROFILE
 F1=Help  F2=List  F3=Return  F4=Output  F10=Submit
 TAB to item, then enter choices (or press F2 for a selection list).
 Items marked "*" are required. ':' optional, "~" information-only
 Press F1 for help. Press F10 to submit the report for processing.

: Output Options : 
 Process > Y (S=Batch  O=Online)
 Print Y (Y=Yes  N=No)
 Graph N (Y=Yes  N=No)
 File N (Y=Yes  N=No)

: Data Selections (leave blank for "all")
 Date Range
 Treat F & C > SAC
 Treat Dpt > MED
 Ref Pac
 Ref Dpt
 Ref Status

+----------------------------------------------------------------------------+

Display 3: "Choices" Screen for Report Class REF03
Display 4: Selection List for REF03 Parameter “Ref Status”

Display 5: “Choices” Screen With User Errors
In this example, the user has omitted a required choice (date range), and entered more than one Facility where only one is allowed

Display 6: Output Management Screen