Using OOP to Enhance the SAS System: An Adhoc Reporting System using SAS Batch, 3D Graphics, and SAS/Web Interfaces

Charlie Bastnagel & Kevin Gates,
Healthsource Provident, Chattanooga TN

Abstract

One of the nice features of the SAS System is that if you need functionality that is not currently provided, then using SAS (strangely enough) you can create that functionality yourself and make it appear as if it is apart of the SAS System. Such functionality as

- **An Adhoc Reporting Tool that encompasses main SAS Procedures and runs in batch mode**
- **Structured batch submission from local clients to remote servers**
- **Macros that create 3D Graphics for Hbar, Vbar, Tables etc...**
- **Automated Remote Signon**

can be developed and added to the SAS System that your users use.

This paper discusses the design features of the Adhoc Reporting System. In particular it focuses on the development of new software components written in the SAS AF & Macro Languages that make an object driven Adhoc Reporting System possible.

Introduction

Our Adhoc Reporting System required features such as:

- **A single tool** that provides most of the basic features of the SAS System and yet doesn't require the user to travel across dozens of screens to accomplish basic tasks.
- A simple method to provide and maintain **multi-client to multi-server access to SAS Unix Servers.**
- **Printable Reports and Graphs** that contain 3D Hbar, Vbar, Tables, etc... with no limitations to placement and aspect ratio as occurs when using SAS Graph Templates.
- **Batch Submission** to Local and Remote Servers using Interactive SAS.

Based on our experience with SAS many of the above features were too complicated and time consuming for average users to perform using SAS. All of the features listed above were considered CORE to the adhoc reporting needs of our company.

Environment

Our SAS environment consists of Clients running Windows (3.1, NT, and 95), OS2, and Unix(Aix 4.1), **utilitizing** SAS servers running Unix(Aix 4.1) and MVS, as well as a Unix Server that holds a Redbrick Data Warehouse. The total volume of Data we use SAS to access is 1.2 Terrabytes (700 gig in SAS Data Warehouses, 500 in the Redbrick Data Warehouse).

We employ SAS version 6.11 on all platforms other than MVS. We are currently moving away from MVS and while some of the enhancements we have made, do work in SAS under MVS, our current production environment is Windows, OS2, or Unix Clients using Local or Remote Unix Servers that we connect to using SAS Connect Software.

Design Requirements

What was the business problem that prompted the need for the Adhoc Reporting System?

- Allowing Free Range use of SAS was costly in terms of training, time, and potential for harmful mistakes due to lack of experience. (i.e. Merging Datasets that contained Duplicates in both sets of by variables and ignoring the warning in the SAS log.).

- Signing on to the various remote servers(Unix and MVS Mainframe) and figuring out which server was active was too confusing to many of our users.

- There was no automatable method for creating 3-D graphs. Users had to use other software like Freelance and Powerpoint to get quality 3-D graphs that we could **PRINT**.

- Most importantly Users spent too much time on the clock waiting for jobs to run. Waiting 2-12 hours for a program to complete is unreasonable and severely dampens productivity.

Although not an initial requirement, it soon became apparent that non-SAS users would need to be able take advantage of the Adhoc Reporting System using Web technology. This meant that the Adhoc Reporting System would have to be accessible from our Company's Intranet and potentially to the outside world.

The Web scenario ran like this: A user starts up Netscape and asks for a pre-defined report, adhoc report, or data subset from our SAS or Redbrick Data Warehouses. When the job completes they would be able to retrieve reports, graphs, and data extracts that could be

downloaded for use with other software like MS Access, MS Excel, Paradox, etc....

Design Implementation

The requirement that adhoc requests could come from both within SAS and outside of SAS via the Web meant that the software we developed, had to mainly be non-Visual and object oriented, so that regardless of the GUI Interface, the software would work the same. In other words, our object oriented enhancements to SAS would have to be capable of serving as the muscle under the hood regardless of the car driving it.

Design Methodology

The methodology we used in Creating the Adhoc Reporting System will be addressed them based on the features that needed to meet the design requirements.

Feature 1 - The SAS GUI Interface

Description: A single tool that provides most of the basic features of the SAS System and yet doesn't require the user to travel across dozens of screens to accomplish basic tasks.

The goal of this interface was to allow the user the ability to create an **adhoc request** using widgets like listboxes, radio boxes, checkboxes, text entry viewers, and some composite widgets we created ourselves. When the user was done with their request, they would be able to save their request, so that it could be retrieved and executed later on. In this model, even where clauses were made separate objects, so that they could be used with other reports and applications.

The substance of the **adhoc request** is less **TASK** oriented than what SAS Assist requires and instead is **PROCESS** oriented. SAS Assist and the Query Window which are the main Adhoc GUI tools that come with production SAS, focus mainly on accomplishing one task, a summary of a dataset, a query where you calculated the mean for the dollar variables by state, or where you merged two datasets.

In our business, Healthcare Data Analysis, most of the projects users were tackling rarely stopped with just one merge or query. Using the Query window, they would run a query, then run another query..., and then go to the program editor to get the data the way they really wanted it and finally print their series of reports. Using SAS Assist was abandoned very early on, due to the fact that it was extremely confusing To do a simple proc summary, you might have to travel 4 to 5 screens and if you forgot to sort, well then you got an error. Another problem with using SAS Assist was due to the volume of our libraries and SAS Assist's reliance on SQL Views to the Dictionary.Tables. It might take up to a minute just to bring up a list of available datasets.

What the users needed was a way to string together a serious of tasks and when they are ready send the entire request to run. By allowing them to save their requrests then we could allow them to come back at a later time and make changes to particular tasks. They could run either the whole request or just parts.

We have learned over our two year stint with SAS that users struggle with applications that require them to go from screen to screen to screen. They get lost, not sure where they are at.

For the developer, the use of multiple frames complicates the application because most often parameters must be passed to each frame. Enhancing your application thus becomes more complicated: If you add a new parameter, you will have to change each frame and SCL entry and then recompile each of the frames.

The **METHODOLOGY** we decided to employ was to use one main AF Frame that the user would recognize as the Adhoc Reporting System. If a certain request required a new frame we would either pop up a small dialog type frame, or we would swap out various objects on the main frame and using the _new_ method in the CLASS.CLASS create new widgets as needed. Creating Widgets on the fly is documented in the Class Class section of the Frame Class Dictionary (order number 55146). This manual is pretty much a requirement for doing this kind of development. I also strongly recommend Jeff Cartier OOP Class at the Cary Institute.

The rule for creating new objects at run time what we call **FLYING OBJECTS**, was that all widgets that were not displayed on the frame but would be at some point, would be created using the _new_ method and then swapped out during the initialization of the frame. When the widget was needed, it would be swapped in. Widgets that would only be used in special cases, would be created and terminated as they were needed.

Below is an example of creating a listbox inside a container box named mcontbox that already exists on the frame.

```
MAKELBOX:

/*
                    MAKE List Box
 first make sure widget doesn't already exist. If so and you
try to recreate it without first deleting then you will get an
error.
 */

if listlen(lboxwid) > 0
    then call send(lboxwid, '_TERM_');

/*
Create an Attribute List and prepare to store ListBox
attributes in it.
*/
attrid=makelist();

 /*
```

```
 Set the Region List Parameters for the Listbox
*/
 regionid=makelist();

  rc=setnitemc(regionid, 'C', 'UNITS');
  rc=setnitemn(regionid, 2, 'ulx');
  rc=setnitemn(regionid, 2, 'uly');
  rc=setnitemn(regionid, 24, 'lrx');
  rc=setnitemn(regionid, 18, 'lry');
  rc=setnitemc(regionid, 'CONTBOX', '_PARENT_');

/*
 Insert the Region List and Instance Variables into the
Attribute List, Load the Class,  Create the Object
*/

 rc=setniteml(attrid, regionid,    '_REGION_');
 rc=setnitemc(attrid, 'LISTID',       'LOCATE');
 rc=setnitemc(attrid, 'SCL_LIST',
     '_POPULATE_');
 rc=setnitemc(attrid,  LISTBOX ,   'NAME');
 rc=setnitemc(attrid, 'LISTBOX',    'LABEL');
  lbox_class=loadclass('sashelp.fsp.listbox.class');
 call send(lbox_class, '_NEW_', lboxwid, attrid);


/* to get a list of a widgets properties do

  tempid=makelist();
  call send('LISTBOX', '_GET_PROPERTIES_',
             tempid);
  call putlist(tempid, 'List Box Properties List', 1);
```

Once you have your bearings you could build a library of widgets and turn this whole process into a method.
*/

```
RETURN;
```

The Key component behind the Adhoc Frame is the use of the Dictionary library. The Dictionary library contains tables which you can best access through SQL. These Tables are quite incredible: They give you information about Libraries, Datasets, Catalogs, Macros, External Files, etc... Needless to say they are a key component in any flexible Table driven application.

Dictionary.Columns is a table we use most often in the Adhoc Frame. It contains information about SAS Datasets, like variable names, formats, labels, etc... Here is an example.

Let's say our user wants to do an adoc query or report against a dataset USERDATA.CLAIMS stored on our ASIMOV unix server. In our SCL this is the code to get all the information about that table and then load certain pieces of it into SCL Lists which populate Listboxes on our Frame. The example below creates 4 lists, a list of variable names, and then a list each for the variable types, variable labels, and variable formats. Note you could use lvarlevel to do this with less code but there may be performance problems. Here's the code:

```
GETINFO:

  lib='USERDATA';
  dsn='CLAIMS';

submit continue remote;
  proc sql;
    create table work.datastru  as
    select name, type, label, format
    from dictionary.columns
   where libname=upcase("&lib") and
          memname=upcase("&dsn");
 quit;

 proc download data=work.datastru
                  out=work.datastru
                  status=no;
 run;
endsubmit;

dsid=open('work.datastru','i');

if dsid=0
   then do;
     put 'Dataset Cannot be Opened.';
     return;
   end;

varlid=makelist();
typelid=makelist();
lablid=makelist();
fmtlid=makelist();

do while (fetch(dsid) ^= -1);
    name=getvarc(dsid,varnum(dsid,'NAME'));
    type=getvarc(dsid,varnum(dsid,'TYPE'));
    label=getvarc(dsid,varnum(dsid,'LABEL'));
    format=getvarc(dsid,varnum(dsid,'FORMAT'));

    rc=insertc(varlid,  name,   -1, name);
    rc=insertc(typelid, type,   -1, name);
    rc=insertc(lablid,  label,  -1, name);
    rc=insertc(fmtlid,  format, -1, name);
end;

if dsid > 0 then rc=close(dsid);
RETURN;
```

Using this method you can build very flexible AF Frames. (See Frank Dilorio's various SUGI papers for more on the Dictionary Library).

Feature 2 - Remote Conectivity (The %REMOTE macro)

<u>Description</u>: A simple method to provide and maintain multi-client to multi-server access to SAS and Redbrick Data Warehouses.

As mentioned in the Environment section, we use OS2, Windows, and Unix Clients with Unix and MVS remote servers. Before we developed %REMOTE an average user would have to issue the following to sign on to a

MVS and a Unix Server setting the SAS Connect options for each server.

While the SAS Connect code is not all that complicated to an experienced SAS Connect User, to a Data Analyst this is a bit much to expect. Plus for every Unix Server you have, and we have 3, there will have to be a separate Unix Script File for each. Every time you have to change that script file you will have to change all 3 of them.

This dilemma gave birth to the idea for a macro program that would use a SAS Dataset called SYS_PROF (system profile) to manage all of our remote connectivity. To add a new server we would just add a new record to it. This would provide us with a centralized methodology for managing remote connectivity. We wrote the %remote macro to handle our Remote Server Connectivity.

A call to %REMOTE signing onto the Asimov Unix Server would look like the following:

```
%REMOTE(remote=asimov);
```

We developed an AF Password Frame which serves as an Interface to the remote macro.

The password frame, which looks like any other signon type frame, could either be called directly through SCL or if you call %REMOTE, it would invoke the password frame for you if the user was not already signed on.

```
%REMOTE(remote=asimov);
```

This password frame is used instead of the SAS Connect signon. This means that you can use %REMOTE for signing on in batch mode, or if you are in an interactive session you can use the Password Frame. In either case it is the %REMOTE macro that does all the remote connectivity work.

How does %REMOTE work? I am going to mainly focus on the Unix Server part of this. For MVS %REMOTE merely submits the APPC settings and then issues a signon.

%REMOTE uses a SAS dataset SYS_PROF to create the necessary variables to supply to SAS Connect and then issues a signon. It also will look in the user's SASROOT directory and if it finds a file named after the server such as ASIMOV.SAS, it will submit that as an Autoexec.SAS file once it has made the connection.

Adding new servers is very easy. To give access to another server, add a record to the SYS_PROF dataset.Here is a partial put statment listing of the SYS_PROF Dataset;

```
REMOTE=ASIMOV
SERVDESC=HS Provident Unix, TN - ASIMOV
DNS_NAME=ASIMOV
COMAMID=TCP
UNIXDIR=/HOME/SAS611
```

The above information is used by %REMOTE to supply macro variables to the TCPAUTO.SCR connectivity script, which is a modified copy of the TCPUNIX.SCR script which comes with SAS.

Feature 3 - 3D Graphs and Presentation Quality Reports

Description: Printable Graphs and Reports that contain 3D Hbar, Vbar, Tables, etc... with no limitations to the number of graphs or other objects on the page

Creating a library of 3D graphic objects and developing our own Report Class, was a very difficult process. We knew early on after playing around with ANNOTATE that DSGI (Data Step Graphics Interface) was the way to develop our graphic objects. We are currently developing 3-D Pie Charts and our own Report Class; The objects that are completed are:

Hbar
Vbar
Stack Hbar
Stack Vbar
LinePlot
Table Draw (for column seperated tables)
Logo Builder (for creating logos)
Text Drawer (for creating Title Pages and miscellaneous images)

What I mean when I say that Hbar and the others are objects is that the Hbar macro, %hbar, is independent and yet can be used along side other graphic objects or inside them if need be. Our Graph Objects do not use proc gchart or any other SAS Graph feature except for DSGI. In other words there is no hard-coding. All of our objects are developed to run on any platform that is running SAS 6.11.

As an example, if in a report exhibit I need Two 3D Hbar Charts and Two Shaded Tables, along with a fancy logo at the top left corner, then the exhibit (this is what we call a report with graphs) would look like this.

```
%hbar(...parameters1,2,3..);
%hbar(...parameters1,2,3..);
%tbledraw(...parameters1,2,3..);
%tbledraw(...parameters1,2,3..);
%logo(...parameters1,2,3..);
```

Each of the graphic objects contains mostly the same parameters. Here are some of them.

```
show=Display Y/N
insert=If the graph already exists. Insert this
       current graph into it.
in_lib=input data library
in_dsn=input dataset name
gr_lib=store graph in this library
gr_cat=graph catalog
gr_name=graph name
gr_desc=graph description
effect=how much 3D effect.
```

autosize=Auto-Resize the graph if too big
scale=Size of the Object
lmargin=Where to begin left corner of object
tmarign=Where to begin top of object
logo=The logo to use for this graph

One of the most difficult standards to maintain was the **Insert feature** of all of these objects. If a graph already existed and the Insert parameter was set to Y then the new object would be placed on the existing graph. The method we used is this: It is like running a sheet of paper on a conveyor belt. Whenever one of our graphic objects is called and the insert flag is Y, that object stamps the new object onto the existing graph. Due to our flexibility concerns we wanted the sheet of paper to be like a canvas. If the user needed 100 objects on a page, then the graphic objects would put a 100 objects on that sheet of paper, even if they wrote over one another.

There is only one way to do this kind of insertion in DSGI. DSGI will not allow you to insert a graph named graph1 into the same graph graph1. Instead you have to rename graph1 to something else and then insert it into a graph called graph1. When you are done you will have two graphs on one sheet.

Because SAS Graph does not store GRSEGS by the name you give them, but rather by the datetime stamp of when they were created, it does no good to rename a graph to some consistent name like GR_COPY and then insert it. No, you have to rename the entry to a datetime stamp name that is legal in SAS and yet is unique within that catalog. If you try to name a graph something that already exists in the current graph catalog SAS will give you its own name like graph11 and not the one you want.

How did we get around this? First of all we wrote a Macro called GRMANAGE that manages our Graph Catalogs for us. In that macro we converted the current name of the graph into a datetime stamp name: this means we took the date and time and converted as follows:

Convert Datetime into a legal SAS Name:

Month - A letter A-L for the month Jan=A Dec=L
Day of the Month - ex. 09 for the Jan 9th
Hour of the Day - A letter for Hour A-X
Minute of the Hour - ex. 30 for 12:30 on Jan 9th.
Second of the Minute - ex. 58 for 12:30.58

For Example: Jan 8 at 12:30:58 becomes A09N3058

Even with the advent of SAS's New Adhoc Reporting Tool that is to be released sometime early 1997, we will still have to have use our own in house developed objects, due to the fact that SAS's new tool will not run in batch mode. You have to run it interactively. We are hoping that perhaps we can subclass certain parts of the New Adhoc Tool if it is truely object driven, but as it is, it will not meet our company's reporting needs so the months of development effort were not wasted and our graphic objects and new report class will probably be needed for at least the next two years.

Feature 4 - SASBATCH Job Submission

Description: Batch Submission to Local and Remote Servers using Interactive SAS.

Since we do not run SAS interactively on any of our Unix Servers, we use the SAS Connect Product to give us connectivity from our OS2, Windows, and Unix Clients to the various Unix Servers. What this means is that when a user submits a program they must wait on the clock for that job to finish before they can continue. With 1.2 terrabytes of data, two and three hours on the clock per submission is not acceptible.

This prompted us to develop the SASBATCH macro (%SASBATCH). The goal of the SASBATCH macro is to take SAS programs *(*.SAS files)* or Source Entries *(library.catalog.entry.source)* and to submit them in batch mode on any of the various Unix Servers. SASBATCH keeps track of the batch job using a performance database that has a record for each job submitted, containing information like the userid, the program, how long it took to run the job, how much memory it used, how many sas users were on the system when the job was submitted. When the job completes the user is notified by email. The email message in the email will contain all the information about the job including whether the job had any errors.

To handle all this SASBATCH employs metaprogramming techniques to write new programs that are uploaded to the remote server and then executed.

The goal of SASBATCH is to write a program that when executed on the remote server does the following:

- Begin the Performance Database
- Include the Users Program and Execute it.
- End the Performance Database and Send Email

Here is a call to SASBATCH:

```
%sasbatch(prgtable=,
        program=c:\programs\houses.sas ,
        progloc= lan,
        loc=    unix,
        remote= asimov,
        autoexec=/sas/sas611/autoexec.sas,
        config= /sas/sas611/config.sas611,
        options= -noterminal,
        showlog= ,
        project=North Carolina,
        desc=  Claims Analysis,
        pdb= userdata.pdb,
        notify=Y
        );
```

Lets look at some of SASBATCH's parameters: **prgtable** is the name of a dataset that contains a list of programs to be run, **program** is the name of a program to submit if prgtable is blank, **progloc** is the location of the program, **loc** is the location where the program should be executed, **pdb** is a performance database that the user can use,

notify is a flag that indicates whether an email is to be sent when the job is complete.

The program table prgtable parameter will look for a variable named program and submit all the observations as separate jobs. Here is an example of how a developer might create such a table.

```
data work.programs;
length program $80;
input program $ 1-80;
cards;
test.sasbatch.testprg1.source
test.sasbatch.testprg2.source
test.sasbatch.testprg3.source;
run;
```

One of the problems with running SAS jobs in batch mode is that reports longer than one page are usually useless because the reports are written to a text file and you lose your pagebreaks. One way around this is to set the Pagesize to 32767.

We have written an interface to SASBATCH that is called Batch Manager - BAT-MAN. BAT-MAN is an AF Application that uses the Text Viewer Class, Toolbar Class and Preview Window to give a user the ability to create and edit SAS programs and Source entries, Submit those jobs in batch mode, View and print the logs and reports that are stored on the remote server, and give them performance information about jobs that have been submitted such as: Is the job done, if so how long did it take etc....

Feature 5 - Web Interface

Description: Provide Capability of running Adhoc Reporting System form the World Wide Web using a Netscape browser.

This is really the same as Feature 1, the SAS Gui Interface, except that we will not be able to use SAS's excellent AF products. Furthermore, because our corporation cannot as yet implement a 32 bit standard, we are writing the Web Interface using HTML and Javascript. We are not using any tool that requires us to load new software on the client or requires a certain Operating System as does Java.

This means that the interface
is being developed using Netscape 2.02 +, HTML, and Javascript. On the Unix side where the queries, reports, and graphs will be created using the Adhoc Reporting System, we are using Perl , Unix Scripts, and SAS. The SAS Graph Grsegs will automatically be converted to JPEG files using SAS's new JPEG Conversion driver in version 6.12. For whole packages of exhibits the SAS Graphs will be converted into a single postscript file that can be printed locally.

Conclusion

This paper has attempted to demonstrate that through the use of Object Oriented Methodologies, the SAS System can be enhanced in unlimited ways.

By developing our own SASBATCH component for SAS, we have increased users productivty many times over. The creation of our own 3D graphics objects makes the development of reporting packages much easier, because we don't have to use other software to do our graphs and build the link that would require. In Treating the features of the Adhoc Reporting System as component objects we have been able to improve our SAS use as a whole and through Web technologies provide a path for our future development.

Charlie Bastnagel can be reached via Email at batnach@hlthsrc.com

Kevin Gates can be reached via Email at gatesk@hlthsrc.com

References

SAS Macro Language

SAS/AF Software Frame Class Dictionary

SAS/AF Software Development Concepts

SCL Reference 2nd Edition

We would like to thank Mark Dalesandro and Di Meng who helped in the development and pushed the envelope of SAS AF development at our Company.

Thanks to John McCall at the Atlanta Institute for getting us started. Thanks also to Jeff Cardier, Eric Waldenbauer, Jude Redmond, and Alex Fernandez at the Cary Institute for their support in this madness.