

Using SAS/AF FRAME Entries to Develop Client/Server Applications

Jeff Butler
Internal Revenue Service
Compliance Research Division

Introduction

Client/Server computing is an enormously popular and efficient solution to a wide variety of rightsizing or reengineering problems. This is primarily due to the growth of small but increasingly powerful network servers, robust network operating systems and transmission protocols, the use of the Structured Query Language (SQL), and the adoption of object-oriented programming techniques for rapid prototyping and development.

SAS/AF® FRAME entries belong to this latter category, providing a highly effective means of developing graphical interfaces between users and database engines. This paper introduces the role of SAS/AF FRAME entries for this purpose, demonstrating their use in conjunction with the Screen Control Language (SCL) and SQL for interfacing with SAS datasets and other database engines. Strategies involved in developing a typical client/server application will expose the various stages of development, including database connection, remote processing, and error trapping.

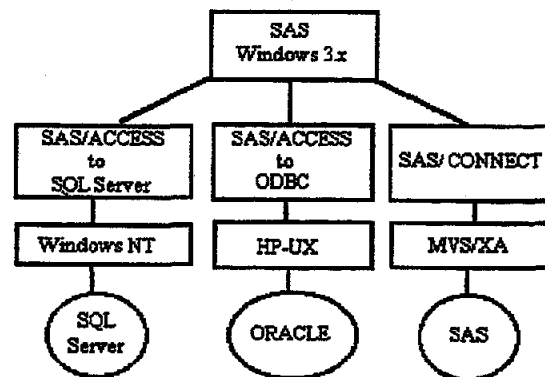
Client/Server Computing

There is currently no single definition of client/server computing, but all definitions have a common basis: the ability to process information across different environments. For example, a Windows application (client) might download data from an Oracle® database table (server) running under Unix. Or, an OS/2® application might update records in a DB/2 database table residing on an MVS host. In yet another example, client software may request that the remote host process a large database table that would otherwise be too computationally expensive for the local host, then download only the data necessary for local presentation by the client. This last method is especially useful when millions of records are being considered.

It is obvious that the particular client/server solution chosen by a business organization depends

on a number of factors, such as database size, resource constraints, and ultimately customer needs. Once adopted, however, the client/server model provides both the functionality and expandability for most business needs.

The SAS System® supplies a comprehensive set of client/server tools for enterprise-wide computing needs, including SAS/CONNECT® for remote server connectivity, SAS/ACCESS® for using third party database engines, and SAS/SHARE® for managing simultaneous database processing in a multi-user environment. The diagram below illustrates the flexibility provided by the SAS System. In this example, SAS is sometimes client, sometimes server, and sometimes both.



SAS Institute, Inc. has also recently included object-oriented development tools to its list of client/server products: SAS/AF FRAME entries and SAS/EIS®. These products provide the capability to develop client interfaces that make complex activities transparent to the user. This is important, since the real hallmark of any good client/server solution is the seamless integration of a wide range of information services.

Object-Oriented Programming

The widespread presence of graphical user interfaces (GUIs) has led to a number of important advances for both application developers and end-users. For developers, one area that has practically

revolutionized both the theory and practice of modern software engineering principles has been object-oriented programming (OOP).

While a discussion of OOP methods is beyond the scope of this paper, it is important to address two major areas of traditional software development that OOP methods have virtually redefined: prototyping and reusability.

Prior to GUIs and OOP, procedural programming in character-based systems resulted in a certain degree of redundancy. For example, a menu for one screen typically couldn't be used in another, although the functions used for displaying the menu might be. With OOP, graphical *objects* such as list boxes can be used over and over in different situations by manipulating one or more of their properties, or *methods*. Moreover, new user-defined objects can be quickly designed and developed that possess, or *inherit*, most of the characteristics of existing objects.

The ability to rapidly prototype new applications simply by using existing objects has meant shorter development cycles and substantially reduced maintenance costs for most businesses, a result that everyone benefits from.

SAS/AF FRAME entries give developers the ability to use OOP in combination with the more traditional programming features of SCL, the Macro Language, and the DATA step. As a result, developers who have historically relied on SAS/AF PROGRAM entries are probably writing significantly more SCL code than necessary, and it might prove beneficial to explore the possibility of converting some PROGRAM entries to FRAME entries if product expandability is an issue.

SAS/AF FRAME Entries and SQL

The role of SAS/AF FRAME entries for client/server models is straightforward: to provide the interface between the user and the database engine. It is the responsibility of the developer to ensure that this interface is as transparent as possible.

Features incorporated into FRAME entries will obviously vary from across applications, but there is typically a certain degree of similarity in all client/server solutions. For example, most well-known database management systems incorporate SQL as their programming language, and thus clients requesting data services will do so using SQL.

SQL Considerations

There are several consequences for development of FRAME entries as a result of using SQL. First, for simple applications that rely on *ad hoc* queries for generating reports and graphics, the *SELECT*, *FROM*, *WHERE*, *GROUP BY*, and *ORDER BY* statements will occur very frequently. At a minimum, this implies the need for *WHERE...* and *GROUP BY...* push buttons on many FRAME entries for subsetting and aggregation.

Second, each part of a sequentially arranged query in one or more FRAME entries will have to be *parsed* into SQL statements native to the external database, including the SQL implementation in the SAS System. This could be accomplished using macro variables, as the following code fragment illustrates.

- * Table fields selected from a list box must be
- * delimited by commas and concatenated for a
- * SELECT statement.
- * Database alias is TEMP

```
proc sql;
  select * from connection to temp
         (select &selstr from &table);
quit;
```

WHERE expressions are especially prevalent in most database applications, particularly when only one or several records are being requested from millions.

Unfortunately, the SAS System limits character variables to a length of 200, a severe constraint on lengthy *WHERE* expressions. Although there may be different solutions to this problem, perhaps the most straightforward is to use macro variables for each 200 character string, as in the following code fragment.

- * Assumes there are *n* macro variables, wstr1,
- * wstr2,...,wstrn, each of length 200 containing
- * where information

```
%macro wherestr();
  %local j;
  where
  %do j=1 %to &n;
    &&wstr&j
  %end;
%mend wherestr;
```

```
proc sql;
  select * from connection to temp
    (select &selstr from &table
      %wherestr);
quit;
```

Third, while simple applications may require access to a single database, the SAS System provides the capability for SQL queries against multiple databases under different operating systems. This is handled by SAS/CONNECT and SAS/ACCESS software.

For example, the client interface developed with SAS/AF FRAME entries under Windows may need to access employee salaries in an Oracle database under Unix as well as employee bonuses in a SQL Server® database under Windows NT. While the table join on employee name will not pose any problem, special consideration must be given to ensure unique database *aliases*.

The code fragment below illustrates a join on different databases for our salary and bonus example.

```
* Assumes SAS/ACCESS to ODBC connection
* to Oracle and SAS/ACCESS to SQL Server
* connection to SQL Server.
```

```
proc sql;
  select * from connection to ODBC
    (create table temp1 as
      select name, salary from dbtable);
  select * from connection to SQLServer
    (create table temp2 as
      select name, bonus from dbtable);
  select i.name, i.salary, j.bonus
    from temp1 i, temp2 j
    where i.name=j.name;
quit;
```

Although not demonstrated above, it is generally the case that, depending on the degree of complexity, such an operation would tend to make extensive use of the SAS macro language.

Error Trapping

When exploiting various features of an external database, FRAME entries should be developed so that trappable errors such as connection and login failures can be displayed. A single FRAME entry can be reused for this purpose, with each unique message being written to a text entry box.

The SAS System provides two macro variables, 'SQLXRC', and 'SQLXMSG', which contain the return code number and description, respectively, of the last SQL operation on the external database. (For SAS formatted datasets, use 'SQLRC'.) That is, each time the SQL *procedure* runs, these global variables are available to check for successful connection, syntax error, permission denial, etc.

A special case arises, however, when checking the success of a connection to a remote host where permanent SAS datasets are being used as the database storage mechanism. It is necessary in this circumstance to read the contents of the SAS log and check for specific NOTES or ERRORS. The following code fragment gives an example of how to check for successful connection of a local Windows session to a remote Unix host.

```
* Step 1: Temporarily redirect the SAS log to
* an external file and login to remote Unix host.
```

```
submit continue;
  proc printto log='sas\temp.log' new; run;
  %let rmtnode=152.225.2.5;
  filename rlink 'sas\connect\saslink\tcpunx.scr';
  options comamid=tcp remote=rmtnode;
  signon rmtnode;
```

```
* Step 2: Release SAS log, read it, and check
* for successful connection to remote session
```

```
proc printto; run;
%global found;
data _null_;
  infile 'sas\temp.log' length=len end=last;
  input @;
  vlen=len-1;
  input text $varying132. vlen;
  string="NOTE: SAS/CONNECT conversation
    established.";
  if index(text,string)>0 then do;
    call symput('found','TRUE');
    stop;
  end;
  if last then call symput('found','FALSE');
run;
endsubmit;

if symget('found')='TRUE' then do;
  ...
else call display('ERRMSG.FRAME');
```

Database Security

One additional issue that is part of many client/server strategies is data security. Most well known database management systems, such as those used as examples throughout this paper, have adequate security features necessary for most application needs. These features include access to the database, tables within the database, and fields within tables. When the user connects to an external database for the first time, as in the following code fragment, a userid/password combination is either

```
* Connect to Sybase® database using
* SAS/ACCESS to Sybase

proc sql;
  connect to sybase (user=&userid, pass=&pw);
quit;

if symget('SQLXRC')^=0 then
  call display('ERRMSG.FRAME');
else
  ...
```

registered or rejected by the database. Once again, a reusable FRAME entry for displaying error messages and their contents could be created. In the next code fragment, the user has been successfully connected to the database, but has no permission to the field

```
proc sql;
  select * from connection to sybase
    (select ssn, name, address
     from &table);
quit;
```

'SSN' in the requested table. When the status of the macro variable 'SQLXRC' is checked, the ERRORMSG.FRAME entry is called and a *text box* could be used to display the contents of the macro variable 'SQLXMSG'.

Strategies for using FRAME Entries in Client/Server Models

All good client/server models have one thing in common: the seamless integration of a wide variety of services. Making multiple external database queries transparent to the user, for example, is a hallmark of a well respected client/server solution. SAS/AF FRAME entries provide the tools for

developing client interfaces that support such requirements. Some hints about what to look for in pursuing this are provided below.

Connecting to the Database

We have already seen several examples of how to connect to an external database. The most important thing to remember here is the use of the macro variables 'SQLXRC' and 'SQLXMSG' for retrieving the status of the connection. Moreover, these variable should be checked after each call to the database to determine the status of the query. Again, a reusable FRAME entry for displaying any errors or warnings trapped in this way is very useful.

Location of Data Dictionaries

One important consideration that is often overlooked is whether to store data dictionaries and other look-up tables locally or as a table in the external database.

If security is an issue, data dictionaries will have to be stored in the external database while the application is not in use. However, as shown below, they will ultimately have to be transferred to the client at run-time.

Let's assume that the operation involves populating a list box with field names from a database table. Doing so can be done in a number of ways, such as specifying a dataset name, a dataset ID, or an SCL list name in the 'Fill Type' box of the object attribute window. Specifying a dataset name or dataset ID requires a *local* library reference; an SCL list simply requires a *list name* which will be filled with items from a local source as well.

It is obvious that if the list box items reside in an external database table, which they typically will, they will need to be transferred to the client. The only remaining issue is how and when. Consider the following code fragment, which populates a list box with items selected from a SAS/ACCESS descriptor

```
INIT:
  submit continue;
  proc sql;
    create view ddict as
      select * from connection to SQLServer
        (select * from tablename);
  quit;
  if symget('SQLXRC')=0 then dsid=open('DDICT');
  RETURN;
```

previously created on the relevant database table. As you can see, this may take some time depending on server traffic, table size, and so on. Perhaps a more efficient way of handling all such instances would be to create these SAS/ACCESS views while the application is initializing, that is, by including the necessary statements in the AUTOEXEC.SAS file. Of course, if 20 or more data dictionaries need to be created for the client, this too may be inefficient.

Messaging

For client/server applications which rely extensively on external database engines for most data processing, it is imperative to keep the user informed of any or all activity on the remote host that may impact the current task. A fast, efficient messaging interface will do this. This is also an area that serves as perhaps the best introduction to the concept of reusability.

Client Utilities

It is obvious that security needs will vary widely from application to application. In fact, some developers may have no reason to incorporate security features into their application at all.

Database security has been previously discussed in terms of permissions to the database, its tables, and their fields. Most of these permissions are often thought of as being determined by a database administrator who implements them from the database server itself. However, most major database vendors supply client utilities for their database engines which allow end-users with appropriate permission levels to do anything on the client that an administrator can on the server.

SAS/AF FRAME entries can be used to create such utilities. That is, if a new user needs to be added to the database, or if existing permissions need to be modified, FRAME entries can be developed to accommodate these needs. This is because once an external database is accessed via the SQL passthrough facility, its native language is in force, not that of the SAS System. As a result, FRAME entries can be developed to manipulate any SQL command native to that database. This is a very powerful feature that often goes unnoticed.

Reports

FRAME entries for creating reports can be very "busy" with widgets depending on the level of functionality desired. For example, in addition to a

list box for selecting table fields and a text box for SQL where expressions, there may be a need for an SQL 'group by' statement or table joins.

Furthermore, each instance of a where expression, group by statement or join may need a separate FRAME entry altogether.

For some applications, it may be important to prevent user access to SAS Display Manager windows, such as OUTPUT. For report-style output, this can be done by writing to the PREVIEW buffer window and displaying this window with a customized menu bar. The following code fragment illustrates this.

```
* Assumes result of current query is written to SAS
* OUTPUT window.
* Save contents of OUTPUT window to a
* catalog entry called TEMP, copy to PREVIEW
* buffer, and define the window attributes

rc=woutput('save','libref.catalog.temp.output');
rc=preview('copy','libref.catalog.temp.output');
call execcmd('color back white; color message red;
              setpmenu libref.catalog.entry.pmenu;
              ...');
call wregion(1,1,30,80);
rc=preview('browse','Window Title');

* When the END command is issued, clear the
* PREVIEW buffer and OUTPUT windows

rc=preview('clear');
rc=woutput('clear');
```

Graphics Output

SAS/GRAPH® output can be incorporated into FRAME entries by using either the SAS/GRAPH Output or Graphics object classes. Most client/server applications will let the client software generate the graphic image after a database request has been satisfied, and using SAS/GRAPH in this capacity would be no different.

Like report writing, a graph can be generated from an existing SAS dataset or SAS/ACCESS view that resulted from a database query, but will typically require more local processing time than a report. As a result, it is a good idea to design graphics-intensive applications so that 'drilling-down' or 'rolling-up' of the data will be handled exclusively by the database

server, relegating the client to a presentation-only status.

If the client/server solution involves SAS as both client and server, however, the processing power of the remote host can be taken advantage of for actually *producing* the graph. This is especially useful for complex tasks, such as generating three dimensional surfaces or detailed geographic maps. In this case, the steps involved are shown in the following code fragment.

```
* This example assumes a graph named GTEMP has
* been created on the remote host and placed in a
* catalog named libref.GRAPH5.

* Step 1. Download the catalog entry GTEMP,
* which will be written locally to WORK.GSEG.

submit continue remote;
  goptions device=grlink;
  proc greplay igout=libref.GRAPH5 nodisplay;
    replay GTEMP;
  quit;
endsubmit;

* Step 2. Check that the downloaded entry exists,
* then display it.

if cexist('WORK.GSEG.GTEMP.GRSEG') then
  graph=('WORK.GSEG.GTEMP.GRSEG');
else ...
```

Reusability

The advantages of reusability cannot be overemphasized. Many client/server applications will offer more than enough opportunities to exploit this concept. FRAME entries provide the tools for doing so.

Aside from the obvious approach to reusing a messaging window as previously discussed, other areas in a typical client/server application can benefit as well. For example, if a FRAME entry used for collecting report query information is not too complicated, it could be reused to collect graphics query information as well. Perhaps the FRAME entry consists mainly of a list box for field names to select, a *WHERE...* push button, and a *GROUP BY...* push button as well. There is no reason that this one FRAME entry cannot be used for both report and graphics queries. Of course, even when entire

FRAME entries cannot be reused, it should be kept in mind that individual graphical objects can be.

Conclusion

SAS/AF FRAME entries are an effective means of creating efficient, user-friendly interfaces for client/server applications. FRAME entries provide the tools to meet a wide range of client/server needs, such as messaging systems, client utilities for database engines, and reusability.

References

SAS Institute (1990), *SAS/CONNECT Software: Usage and Reference, Version 6, First Edition*.

SAS Institute (1992), *SAS Technical Report P-247. SAS/ACCESS Interfaces for UNIX Systems and Derivatives: SQL Pass-Through Facility*.

SAS Institute (1993), *SAS/AF Software: FRAME Entry, Usage and Reference, Version 6, First Edition*.

SAS is a registered trademark of SAS Institute, Inc.