

SAS® and Client/Server: An Approach

Robert M. Ormsby, Computer Sciences Corporation PSG, Rockville, MD

Abstract

This paper presents a methodology using the SAS System to implement a client/server application. It provides a definition of the term "client/server" and discusses when it may be appropriate to develop a client/server application. The paper begins by describing the application we developed for a national health insurance company. It goes on to illustrate our approach using pseudo code; it continues by implementing the pseudo code using base SAS, macro program statements, and SAS/CONNECT™ statements. The paper concludes by listing the advantages and disadvantages of using the SAS System.

Introduction

This paper considers an application to "be" client/server when it uses multiple processes to accomplish a task. The relationship of these processes is such that one process (the client) invokes another process (the server) to accomplish some task. An approach to client/server computing calls for the server to carry out the majority of input and processing tasks while the client is responsible for managing the output functions. This type of approach is common in organizations that have personal computers connected to a mainframe computer. It is easy to see how these platforms complement each other: the mainframe easily stores and processes large amounts of data but does not provide a very friendly interface. Desktops, however, have friendlier interfaces (graphical user interfaces [GUI] like Microsoft Windows) but have limited storage and processing capabilities.

A national health insurance company that operates in this type of environment asked us to develop a system that reports the results of their participation in the federal government's annual "Open Season." Once a year federal employees have the opportunity to enroll in one of several health care plans. Employees may also change health plans, moving from one company to another. The government refers to this enrollment period as "Open Season," and, although the enrollment period itself lasts only about one month, it takes approximately three months for the all the transactions to flow in from around the world. In previous seasons the transactions arrived at the company's off-site computer operations center, where programs running on a mainframe computer generated daily reports summarizing the transactions. The operations center staff hand-delivered these reports to managers located across town.

The insurance company asked us to create an electronic version of the summary reports. The transactions would still arrive at the computer operations center, but instead of hand delivering reports, the new system would "deliver" the reports every morning to all the managers automatically; all the managers would have to do is click on an icon to see the daily results.

This paper describes a simple way of using the SAS System to implement this type of client/server application.

Client Environment

The client platform is a personal computer (PC) running under Microsoft Windows version 3.1. The base SAS product and SAS/CONNECT (version 6.08 for Windows) are on the machine. The PC is part of a token-ring network. The network connects to the mainframe through a gateway. All the SAS code on this platform is in macros. The code itself consists primarily of macro %DO %UNTIL statements and within these statements are typical DATA steps and procedures. There are also SAS/CONNECT statements and procedures. All SAS code on the client executes in an interactive session.

Server Environment

The server platform is an IBM mainframe using the MVS operating system. The base SAS product and SAS/Connect (version 6.07) are on the machine. The majority of SAS code on this platform consists of macro program statements, file and library references, and DATA steps. Like the client platform, the server also uses SAS/CONNECT statements and procedures. The majority of SAS code executed on the server runs in an interactive SAS session. However, some SAS code runs in a batch environment.

The Connection

IRMLAN Client for Windows serves as the connection between the client and the server. SAS/CONNECT is the conduit that enables the SAS code running on the client platform to instruct the server to execute interactive or batch SAS programs. SAS/CONNECT also provides the capability for the platforms to transmit data between each other.

The Approach

The following pseudo code provides the basic structure for supporting this client/server application. Match the numbers in the pseudo-code to the explanations. Remember that the client controls the server's actions; therefore, the server does not do anything unless the client tells it to.

```

1  do (client executes following code)
2    do until (data is available on
      server platform)
      client instructs server to check
      if data is available
      if data is available
      then client instructs server to
        process the data
      end
3    do until (server completes its
      process)
      client checks if server
      completed processing the
      data
      if server completed processing
      then client continues to
        step 4
      end
4    do until (client receives data from
      server)
      client instructs server to
      transfer data to client
      end
5    do until (client completes its
      process)
      client processes data
      end
end

```

One of the first things that is obvious about the pseudo code is that it consists of several iterative "do" constructs. The reason for this is that there is a timing consideration that exists when the client is interacting with the server. For example, in step 2 the client wants the server to process some data. However, before the server can process the data, the data must be available. Control loops until the data is available and the server can process the data. The entire application relies on this approach.

The Code

The following SAS code is an implementation of the pseudo-code presented earlier. There is not enough space to present the entire application; but the "condensed" code is sufficient to illustrate the approach.

Following the code is an explanation of each section. The code is numbered--match the code numbers to the explanations. Note that the numbering also corresponds to the pseudo code. This permits you to trace the logic from pseudo code to program statements to explanations.

```

1.  %macro driver;
1.1    filename rlink "c:\script.txt";
1.2    %let stopit = 0;

2.  %do %until(%stopit = 1);
2.1    signon rlink;

2.2    rsubmit;
      filename infile "input-file";
2.3    %if %sysfilrc = 0
      %then %let stopit=1;
      %else %let stopit=0;

2.4    data compcode;
      stopit=%stopit;
      run;

      proc download
        data = compcode
        out = compcode;
      run;
      endrsubmit;

2.5    signoff rlink;

2.6    data _null_;
      set compcode;
      call symput('stopit',stopit);
      run;
      %end;

2.7    %do %until(%stopit=1);
      signon rlink;
      rsubmit;
      data _null_;
      x submit
      ('sas.batch.program');
      run;
      endrsubmit;

      signoff rlink;
      %chkcode
      %end;

3    %do %until(%stopit=1);
3.1    data _null_;
      sleep = wakeup(300);
      run;

3.2    signon rlink;
3.3    rsubmit;
      libname library
      "sas.data.library";

      proc download
        data = library.compcode
        out = compcode;
      run;
      endrsubmit;

      signoff rlink;
      %chkcode
      %end;

4    %do %until(%stopit=1);
      signon rlink;
      rsubmit;
4.1    data compcode;
      stopit = 0;

```

```

run;

4.2      libname "sas.data.library";

        proc download
          data = library.summary
          out = summary;
        run;
        endrsubmit;

        signoff rlink;
      %end;

5      options xsync;

        x 'c:\pdxwin\pdxwin.exe
          startup.sdl';

        options noxsync;
      %mend driver;

6 %driver;

1.      The client controls the entire
        application. %DRIVER resides on the
        client. To start the application,
        invoke %DRIVER.

1.1     The FILENAME statement references a
        SAS/CONNECT script file. The script
        file contains instructions that the
        local SAS session uses to establish a
        remote SAS session.

1.2     STOPIT is a macro variable that
        controls when the various do structures
        end.

2.      This section of code checks if the data
        is available to the server. It will
        continue to check until macro variable
        STOPIT = 1.

2.1     SIGNON is a SAS/CONNECT statement that
        connects to the mainframe and
        establishes a remote SAS session on the
        mainframe. RLINK is the file reference
        established in section 1.1.

        This statement enables the client to
        establish a connection with the server.
        Now two SAS sessions are active: one on
        the desktop (client or local) and one
        on the mainframe (server or remote).

2.2     The client uses the SAS/CONNECT
        statements RSUBMIT and ENDRSUBMIT to
        send SAS statements to the remote SAS
        session.

2.3     The client is now instructing the
        remote SAS session to establish a file
        reference to an input file. If
        FILENAME executes correctly, then the
        automatic macro variable SYSFILRC = 0.

2.4     Create a temporary data set on the
        server with 1 variable STOPIT that
        receives its value based on the
        preceding macro statements. Procedure

```

DOWNLOAD copies the data set COMPCODE from the server to the client.

2.5 The SAS/CONNECT statement SIGNOFF terminates the remote SAS session using the script file referenced by RLINK.

2.6 The client now examines the contents of the data set COMPCODE that it received from the server. The value of macro variable STOPIT determines whether or not the macro variable STOPIT is "1" or "0". If the value is "0", then the DO BLOCK repeats. A "1" means the data is available and the client can continue to the next phase.

This code is in a macro called %CHKCODE. Subsequent sections of code invoke %CHKCODE.

2.7 In this section of code, the client establishes a remote SAS session on the server, and submits a SAS program to the batch machine. This program creates data sets containing summary data. The last thing the batch program does before it terminates is to set a flag that indicates it ran successfully. Later on the client will access this flag to determine the batch program's status. Note that when the client instructs the server to submit the batch SAS program, there are now three SAS sessions active: an interactive session on the client and two sessions on the server--an interactive one and a batch one.

After submitting the program the client terminates the remote session. The batch session does not terminate with the remote session. It is a separate entity.

3. Now the client checks periodically to see if the batch program has finished executing.

3.1 This DATA_NULL step tells the client to sleep for five minutes before checking the status of the batch job.

3.2 After sleeping, the client establishes a remote SAS session on the server.

3.3 The client now establishes a library reference on the server and instructs the server to copy the data set to the client. The client terminates the remote session and invokes the macro %CHKCODE (see step 2.6) to ascertain the status of the batch job. If the job has not finished, then the DO BLOCK iterates; otherwise, the client continues to the next step: transferring the data to the client.

4. The client is ready to receive a copy of the file that the batch program

created. First it establishes a remote SAS session on the server with the SIGNON RLINK statement.

- 4.1 This data step initializes the flag to 0.

- 4.2 The remote session issues a library reference to the SAS library that the batch program stored the summarized data.

PROC DOWNLOAD instructs the server to copy the summary data to the client platform. After the client receives the data, it terminates the remote session. The client is now ready to process the summarized data.

5. After transferring the data, the original code created an external file from the SAS data set. The condensed code does not show this step. After creating the external file the next step is to load it into a Paradox for Windows table. This section of code invokes a Paradox script file. This script loads the external file data into Paradox tables used in a network application.

The XSYNC option tells the SAS System to wait for the Paradox application to quit before SAS itself continues. After the Paradox script finishes executing, the client SAS session resumes. In this case the client reaches the end of %DRIVER so the client SAS session terminates.

6. This macro invocation starts the client process.

Limitations

Although the SAS System's capabilities made this type of client/server application easy to implement, using SAS does have drawbacks. First, we selected Paradox as the GUI because it is less expensive. Licensing 40 copies of the SAS System simply to use it as a GUI was not a viable option. Since the company had standardized on Paradox for Windows as its GUI, this choice made sense.

Another concern is system interruptions that occur when using SAS/CONNECT. When a system interruption occurs, SAS/CONNECT displays one of two "break windows." You must manually respond to these windows by selecting an option. SAS/CONNECT does not let you write programming statements that deal with these error conditions. This means that you can develop automated routines such as the one above, but your application is susceptible to interruptions.

Another deficiency is the rudimentary script language that SAS/CONNECT uses. The

Institute could increase its flexibility by permitting scripts to access DATA steps and the SAS System's macro facility.

Advantages

In this situation, the advantages to using the SAS System outweighed the disadvantages.

The greatest advantage to using the SAS System is SAS/CONNECT. This software is relatively easy to use and provides a simple mechanism for establishing SAS sessions on a remote platform; it easily facilitated communications between the two sessions.

Another benefit of using SAS/CONNECT is its data transfer capabilities. The software easily handles translating and transferring SAS data sets from one platform to another.

Finally, since the SAS System resides on both platforms, there is no need for you to learn an entirely new programming language. For the most part, your experience with the SAS System applies to both platforms.

Conclusion

This paper presented a methodology for using the SAS System to implement a client/server application. It defined an application as "being" client/server when it uses multiple, interacting processes to accomplish a task. The paper presented the approach in pseudo code and then used DATA step and macro program statements and SAS/CONNECT features to implement the logic.

In the end, the paper described the drawbacks and advantages of using the SAS System (particularly SAS/CONNECT) to implement this type of application.

Acknowledgments

The author thanks Jeff Phillips and Scott Rompala, both of CSC Professional Services Group, for reviewing this document and providing valuable comments. Any errors belong solely to the author.

SAS and SAS/CONNECT are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.