Paper 053-2008

Exploit SAS® Enterprise BI Server to Manage Your Batch Scheduling Needs

Troy B. Wolfe, Qualex Consulting Services, Inc., Miami, Florida

ABSTRACT

As someone responsible for maintaining over 40 nightly batch processes created in SAS® Data Integration, the author has taken control and created a batch scheduler that utilizes basic SAS coding and the operating system's scheduler to manage the task. This paper will show how to implement such a scheduler that can be used in any SAS environment. The main purpose is to show how one can exploit the EBI architecture in SAS to manage a batch process scheduler through the SAS® Information Delivery Portal. Some of the advantages the author has found using this includes:

- Complete control over scheduling including dependence on other processes
- Ability to maintain and calculate many different statistics of interest including when and how long a process usually runs
- Ease of use through incorporation into the SAS Portal
- Complete integration into the SAS Metadata Repository including security
- Flexible reporting with estimated time remaining to run a particular process based on history

INTRODUCTION

The following is a discussion on an application that was written to overcome stability problems with scheduling processes to run overnight. For various reasons, the software that was being used to manage jobs was not performing as needed. It occurred to us that creating our own scheduler using SAS and the native operating system's scheduling features would not be a difficult task. What we didn't realize is how easy it was to actually implement this and utilize capabilities that were available to us in the SAS Enterprise BI Server environment. First, we will discuss the core scheduler which can be written and implemented in any SAS environment using the native operating system's scheduler. Then we will show how EBI can build off of that basic program to create an easily maintained application that can give easy access to useful reports.

CORE SCHEDULER

TABLES

The batch scheduler runs off of some very basic tables. These constitute the foundation of the scheduler. Once set up properly, the rest of the application could easily fall into place. The most foundational table is the BatchJobs table.

BATCHJOBS Table

The BatchJobs table contains the specific information needed to execute a job. Information about where the SAS code is stored, the time and frequency it should run, and what server to run it on is stored here. In addition, some status columns are included. This table becomes the basis for many of the reports. Figure 1 shows a detailed description of the table.

Figure 1
BatchJob Table Structure

Type	Column Name(s)	Purpose
Active Controls	Active	Turn “on” or “off” to enable/disable
	Wait_Flag	Set “on” to delay running
	Today_Flag	A Flag set at midnight every day
	todayStart	Actual time job will next run (if Today_Flag is “on”)
Identifying Information	Process_Name	Name of Process
	Full_Path_Name	Location of Code
	Repository	Repository in SAS Metadata Server for DI jobs
	Server_Name	Name of Server to run on
Timing Details	Start_Time	Time it should not run before
	Daily_Flag	Set to “on” if job should run every day
	Sun Through Sat Flags (seven columns)	Set individual days to “on” to control what days job should run (ignored if Daily_Flag is set to “on”)
	Monthly_Day_Number	Set a specific day of month to run (ignored if Daily_Flag is set to “on”)
Status	Status	Current status (blank, “Running” or “Complete”)
	Status_Details	Current details (i.e. when it started, “Success”, “Error”, or “Warning”)
	Last_Successful_Complete	Last successful complete date and time
	Normal_Run_Time	The median run time considering all runs
	Run_Time	Current or most recent run time

BATCHJOBDEPENDENCIES Table

In order to have a flexible application that could have any number of predecessors required to complete successfully before running any job, the BatchJobDependencies table was created. This table only has two columns, PROCESS_NAME and PREDECESSOR. If at least one row exists in this table for any Process_Name, the scheduler will not run that process until all predecessors have successfully completed.

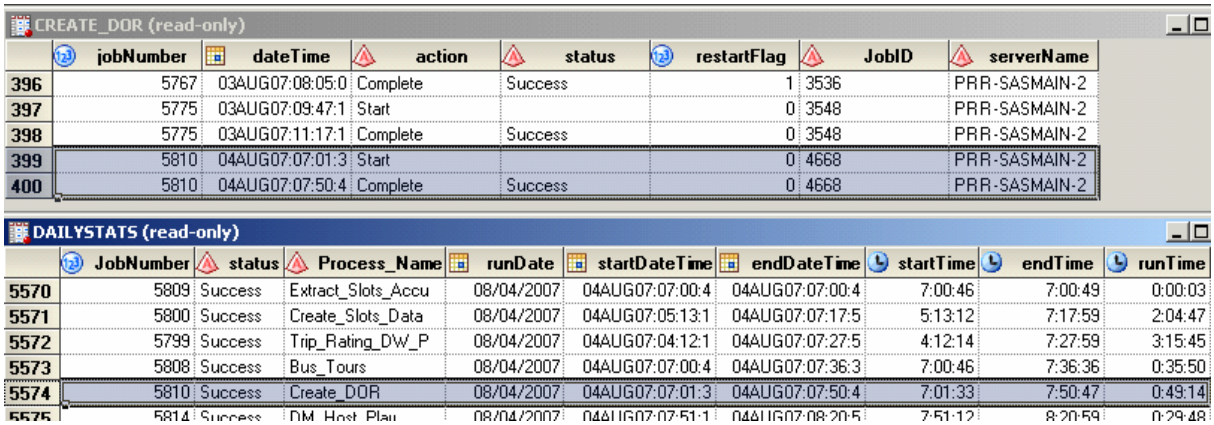
JOB STATUS Tables

In order to retain detailed information without creating one large table to query and to avoid any problems related to multiple processes updating one table, each process automatically creates its own table to hold historical facts about every time it was started and completed. These tables are updated with a macro that is called at the beginning and end of each job. Optionally, a process could update this table several times within a particular job and post status messages to report where exactly it is in the process. These are all stored in a single JOBSTAT library.

DAILYSTATS Table

One last important table is the DailyStats table. This table gathers the start and stop times for each job that has a table in the JOBSTAT library. At the completion of each job, a single row is added to summarize that job and the actual run time is calculated. Figure 2 shows the relationship of the Job Status table for the “Create_DOR” job with the DAILYSTATS table. Two detail rows in CREATE_DOR for JobNumber 5810 are merged together to form one row in the DAILYSTATS table so statistics can be easily computed in the reporting environment.

Figure 2
Converting a completed job into a daily statistic



CREATE_DOR (read-only)

	jobNumber	dateTime	action	status	restartFlag	JobID	serverName
396	5767	03AUG07:08:05:0	Complete	Success	1	3536	PRR-SASMAIN-2
397	5775	03AUG07:09:47:1	Start		0	3548	PRR-SASMAIN-2
398	5775	03AUG07:11:17:1	Complete	Success	0	3548	PRR-SASMAIN-2
399	5810	04AUG07:07:01:3	Start		0	4668	PRR-SASMAIN-2
400	5810	04AUG07:07:50:4	Complete	Success	0	4668	PRR-SASMAIN-2

DAILYSTATS (read-only)

	JobNumber	status	Process_Name	runDate	startDateTime	endDateTime	startTime	endTime	runTime
5570	5809	Success	Extract_Slots_Accu	08/04/2007	04AUG07:07:00:4	04AUG07:07:00:4	7:00:46	7:00:49	0:00:03
5571	5800	Success	Create_Slots_Data	08/04/2007	04AUG07:05:13:1	04AUG07:07:17:5	5:13:12	7:17:59	2:04:47
5572	5799	Success	Trip_Rating_DW_P	08/04/2007	04AUG07:04:12:1	04AUG07:07:27:5	4:12:14	7:27:59	3:15:45
5573	5808	Success	Bus_Tours	08/04/2007	04AUG07:07:00:4	04AUG07:07:36:3	7:00:46	7:36:36	0:35:50
5574	5810	Success	Create_DOR	08/04/2007	04AUG07:07:01:3	04AUG07:07:50:4	7:01:33	7:50:47	0:49:14
5575	5814	Success	DM_Host_Play	08/04/2007	04AUG07:07:51:1	04AUG07:08:20:5	7:51:12	8:20:59	0:29:48

MASTER MACRO

With the foundational tables in place, code could be written to kick off jobs at predetermined times, update status information, and recalculate statistics. In this case, there is one main macro that was written to facilitate these actions. The main purpose of this macro is to be called from the operating system scheduler (at midnight) and loop through every 30 seconds to re-update job statuses and kick off ready jobs. This macro can be broken up into three main pieces:

- Updating the status columns in the BatchJobs table,
- Determining jobs that are ready to run, and
- Executing those jobs and sleeping before looping back again.

UPDATING STATUS COLUMNS

First, the jobs that are currently marked as “Running” or “Submitted” need to be assessed as to their current status, since a job that was running may have completed and therefore will affect any jobs that are dependent on its successful completion. If a job is considered “Running”, there are two items that need to be checked.

First, each job considered “Running” must have its status table checked to see if it has completed. If it has completed, the job would have appended a final row with a status of “Complete” (as row 400 in the CREATE_DOR table in Figure 2 shows). Any job marked complete during this process will have its detailed information summarized and stored in the DAILYSTATS table as described in Figure 2 above. Then the normal run time will be recalculated by finding the median run time for this job for all successful runs recorded in the DAILYSTATS as shown in Figure 3. Jobs deemed complete will be updated in the BatchJobs along with its newly calculated normal run time.

Figure 3
Code used to calculate a new normal run time

```
proc summary data=JobStat.DailyStats
     (where=(status="Success" and
             Process_Name="&&processName&i"))nway noprint;
  var runTime;
  output out=processSummary(drop=_type_ _freq_) median=;
run;
```

Next, jobs that are deemed to be still running after checking their own status tables need to be verified that, in fact, they are *actually* still running, since a severe error may prevent the job from updating its own status table. To accomplish this, a JobID was captured using an operating system call when the job was kicked off and stored in the jobs own status table, so that it can be used to verify it is still running and kill it (which is discussed under Administering the Scheduler) if that is necessary. In Figure 2 above, notice that in CREATE_DOR, jobNumber 5810, which is a unique sequential number assigned by this application to each job kicked off, has a JobID of 4668. The 4668 number is the operating system Job ID assigned to this process and is the value used to query and or kill a job. More detail on obtaining this ID is discussed in the section on Executing Jobs.

Figure 4 shows how to query to see if a job is currently running. The filename statement uses the keyword “pipe” to execute an operating system command and pipe the output of that command to a filename, in this instance, called “tasklist”. Then the output is examined in a datastep to see if the key string “INFO:” is found. In Windows® 2003 Server, for example, a job that is not running will give this output: “INFO: No tasks are running which match the specified criteria.” On the contrary, if the job is still running, the output will detail information about that process and therefore will be interpreted by the code in Figure 4 as running and set the macro variable “jobComplete” to “N”. If a job that appears to be running, but is not found using the Job ID, the BatchJobs table will be updated in the following way:

STATUS: Complete
STATUS_DETAILS: Warning: Job Abended

Figure 4
Code used to check on running jobs

```
/* This line likely will need to be changed to an appropriate      */
/* command for an operating system different than Windows Server   */
/* 2003.                                                           */
filename tasklist pipe "tasklist /fi ""pid eq &jobID""";

data _null_;
  infile tasklist;
  length readLine $5;
  input readLine $;
  /* In Windows, a jobID that no longer exists will put out a      */
  /* message that starts with "INFO:", otherwise we can assume it  */
  /* is still running.                                             */
  if readLine eq "INFO:" then
    call symput("jobComplete","Y");
  else
    call symput("jobComplete","N");
  stop;
run;

filename tasklist clear;
```

However, if the job is in fact still running, the BatchJobs table will be updated in the following way:

STATUS: Running
STATUS_DETAILS: Started At 00:00 on 00/00/0000
RUN_TIME: {Calculated current datetime – start datetime}

DETERMINING JOBS READY TO RUN

Once the current status of things has been determined, the next section of code determines what now can be run. This involves two steps. First, all jobs that have not started or completed (or in other words, their STATUS column is empty), are set to run today (by virtue of the TODAY_FLAG set to 1), and the datetime value in the TODAYSTART column has passed (i.e. TODAYSTART less than current datetime value) are identified. Second, the BATCHJOBS table is merged with the BATCHJOBSDEPENDENCY table to get a list of all jobs that are running, have not started, or unsuccessfully completed that are listed as a predecessor to some job. Finally, these two tables are joined and any job from the first table that has a corresponding row in the second is deemed to be unable to run, since a required predecessor has not yet completed successfully for the days runs. Those remaining jobs that pass the predecessor test are then considered ready to run.

EXECUTING JOBS

The final stage of the master macro is to actually kick of any jobs that are ready to run. To do this, another call to the operating system is made using the X command as shown in Figure 5.

Figure 5
Code used to execute jobs ready to run

```
options noxwait noxsync;

data _null_;
  xcommand = "'"!!"{path}\sasBatchProcessSubmit.bat &&processName&i &&process&i &JobStat_JobNumber"!!"'";
  call symput("xcommand",trim(left(xcommand)));
run;

/* Run batch job                                                    */
x &xcommand;
```

First, note that the SAS system options NOXWAIT and NOXSYNC must be set in order for the xcommand to run the operating system command without requiring user interaction and allowing control to be returned immediately to SAS after executing the command. Also not that the X command is specific to Windows operating systems, a different operating system may require a different SAS command to run an operating system command. See SAS documentation for the operating system to determine the appropriate command.

The command submitted to the operating system is a batch file that was created for this purpose. The three macro variables passed in are parameters expected by the batch file (&&processName&i = process name, &&process&i = full path name for code, &JobStat_JobNumber = unique job run identifier). Figure 6 shows the content of the sasBatchProcessSubmit.bat file.

Figure 6
sasBatchProcessSubmit.bat file

```
①set JobStatNumber=%3

call②D:\SAS\SASSolutionsConfig\Lev1\SASMain\BatchServer\sasbatch
    ③-log D:\SAS\SASSolutionsConfig\Lev1\SASMain\BatchServer\logs\%1_#Y.#m.#d_#H.#M.#s.log
    -batch -noterminal -logparm "rollover=session"④-set SchedName %1⑤-sysin %2
```

In a batch file, %# allows one to access the parameters passed to the batch file. Therefore, %1 refers to the process name, %2 refers to the full path name of the source SAS code, and %3 refers to the unique job number assigned by this application. A brief discussion of each number identified in Figure 6 is as follows:

1. The JobStatNumber is set to an environment variable so the called program can access that information to be stored in its own status table.
2. This is a SAS supplied batch program that can be used to start a batch SAS program
3. The log file uses the process name concatenated with datetime variables
4. The process name is set to an environment variable “SchedName” so the called program can access that information and use it as the name of its own status table
5. SYSIN is the SAS option that will run the specified full path name for the source SAS program

One final piece to understand in the execution of a job is that each job must have a reference to the updateProcessStatus macro at the beginning and end of the code. The updateProcessStatus will append a new row (or create a new table with one row if the process table does not yet exist) for each time it is called to a table by the name of the process name (note the process name should be a SAS name—that is to say it should have

no spaces, use characters, numbers, or underscores, and not start with a number). Before the final call to the macro, the following options statement is submitted:

```
options nosyntaxcheck obs=max;
```

These two options will allow the macro to execute even if an error is encountered in the process and the syntaxcheck option gets turned on (i.e. puts execution into debug mode) and the obs is set to zero, both of which are default behaviors when an error is encountered in SAS.

RUNNING THE SCHEDULER

Once the job is in place, it must be scheduled to run every night at midnight. To avoid multiple schedulers running at one time, the SAS code was written to terminate after all jobs have run for the day. If by chance a job is still running, the scheduler will terminate itself after 11:59:00 pm and restart again at midnight. At midnight, a process, called startBatchScheduler.sas, is run by the scheduler that resets all the status columns (except any jobs currently running), identifies jobs that should run that day, and updates the Today_Flag and TodayTime columns (including jobs that are still running). Then this process calls the main scheduler macro which will loop through all day long until all jobs scheduled for that day are complete.

For the purpose of this paper, the Microsoft Windows scheduler will be used as an example. Before actually scheduling anything, a batch file, startBatchScheduler.bat, was created that would simply invoke startBatchScheduler.sas. This file is scheduled to run under a User ID that is allowed to run batch processes on the server. In this example, the User ID is “sasbatch”. The code in this batch file is shown in Figure 7. To understand the syntax, see the discussion of Figure 6 above.

Figure 7
startBatchScheduler.bat file

```
call D:\SAS\SASSolutionsConfig\Lev1\SASMain\BatchServer\sasbatch -log
     D:\SAS\SASSolutionsConfig\Lev1\SASMain\BatchServer\logs\sasBatchScheduler_#Y.#m.#d.log
     -batch -noterminal -logparm "rollover=session"
     -sysin D:\CRM\DataStores\Warehouse\ETLProcesses\startBatchScheduler.sas
```

UTILIZING SAS ENTERPRISE BI

Up to this point it has been very basic SAS programming and now the scheduler will start at midnight, stop just before midnight, and run all jobs as configured in the main scheduler tables. As stated earlier, the purpose of this paper was not just to show a basic application, but to show how SAS Enterprise BI can be utilized to run batch jobs on different servers, show job status and statistics in an easily readable format that is available across the enterprise while applying security constraints, and allow it to be maintained through the Portal.

RUNNING JOBS ON DIFFERENT SERVERS

With the availability of a SAS/Connect® server in SAS Enterprise BI, one can easily code the scheduler to run any job on any available SAS server. Let us take a look at what simple changes can be made to the code to allow this to happen. In this example it is assumed a SAS/Connect Server is set up and running on a server separate from the main Data-Tier server.

There are two points in the main code that need to be adjusted to make this work. First, obviously when the batch job that needs to run on another server is kicked off, code must be put in place to make a connection from the server the scheduler is running on and the server that is going to run the job. Then the job can be run on the remote server. The code to do that is shown in Figure 8.

Figure 8
Code used to start a process on a remote server

```
%if &&processServer&i ne Local %then
%do;  /* If the process Server is not "Local" then remote submit */

  /* Retrieve UserID and password stored in a SAS stored program */
  data pgm=dw.sasbatchpass;
  run;

  /* Set options to sign on and sign into remote host            */
  %let fmhost=prr-sasfm-2k3;
  options comamid=tcp;
  signon fmhost.7551 user="&userid" password="&password";

    /* Assign variables from local host to remote host          */
    %syslput i=&i;
    %syslput processName&i=&&processName&i;
    %syslput process&i=&&process&i;
    %syslput JobStat_JobNumber=&JobStat_JobNumber;

    /* Submit the code on the named remote server               */
    rsubmit FMhost.7551;
      ...
    endrsubmit; /* End remote submit                           */
  signoff fmhost.7551;
%end; /* End process server is not "Local"                     */
```

The second point in the code that must be considered is when the “running” jobs are verified. The same code is used as in Figure 8, however, as you may notice as shown in Figure 4, there is a macro variable, jobComplete, that is returned. This variable must be passed back to the local server to tell the scheduler if the code is in fact running. To do this, the following code needs to be used just prior to the “endrsubmit” command:

```
%nrstr(%sysrput jobComplete=&jobComplete);
```

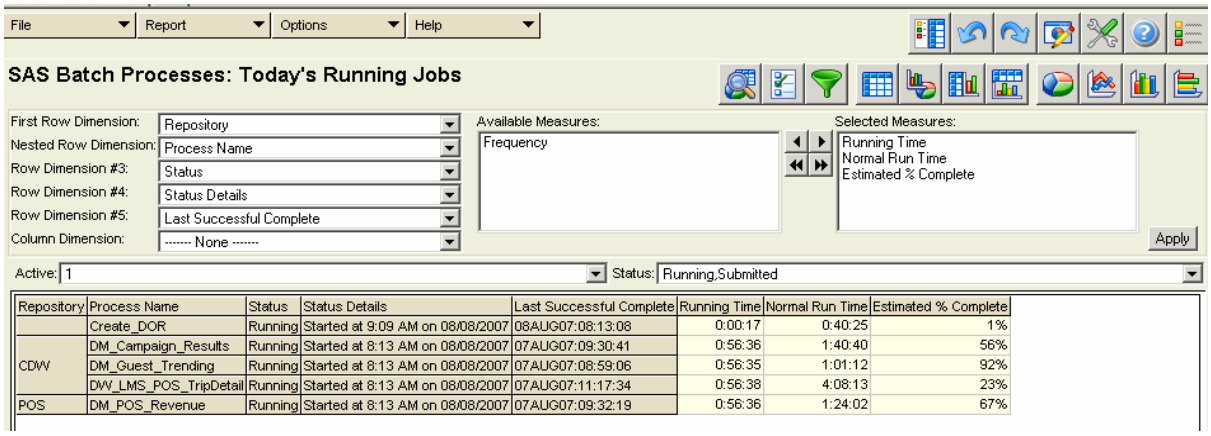
The final configuration consideration is that the remote server needs to have access to the libraries that contain the individual Job Status tables. In this case, windows share directories are employed.

REPORTING

An important piece to this process is the ability to view the status of the scheduled jobs and appropriate statistics. Obviously, since the status is simply in SAS tables, queries can be written in Enterprise Guide to get the information. A more interesting approach and an easier way to disseminate to others, if desired, would be to use the power of the SAS Information Delivery Portal.

Reports could be written using SAS® Web Report Studio or viewed through SAS® Web Report Viewer. In this case, an OLAP reporting tool, Futrix®, was chosen. Figure 9 shows a view of a report that shows jobs that are currently running.

Figure 9
Screen shot of web report showing currently running jobs



Some very useful pieces of information that this report shows include when a job started, how long the process has been running, what the normal run time is (which is simply applying the median statistic of all successful runs), and, based on that information, approximately how much of that process is complete. A complete list of all reports that are currently defined with a description is shown in Table 2.

Table 2
Defined Reports

Report Title	Report Description
Daily Statistics	Shows statistics pertaining to each defined batch job including last successful completion, normal run times, etc. Plus ability to drill into each job and view statistics for individual runs (See Figure 10).
SAS Batch List With Predecessors	Quick view of each defined batch job and the jobs required to complete before it will run.
Today's Completed Jobs	What jobs completed for today including when and the status of that job on completion.
Today's Jobs Not Yet Started (W/ Predecessors)	A list of jobs and that haven't started running yet and the status of all of its predecessors.
Today's Jobs Not Yet Started	A smaller list of jobs that have not yet run (easier to read without predecessors).
Today's Running Jobs	List of jobs that are currently running (See Figure 9).

One report of interest is the Daily Statistics view. This particular report allows a user to drill into a specific job and view statistics for its individual runs. See Figure 10 for a view of this report. In this figure, the Daily Statistics report is shown at the top. Below, it can be seen that double-clicking on "Create_DOR" will bring up a list of dimensions that can be drilled into from Process Name (the name of the column "Create_DOR" occupies). Selecting "Job Run Number" will result in the bottom right view where the report will list all runs, by "Job Run Number", for Process Name "Create_DOR". This report is very useful to see at a glance where the longest running jobs are and how long they generally run. This report can be sorted by any one of the available columns. Also, jobs that seem to be taking a long time can be further investigated by looking at particular Job Run Numbers. From there, one can create a link to a log associated with that particular run (assuming at least some of the historical logs are saved out somewhere). Also, a drill-through feature will allow a user to even view the base table this report is based off of showing all details of the report.

Figure 10
Web report showing Daily Statistics

Process Name	Latest Run Date	Normal Run Time	Normal Start Time	Normal Complete Time
Bus_Tours	08/09/2007	0:36:46	7:00 AM	7:38 AM
Clear_Log_Files	08/08/2007	0:00:05	11:00 PM	11:00 PM
Comp_Point_Balances_ODS_DW_Process	08/09/2007	0:03:24	12:00 AM	12:03 AM
Comp_Point_Detail_ODS_DW_Process	08/09/2007	0:39:17	12:00 AM	12:39 AM
Create_DOR	08/09/2007	0:40:46	7:17 AM	8:13 AM

Drill "Create_DOR" to "Job Run Number"

Process Name	Latest Run Date	Normal R
Bus_Tours	08/09/2007	
Clear_Log_Files	08/08/2007	
Comp_Point_Balances_ODS_DW_Process	08/09/2007	
Comp_Point_Detail_ODS_DW_Process	08/09/2007	
Create_DOR		
Create_Slots_Datamart		
Customer_Address_ODS_DW_Proc		
Customer_ODS_DW_Process		
DM_BounceBack		
DM_Campaign_Results		

Process Information
Process Name
Status
Job Run Number
Start Date Time
End Date Time

Process Name: Create_DOR

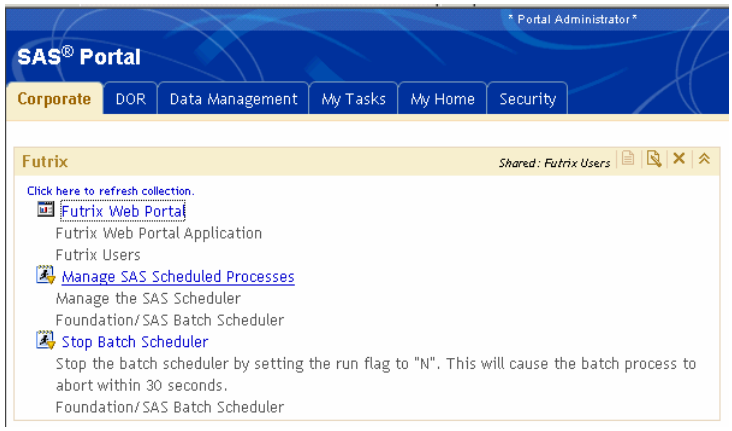
Job Run Number	Latest Run Date	Normal Run Time	Normal Start Time	Normal Complete Time
528	03/22/2007	2:07:31	11:40 AM	1:47 PM
650	03/26/2007	0:28:32	7:01 AM	7:29 AM
663	03/26/2007	1:56:01	1:54 PM	3:50 PM
688	03/27/2007	0:30:35	7:01 AM	7:31 AM

INFORMATION DELIVERY PORTAL

The reporting tools mentioned above are all available via the SAS Information Delivery Portal. Not only can the reports be placed in the Portal, but they can also have security applied to them that only allows to access those who need them. In this case, it not only applies to allowing users access to Futrix, but even to individual reports within Futrix. For instance, it may be that you want to allow a user access to see the status of a few jobs relevant to their duties. In this case, this user can be given access to the Futrix reporting environment via the SAS Metadata used by the Portal, and then within Futrix, a user can be granted access to only information about a list of particular jobs. In this way, they are able to determine the status of their own jobs without being mired down with all the details of every other job out there.

Figure 11 shows a Web Portal page where Futrix is enabled and a couple administration interfaces are enabled for the user who has signed in. The visibility of each of the three components, "Futrix Web Portal", "Manage SAS Scheduled Processes", and "Stop Batch Scheduler", can be controlled by an administrator. Thus, a user who is given the ability to view a report, but not administer the scheduler, will not see the bottom two administrative components. A user who has no rights to view any, will not even see this "Futrix" portlet (a portlet is just a component that contains web content that can be hidden or viewed depending on access rights). Only those users who are part of the "Futrix Users" group in the SAS Metadata Server will be able to see this portlet.

Figure11
Information Delivery Portal screen shot



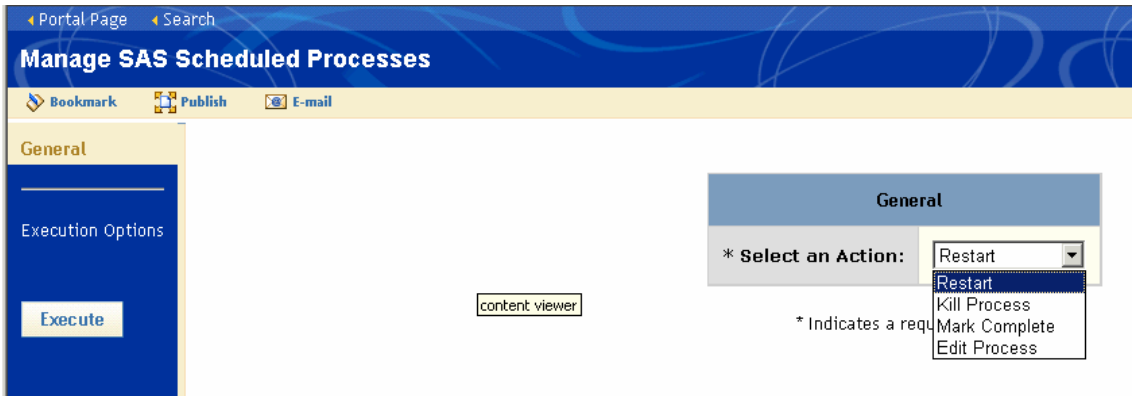
ADMINISTERING THE SCHEDULER

The final aspect of the scheduler, as alluded to in the previous section, is the ability to administer it. Up until this point, one could restart, mark complete, or update a job by updating the table directly. Also, a job could be killed by ascertaining the operating system Job ID and issuing a command at the command line to kill it. To make this easier and more robust, interfaces can easily be created using Stored Processes to administer the scheduler

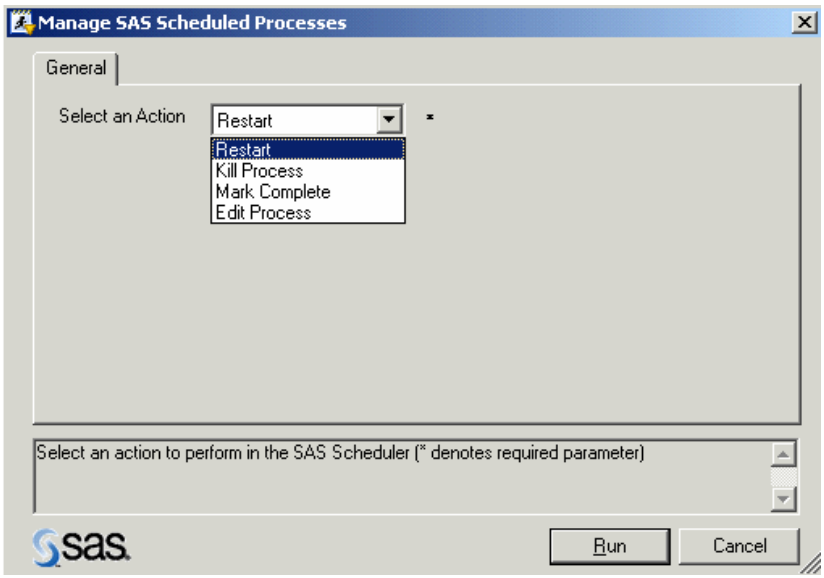
through the Portal or SAS Enterprise Guide. A Stored Process is simply a SAS program that can be accessed from various sources. In creating a stored process, metadata is generated about the SAS code with regards to what inputs it expects, where it is stored, and information of the like. If inputs are expected, interfaces that are designed to handle stored processes will automatically present the user with prompts to gather the necessary data. Figure 12 compares the Portal interface with the SAS Enterprise Guide interface. In this case, selecting

Figure12
Information Delivery Portal screen shot

Information Delivery Portal Stored Process Parameter Interface



Enterprise Guide Stored Process Parameter Interface

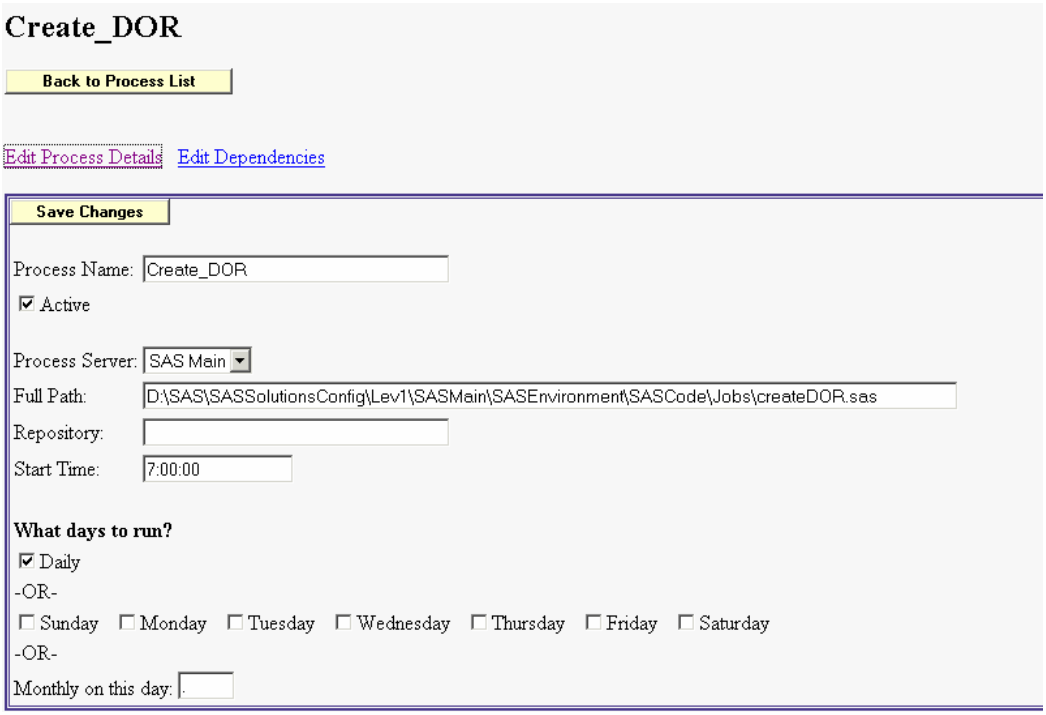


one of the first three options will trigger another stored process that will ask for what process to Restart, Kill, or Mark Complete. The last “Edit Process”, will call a different Stored Process that requests the user to choose a process to edit and it can be edited as can be seen in Figure 13 below. The easiest way to create a stored process is to use SAS Enterprise Guide. From File -> New -> Stored Process, one will be guided through a wizard to create a stored process.

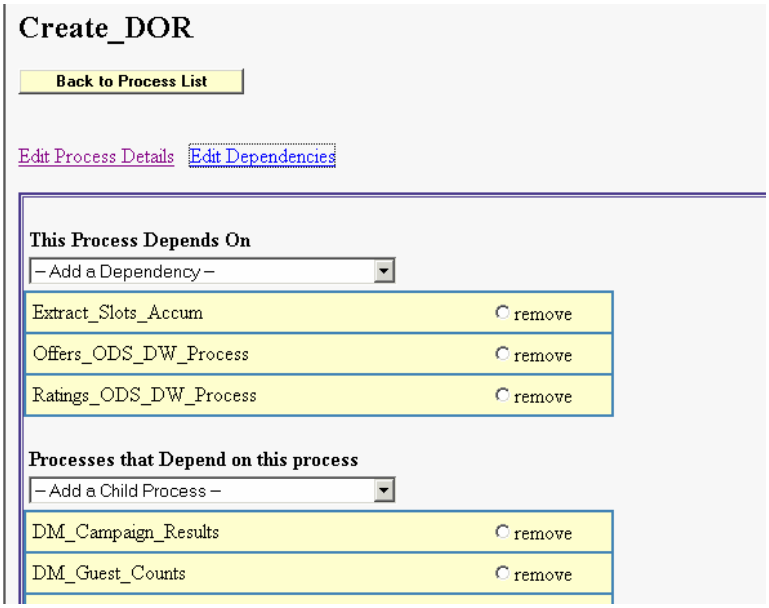
One final item to look at is how one stored process calls another. Figure 14 below shows the screen that is displayed when “Restart”, “Kill Process”, or “Mark Complete” is called. Note that this just appears to be a regular web screen. In fact, the code behind this creates an HTML document to be rendered by a browser.

Figure13
Edit a Process in the Portal

Edit Process Details



Edit Dependencies



Let us take a quick look at the code that was run by the first screen to create this screen when "Restart" was selected. First, the selection made on screen one is stored into a global macro variable so we can ascertain which of the first three options were selected.

```
%if &performAction eq Mark %then
  %let whereClause = status ne "Complete"
and status ne "Killed";
%if &performAction eq Restart %then
  %let whereClause = status ne "";
%if &performAction eq Kill %then
  %let whereClause = status eq "Running";
```

Figure14
Restart HTML screen

The action that is selected will set a macro variable, called whereClause, so that the BatchJob table can be queried to present only jobs that make sense for that action. Next, The beginning of the HTML code needs to be created and sent to the "_webout" destination, which is an automatic output to a web interface. The contents are written to a buffer and sent when the code is complete.

```
data _null_;
  file _webout;
  put '<HTML>';
  put '<BODY BGColor="#f5f5f5">';
  put "<Center><H1>Select a Process to &performAction</H1>";
```

Next, special code is needed to point back to the stored process engine and the stored process that will be executed is identified.

```
  /* Create reference back to the Stored Process
     Web Application from special automatic
     macro variable _URL. */
  put "<FORM ACTION='&_URL'>";

  /* Specify the stored process to be executed using
     the _PROGRAM variable. */
  put '<INPUT TYPE="hidden" NAME="_program" VALUE="/SAS Batch Scheduler/Manage a Process">';
```

Then name/value pairs that need to be passed to the next process are defined. These name/value pairs are referenced from the receiving process as global macro variables.

```
  put '<INPUT TYPE="hidden" NAME="performAction" VALUE="' "&performAction" '"><BR>';
```

Now the screen can be built using standard HTML code.

```
  put '<INPUT TYPE="button" VALUE="Cancel" onClick="javascript:window.close();">';
  /*put '<INPUT TYPE="submit" VALUE="Submit"><BR><BR>';*/
  put '<BR><BR>';

  /* Pass first name value on to next program.
     The value is user entered text, so you must
     encode it for use in HTML.  */
  /*processName = htmlencode("&processName", 'amp lt gt quot');*/
  put '<SELECT NAME="processName" onChange="submit()">';
  put '<OPTION>(Select A Process)';
run;
```

Since this process needs to populate a list, a separate data step is used to get that data and add it to the SELECT list via the OPTION tag. Note that this is still writing to the _webout file and is subsetting the data.

```
data _null_;
  file _webout;
  set DW.BatchJobs;
  where (&whereClause) and Today_Flag;
```

```
    put '<OPTION Value="' Process_Name '">' Process_Name ' - (' status ')';
  run;
```

Finally, the HTML code is closed with the ending tags. And the entire contents of _webout is passed back for the browser to display. Thus, the screen in Figure 14 is rendered.

```
  data _null_;
    file _webout;
    put '</SELECT><BR>';
    put '</FORM></Center>';
    put '</BODY>';
    put '</HTML>';
  run;
```

CONCLUSION

This simple application demonstrates how easy it is to take a core SAS program and open it up to the enterprise with security and ease of use in mind. Statistical information previously unavailable to us in other software is readily available, not only in reports, but also for easy analysis since it resides in SAS datasets. In addition, since the code is easily accessible, and the core of it is not complex, any customizations in the future are a breeze. The basic application itself (excluding the administration screens and ability to run on remote servers) took only a couple weeks to write and implement. While there are many software packages that could be used to meet ones scheduling needs, the above approach has brought simplicity and stability in scheduling our overnight batch jobs.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name	Troy B. Wolfe
Company	Qualex Consulting Services, Inc.
Address	423 Castle St.
City State ZIP	Geneva, NY 14456
Work Phone:	(315) 781-0243
Email:	troy.wolfe@qlx.com
Web:	www.qlx.com

SAS®

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

WINDOWS®

Windows is a registered trademark of Microsoft Corporation in the United States and other countries.

FUTRIX®

Futrix is a registered trademark of Futrix Software Limited in the USA and other countries. ® indicates USA registration.