How to Use SAS/IntrNet® and Client-Side Functionality to Automate Your SAS® Processing
James L. Ribble, Centers for Disease Control & Prevention; William P. Bartoli, Northrop Grumman Information Technology

ABSTRACT
This paper outlines a method of combining SAS/IntrNet tasks and client-side JavaScript functions to create an application that can automatically detect new data arrivals and spawn additional background tasks to process the data and create multiple types of output for both online viewing and distribution. The system provides for the establishment of task dependencies which allow entire SAS batch-processing streams to be triggered and managed automatically without the need for user intervention.

INTRODUCTION
Over the years my current client had developed an ASP based system for the tracking and viewing of its analytic reports which were created by various SAS programs executed by hand. Originally it had been envisioned that this system would be expanded to facilitate the actual submission and management of the various SAS programs themselves. However, implementing this later functionality proved to be problematic as they attempted to resolve the many security issues associated with moving both programs and their various types of output back and forth between different applications located on multiple servers. As a result of organizational changes this system was eventually left unsupported and unusable.

Almost simultaneously the SAS programming staff was tasked with developing a mechanism by which our data would be processed automatically upon arrival rather than waiting for the manual execution of the SAS programs by the programming staff.

Needing to replace the previous system while also meeting the new turn-around requirement with no additional head count it was decided to attempt to engineer as close to an all-SAS solution as possible. This would avoid the past security issues by localizing the processing to the single server where SAS resided as well as allow the system to be grown and maintained by our SAS programming staff.

The task of engineering a solution to this challenge fell to me.

BACKGROUND
In our environment data files are submitted to us for processing monthly from multiple states and territories. When these data files arrive they are deposited in a central directory from which a sequential series of SAS jobs is run against them. These jobs create new data files and analytical reports which are then distributed back to the field. Other reports are created and maintained for analysis by our own staff. It is important for the programmers and analysts to be aware of the current status of both the incoming and outgoing data files as well as the report production. I had already engineered a SAS/Intrnet application with which we could view our reports online as well as track the status of our data files and their processing. However the SAS programs to feed these displays were all still being run manually by our programming staff.

The new application I was about to engineer to execute these tasks automatically would come to be known as the Task Manager.

THE BASIC TOOLS
Often the answers we need can be found in what seems to be a mistake or by dogged experimentation. The solution I needed came as I explored the possibilities of the SAS/Intrnet product.

SAS/INTRNET BASICS
SAS/Intrnet tasks can in their basic form be executed just as if they were a web page by entering a specifically constructed URL in the user’s web browser window. This URL contains, in part, the SAS/Intrnet web server location, the name of the SAS program to be executed and any parameters being passed to the program. The requested SAS program is then executed and its output spooled back to the user’s browser. Whereas the SAS logic is executed on the server the program’s output is subsequently rendered by the user’s web browser. In addition to HTML markup
THE NEVER ENDING TASK

One day I feared a program I was testing might have entered a never ending loop and so I closed the program’s browser window. After making a correction to my logic I invoked the program again to test it. I was surprised to see this program seemingly sit and do nothing for more than a minute before it appeared to actually execute and generate output to be displayed in my browser window. After further investigation I was even more surprised to find that the second execution of my program appeared to have been waiting on the first execution to complete. The SAS program I had thought to have been terminated when I closed the browser window associated with it had apparently kept on running! It was time to hit the books. I found the explanation in my SAS/Intrnet documentation in the form of a warning that closing a browser session does not necessarily halt the execution of the SAS task associated with it. Apparently my termination of the application’s browser session had left its SAS program running invisibly on the server! This was just the sort of unattended processing I was looking to perform. But how could I exploit this feature?

LETTING THE CLIENT DO THE WORK

The answer to this question lay in HTML and the client-side scripting capabilities incorporate in present-day web browsers such as Internet Explorer. Javascript was my scripting language of choice due to its broad support by web browsers in addition to its not requiring any server-side components. To achieve my objective it would be necessary to launch a SAS/Intrnet application in a new browser window and then simulate the closing of that window by the user.

To make the process self-perpetuating I would also need a mechanism by which processing could be reinitiated on a regular basis without the need of human intervention.

I subsequently identified three simple client-side constructs that would give me the functionality I needed. The first was a Javascript statement which in its most basic form looks like this:

    WINDOW.OPEN(url)

This statement will open the specified URL in a new browser window. Although there are additional parameters which can be used to tailor the new window's appearance they are optional and will not be covered here. What is important to note here is that the URL points to a SAS program that we wish to execute from our SAS/Intrnet program library.

The second construct was also a Javascript statement which looks like this:

    WINDOW.CLOSE()

By imbedding this statement within the browser output from a SAS/Intrnet program the browser will actually close its window immediately just as if the user had done it with their mouse. This then leaves the SAS program running invisibly in the background. (Note that due to built in security restrictions your browser will not allow Javascript to close the user’s primary browser window but only those additional browser windows opened using the WINDOW.OPEN() construct.)

Since the user’s browser renders the markup sequentially as it is received from the SAS program having the program imbed these Javascript statements within its output stream will case them to be executed immediately upon receipt by the browser. Here is a simplified example of the correct syntax for placing these Javascript statements within an HTML document:

```
<HTML>
  <HEAD></HEAD>
  <BODY>
    <SCRIPT LANGUAGE="Javascript">
      WINDOW.CLOSE()
    </SCRIPT>
  </BODY>
</HTML>
```

The third item that was needed was an HTML Meta tag:

```
<META HTTP-EQUIV=REFRESH CONTENT=30>
```
This statement goes within the <HEAD> section of an HTML document. The ‘30’ in this example tells the browser it should refresh the current page every 30 seconds. This is just like having the user click on their browser’s refresh button. When the page is refreshed the URL is once again accessed and the SAS program referenced by that URL is executed.

Although there are multiple ways in SAS to insert these statements into our output here is a basic Data Step example of how a web page might be written out with the <META> tag and WINDOW.CLOSE() statements inserted within it:

```sas
DATA _NULL_;
  FILE _WEBOUT;
  SET XYZZY END=EOF
  IF _N_=1 THEN PUT "<HTML>" /
    "<HEAD>" /
    "<META HTTP-EQUIV=REFRESH CONTENT=30>" /
    "</HEAD>" /
    "<BODY>" /
    "<SCRIPT LANGUAGE='javascript'>" /
    "WINDOW.CLOSE()" /
    "</SCRIPT>";
  /***
   PUT statements to create user report output go here ***/
  IF EOF THEN PUT "</BODY>" /
    "</HTML>";
RUN;
```

Since all of these statements need to be incorporated into the HTML markup sent back to the browser and it is our SAS programs which are actually generating this markup dynamically it should be a relatively simple matter to code SAS logic to insert these statements where and when they are needed in our display output.

**ASSEMBLING THE TASK MANAGER**

Now that we have the basic tools needed we must assemble them into a usable application. The overall flow of this process should look something like the diagram below.

A main browser session is kept constantly running which recursively executes a SAS program. This SAS program, the heart of the Task Manager, goes through several processing steps, determines task eligibility and then spawns the appropriate SAS programs in their own browser sessions as needed.

Let’s take a brief look at what each of these processing steps does.

The first step gathers data from a variety of sources. This data includes Windows directory information on newly arrived or updated files as well as data on the status of previously executed SAS tasks.

The data gathered in the first step is then evaluated to see if the criteria have been met for any of the tasks being managed by the system. If a task is found to be eligible for execution the SAS program writes the Javascript WINDOW.OPEN(url) statement to the user’s browser resulting in an additional browser window being opened and the SAS program referenced in the URL being executed in it. Imbedded near the top of the output from this spawned program is a WINDOW.CLOSE() statement which causes the browser window to close immediately and allows the subsequent SAS logic contained in the program to execute ‘in the background’.

After spawning the eligible tasks a display is written out showing the current status of the system. Once this is complete the SAS program is finished and execution stops. However, imbedded within the web page output from the SAS program is the Meta tag which causes the page to be refreshed. After the user defined time interval has passed the browser refreshes its display without intervention on the part of either the user or SAS/Intrnet. When this refresh occurs the SAS program which generated the page is re-executed and the entire series of steps is performed again.
from the beginning. Barring any external interruptions such as a power cut or termination of the SAS/Intrnet environment this process could be continued indefinitely without user intervention.

All of the spawned tasks contain a wrapper of SAS logic that includes some housekeeping routines. The very last of these routines marks a status or 'checkpoint' file indicating that the task has successfully completed. This updated status info is then evaluated during the next cycle of the main routine to determine if any downstream processes triggered by this task are eligible for processing. In those cases where a spawned task has a critical SAS error (resulting in SAS setting OBS=0) or for some other reason does not run successfully to completion the checkpoint file is never updated to indicate that the task has finished. When the main routine sees that a task has not completed within the expected length of time it will mark the task as once again eligible for execution and cause it to be spawned and executed again. As a safety measure once a task has been requeued for execution three times without successfully completing it is marked as ineligible for execution and notification of the problem is sent to the responsible programmer so that they may investigate the cause.

HANDLING INTER-TASK COMMUNICATIONS
Two files within the system are responsible for tracking task status and facilitating communication between the main Task Manager session and the spawned sessions.

THE EXECUTION QUEUE
When a task is determined to be eligible for execution an entry is placed in this queue. Tasks are then spawned and executed based on their priority and queuing order (FIFO). When a task is actually spawned its entry is updated to indicate that the task is currently executing. When the spawned task is done it removes its own entry from the execution queue to indicate its completion. The queue entries track all the information pertaining to a task such as its priority, who to notify when the task completes, the state and year the task is processing data for, how many times the task has been re-queued and whether or not its downstream dependant jobs should be triggered.

THE CHECKPOINT FILE
The checkpoint file tracks the spawned tasks’ completion history. The data we process at my site is gathered and processed monthly by state. When a task spawned by the Task Manager completes a binary flag for that task and state is set within the checkpoint file. These flags are then queried by the Task Manager’s main logic to ascertain what tasks have completed execution and hence what downstream tasks are eligible for execution.

SYSTEM CONTROLS
Although this overview of the process seems rather simple there are numerous tables and mechanisms which allow control by our staff over the number and type of tasks allowed to execute concurrently based on workload and environmental factors.

In most cases the execution-time controls for the system are maintained in Access or SQL Server tables. Below is an overview of the key tables and files used in the system.

WORKLOAD DEFINITIONS
When the SAS/Intrnet product launches a SAS program it assigns a port number to the session. The number of ports available to service user requests limits the number of tasks than can be initiated simultaneously. Tables within the system allow for the definition of workload groups containing one or more tasks with similar resource requirements and/or execution times. Using this information the Task Manager determines on each cycle the correct number and mix of tasks to spawn for execution based of the number of ports available for work.

TASK DEFINITIONS
A task definition table defines the tasks to be managed by the Task Manager. The principal task attributes defined in this table are:

- The task name.
- The task type. This can be either ‘Core’ or ‘Spawned’. Core tasks are execute as part of the Task Manager logic during each cycle rather than being spawned in a separate browser session. They serve the purpose of performing the initial checks to determine if new data files have arrived, etc. requiring that a processing task be spawned. Spawned tasks are those tasks that are actually launched by the Task Manager in a new browser window/session.
- The task level. ‘S’ for state or ‘U’ for U.S. This tells the system if the task is to be run each time a new state data file arrives or only when all the state files for the entire U.S. have arrived, respectively.
- The relative priority (execution order) for the task once is has been put into the execution queue.
- The logon ID of the user to notify when the task has completed execution, experiences problems, etc.
- The physical location of the source code for this task. If no location is specified the task is assumed to be a previously compiled macro by the same name as the task.
- The physical location (path) where the SAS log for this task should be written. If left blank the log for this task is not saved.
- The task’s display name. This text is used when referring to the task in online displays, etc.

TRIGGER DEFINITIONS
One of the key tables within the system is the trigger definition table which specifies the scheduling dependencies between the different tasks. One or more tasks can be set as a predecessor to another task. From this file a binary mask is created of each task’s prerequisites. These masks are then used against the values in the checkpoint file to ascertain if all of a task’s predecessors have executed.

THE TASK MANAGER DISPLAY
Information from all of the above sources are used by the Task Manager to construct a display showing information on the current environment, task status and event history.
Here is an example of the Task Manager display.

<table>
<thead>
<tr>
<th>--- MANAGER CONTROLS ---</th>
<th>--- WORKLOADS ---</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process cycle time (seconds)</td>
<td>60</td>
</tr>
<tr>
<td>Available ports (sessions)</td>
<td>3 [5012 5013 5014]</td>
</tr>
<tr>
<td>Year(s) being monitored</td>
<td>2005</td>
</tr>
<tr>
<td>Workload</td>
<td>Task Limit</td>
</tr>
<tr>
<td>ONLINE</td>
<td>1</td>
</tr>
<tr>
<td>SHORT BATCH</td>
<td>1</td>
</tr>
<tr>
<td>LONG BATCH</td>
<td>1</td>
</tr>
<tr>
<td>EXTENDED BATCH</td>
<td>0</td>
</tr>
</tbody>
</table>

--- EXECUTION QUEUE ---

<table>
<thead>
<tr>
<th>PRN</th>
<th>FRTY</th>
<th>STATUS</th>
<th>QUEUES</th>
<th>DURATION</th>
<th>ST</th>
<th>YEAR</th>
<th>SURVEY</th>
<th>NOTIFY</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRFSS_RESET</td>
<td>2</td>
<td>1</td>
<td>Executing</td>
<td>0</td>
<td>0:00:00</td>
<td>VA</td>
<td>2005</td>
<td>BRFSS</td>
</tr>
<tr>
<td>RISKDATA</td>
<td>3</td>
<td>3</td>
<td>Executing</td>
<td>0</td>
<td>0:00:00</td>
<td>IL</td>
<td>2005</td>
<td>BRFSS</td>
</tr>
<tr>
<td>RECVRDATENV</td>
<td>2</td>
<td>1</td>
<td>Queued</td>
<td>0</td>
<td>0:00:51</td>
<td>US</td>
<td>2005</td>
<td>BRFSS</td>
</tr>
<tr>
<td>DSUPDOT</td>
<td>2</td>
<td>2</td>
<td>Queued</td>
<td>0</td>
<td>0:00:52</td>
<td>IL</td>
<td>2005</td>
<td>BRFSS</td>
</tr>
</tbody>
</table>

--- SESSION LOG ---

<table>
<thead>
<tr>
<th>DATE</th>
<th>TIME</th>
<th>SESSION ID</th>
<th>TASK</th>
<th>STATE</th>
<th>YEAR</th>
<th>MESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/13/06 16:52:15</td>
<td>319e697d952</td>
<td>BRFSS_RESET</td>
<td>IL</td>
<td>2005</td>
<td>Ended</td>
<td></td>
</tr>
<tr>
<td>USC1097d952</td>
<td>QC_DATA</td>
<td>IL</td>
<td>2005</td>
<td>Ended</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:52:14</td>
<td>519e697d952</td>
<td>BRFSS_RESET</td>
<td>IL</td>
<td>2005</td>
<td>Started</td>
<td></td>
</tr>
<tr>
<td>USC1097d952</td>
<td>QC_DATA</td>
<td>IL</td>
<td>2005</td>
<td>Started</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:51:53</td>
<td>XgRTxf76d952</td>
<td>RISKDATA</td>
<td>IL</td>
<td>2005</td>
<td>Triggered</td>
<td></td>
</tr>
<tr>
<td>XgRTxf76d952</td>
<td>QC_DATA</td>
<td>IL</td>
<td>2005</td>
<td>Triggered</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16:51:45</td>
<td>319e697d952</td>
<td>BRFSS_RESET</td>
<td>IL</td>
<td>2005</td>
<td>Ended</td>
<td></td>
</tr>
</tbody>
</table>

The “Manager Controls” section shows the number of ports available for work and the years being managed by the system. (Although the data we process is discrete for a given year there is always an overlap in processing during first quarter due to the arrival of the new year’s data while we are still finalizing the previous year. Consequently the system is designed to monitor and spawn tasks to process data for multiple years.) The “Process Cycle Time” pull-down menu allows the user to set the page refresh/cycle interval.

The “Workloads” section shows what workloads are active (based on the number of available ports) as well as the number of tasks that can be spawned for each workload (derived from the workload and task definition tables).

Next the “Execution Queue” section displays the formatted contents of the task execution queue. All of the relevant information regarding a task is shown here. The “Delete”, “Hold” and “Release” buttons at the right allow a user to either delete a task from the queue, place it on hold (i.e. defer its execution) or release it from hold, respectively.

Lastly the “Session Log” is a scrollable area that displays a history of the Task Manager’s activity.

THE BIGGER PICTURE

The Task Manager application is now a tightly coupled part of our previously developed SAS/Intrnet report viewing system. User message queues allow notification (Alert boxes) to be triggered alerting staff members to the arrival of new data, a task’s completion or if a task has encountered a problem. Various reporting tasks generate not only RTF output, which is used for distributing reports to the field, but also HTML versions of the reports which can be accessed for viewing from within the online system. Web pages are also generated from which users can glean the current status of data and the report generation processes. Another page allows Task Manager processes to be triggered manually giving the programming staff both automatic and manual control of the system’s processing. As new data arrives XML files are automatically generated which contain information on when the data was received for processing. Another automated process then picks up these XML files and formats them for display on the Internet.
thus allowing state agencies to verify when their data was received for processing. Overall the system flow looks something like this:

![System Overview Diagram]

**CONCLUSION**

As can be seen from the example in this paper there is often functionality and therefore possibilities beyond what the software developers originally intended or documented. I doubt that this sort of application was what the people at SAS Institute had in mind when they developed the SAS/IntrNet product. Nevertheless with some experimentation and a little ‘thinking outside of the box’ I was able to come up with an application that met our needs without the expenditure of additional hard dollars. Many shops with a more formalized methodology to application development would not take such an approach. It is always difficult to find that middle ground between what one wants, what is actually needed, the cost and the level of effort needed to implement a solution. But in those cases where a less formal approach can be used creative solutions can always be found by rolling up one’s sleeves and exploring the possibilities that can be read between the lines of your software’s documentation.

**ACKNOWLEDGEMENTS**

I would like to thank my coworker, coauthor and number one user Bill Bartoli whose steady stream of ideas, corrections and original thinking drove me to address questions I hadn’t even thought to ask. To see more of how the application outlined in this article fits into the larger picture of our day-to-day processing please see Bill’s paper SAS/AF®, SAS/IntrNet®, and ODS: Automation of State Health Data Collection and Report Generation (#130-31).
**CONTACT INFORMATION**
Your comments and questions are valued and encouraged. Contact the author at:

- James L. Ribble
- Centers for Disease Control & Prevention
- Mailstop K66
- 4770 Buford Hwy.
- Atlanta, Ga. 30341-3717
- Work Phone: 770-488-5262
- Fax: 770-488-8150
- E-mail: SUGI@CivilWarGeorgia.com

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.

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