Web-Based Code Generation for Derived Data Using SAS/Intrnet®
Luai Alzoubi, Pfizer Global Research and Development, Ann Arbor, MI
Kent Nassen, Pfizer Global Research and Development, Ann Arbor, MI
Subrahmanyam Pilli, Pfizer Global Research and Development, Ann Arbor, MI

ABSTRACT
Do you want to know how to generate SAS code using SAS/Intrnet? This paper describes a web-accessible SAS/Intrnet-based application that generates dynamic SAS code that can be run to create derived data sets with computed variables. The system accesses Oracle Clinical SAS views and features include user selection of the clinical study, data sets and variables to use. The user must specify analysis week definitions, which weeks are included in the baseline period and which week will be used to carry forward the last observation. The output includes a SAS command file with the customized code to create two SAS data sets for clinical efficacy data and ODS HTML output to display the code in the web browser.

INTRODUCTION
This paper gives an overview of our proof of concept implementation of a SAS/Intrnet-based derived data code generating system in a Unix environment. Our purpose was to minimize some of the tedious, repetitive coding that takes place for most clinical trials. The system is accessed from a web browser. Screens prompt the user for the information necessary to dynamically produce SAS code that, when run, produces two derived data sets with computed variables and a last-observation carry-forward algorithm applied. The application consists of a series of web pages, each generated by SAS/Intrnet, which collect the required information from the user. Each page must pass on required variables to the next page in order to ultimately produce the code that generates the derived data sets. We also discuss several of the problems we had getting the system running.

CONCEPTS AND METHODS
Many clinical studies we work on use derived or analysis data sets. Derived data sets are SAS data sets built from Oracle Clinical raw data, with derived or computed variables and data imputations, that are ready to be used for further analysis without additional processing. These derived data sets have a common structure across most clinical studies, so this allows us to use common code. Our thought was to dynamically generate this common code base, but using specific information about the clinical study. This should, in theory, save the programmer a significant amount of time and allow the programmer to concentrate on the study-specific calculations, which must be added to the code by hand.

Common tasks or data structures that we were able to centralize in a code-generating program include last observation carry-forward (LOCF) processing and creation of variables, such as baseline values, change from baseline values, and percent change from baseline values. Tasks that are specific to the study that cannot be dynamically generated include creating study-specific indicator variables or other data types.

Why use SAS/Intrnet and not Perl or some other popular CGI language? While other languages can certainly generate SAS code as output, we needed to access and manipulate the information and data from Oracle Clinical-generated SAS views. Access to Oracle Clinical data is easy from SAS, so it made sense to use SAS to do that processing. Also, the processing is faster using SAS/Intrnet to directly manipulate SAS data rather than calling SAS from another programming language and then parsing the SAS output.

We realized early on that multiple steps would be necessary because each step provides the user-entered information necessary to display the selection criteria for the next step. This resulted in the need to pass variables between screens using hidden HTML/CGI variables. Because we needed information collected in early screens to be carried forward to the final set of screens, we had to pass that information from screen to screen.

Our operating environment was Unix running the Apache web server. Our SAS/Intrnet installation was on the same Unix system. It was not set up with authentication required for access.

APPLICATION OVERVIEW
In this section we describe each of the web screens created by our SAS/Intrnet application. See Appendix A for screen shots.

Login screen. The user runs this program explicitly from the web browser. The remaining screens are generated by SAS/Intrnet. This part of the program was done in Perl with the CGI module and JavaScript. This was necessary because we needed to authenticate the user’s username and password with the encrypted Unix password database. It was significantly easier to do this in Perl, using its existing crypt() function, than in SAS, which has no such built-in function. After authenticating the user, Perl creates the first web page, a form to select the study to process. This form will then call SAS/Intrnet using the default connection method to run the first SAS program.

Screen 1: Select the Study to process. Here the Oracle Clinical study is selected by the user from a scrolling selection list. This information is needed later to find the corresponding Oracle Clinical views and to select relevant and necessary SAS variables. The study names are collected by piping the output of the Unix commands “ls” and “egrep”, to select just the study names desired, from a filename statement into a SAS data set via a data step using the INFILE command (see Listing 1). The “egrep” command selects only file names beginning with an “s” followed by a digit.

Listing 1. Using “pipe” to get Oracle Clinical study names

```bash
filename data_in pipe '/usr/bin/ls -1
/sas_views/oc/ | egrep "^[s-9]*$";
```
A DATA _NULL_ step is used (see Listing 2) to produce the HTML code (with PUT statements), using SAS/Intrnet to dynamically produce an HTML form with a scrolling selection list of study names. Upon selecting a study, the Application Broker is called to bring up the next screen.

### Listing 2.

```sas
data _null_;
  file _webout;
  set studies end=last;
  if _n_ = 1 then do;
    put '<form method="POST" ' action="http://company.com/cgi-in/sasweb/broker" >/center>';
    put '<html><head></head><body bgcolor="#CC99FF">';
    put '<form method="POST" action="http://company.com/cgi-in/sasweb/broker">';
    put '<html><head></head><body bgcolor="#CC99FF">';
    put '<form method="POST" action="http://company.com/cgi-in/sasweb/broker">';
    put '<html><head></head><body bgcolor="#CC99FF">';
  end;
  if last then do;
    put '</option>' study '</option>';
    put '</option>' study '</option>';
    put '</option>' study '</option>';
  end;
run;
```

Screen 1.5: Select Oracle Clinical View. Similar to how study name was selected in Screen 1, here the Oracle Clinical snapshot name (e.g. stable, current) is selected from a scrolling list generated by a DATA _NULL_ command, which takes piped Unix “ls” output converted to a SAS data set and PUTs the desired HTML tags to create the web page.

Screen 2: Select Data Sets. The next screen allows the user to select the desired SAS views belonging to the Oracle Clinical view previously selected. Here, a PROC SQL step is used to obtain the data set names (see Listing 3), and a DATA _NULL_ command similar to that used in Screens 1 and 1.5 is used.

### Listing 3.

```sas
proc sql noprint;
```

Screen 3 (Steps 3a and 3b): Select Variables from the previously selected data sets. Here, a PROC CONTENTS step on the selected data sets with output sent to an output data set is used to generate a SAS data set containing the list of variables. Three DATA _NULL_ steps are used to generate the HTML code, consisting of one list for each of the two data sets in our example, plus one list for the laboratory efficacy variables (lipid parameters in our example). The first DATA _NULL_ PUTs the opening <HTML> <BODY> and <FORM> tags and the first selection list. The second DATA _NULL_ PUTs just the second selection list. The third DATA _NULL_ PUTs the last selection list and the closing tags for <FORM>, <BODY>, and <HTML>. Required variables are pre-selected using the HTML form command “<option selected>”. In each of these selection lists, the user is allowed to select multiple variables. This page is displayed as one HTML form with the three selection lists and one SUBMIT button.

Screen 4: Enter number of Weeks. The efficacy data sets that will be produced by the generated code are structured as multiple observations per patient in time units of weeks. Here, the user is simply asked for the number of weeks to be used for the study analysis.

Screen 5: Enter Start and End Times for Weeks. Weeks are based on study days (days since patient first started treatment). These intervals define the time windows used in the analysis. Observations will be categorized into the weeks defined on this page, depending on the study day they occurred. This form collects start days, end days, and week names. This results in several more variables to pass on to the remaining stages of processing.

Screen 6: (Steps 6 and 7) Select Baseline Week(s) and LOCF Week(s). Further information is collected, this time for the computation of baseline average values and for the LOCF algorithm, which needs to know which weeks the last observation carry-forward will be applied to.

Final Screen. The last program of the system pulls together all this collected user information and uses it to dynamically create SAS code, which applies the desired rules. The method of generating SAS code consists of a series of DATA _NULL_ steps with PUT statements. Macro variables are expanded during this run to apply the information previously gathered. The HTML screen generated by SAS/Intrnet tells the user where to find their output.

One final bit of information provided to the user is a quick listing of the generated code, produced by ODS HTML. The generated SAS code is read into a SAS data set, which is then returned to the web browser with a PROC PRINT. This allows the user to verify that their code has been generated correctly without leaving their browser.
DISCUSSION
During the development of this code generation system we encountered several problems. We initially began testing on static data sets. This was fast and there were no problems with excessive access times. However, when we switched to our target Oracle Clinical SAS views we found the timeout interval to be too short for some studies. Accessing Oracle Clinical views can be slow, especially for very large databases. SAS/Intrnet will “timeout” in these instances. This led to increasing the default timeout interval in the SAS/Intrnet installation.

We used multiple screens rather than one large screen or one screen with HTML frames. This was necessary because information entered on one screen was used to build the next screen. We could have used HTML frames, but for a proof of concept application, this would have to wait until a later revision.

We had Oracle Clinical SAS views available on the same Unix system we were running the SAS/Intrnet application on. Since they pointed to the Oracle Clinical database, we needed to authenticate to the SAS/Intrnet “pool” service to see them and access them. This required holding the user’s username and password in macro variables at least up to the last access of the Oracle Clinical SAS views (Screen 2).

Passing the user’s password from screen to screen meant hiding it from the casual user. By using the POST method for HTML form submission, we avoided seeing the user’s password and username on the browser command line. However, this did not prevent a user from using the SAS/Intrnet DEBUG=131 command, which would then show all variables, hidden and unhiden. This is a problem we hope to eliminate in our production version of the code.

There was no shortcut to get a variable value from first screen to last screen, resulting in tedious copying of variables. For example, we needed the user’s username in the very last program in order to correct the ownership of the generated SAS code (files are by default owned by the owner of the SAS/Intrnet process—which is not the user running it from the web). This required passing the username from the very first screen through all intermediate screens, to the last screen. By the time we were half way through the gathering of information we already had multiple week definitions, variables selected, and data set names to pass. This was quite list of hidden variables! While this is mostly tedious to do, it does present possibilities for errors, and points out the need to be well organized before sitting down to write code like this.

We made heavy use of the SAS/Intrnet DEBUG=131 facility during development. This shows the user the values of the SAS variables being passed and processed and can be a lifesaver when you aren’t sure what is going where (or when!).

We would have liked to have made the application entirely in SAS, but we didn’t have time to implement the Unix crypt(3) encryption algorithm in SAS for the authentication of Unix passwords. This authentication step was the only use of Perl in the entire application.

CONCLUSION
SAS/Intrnet was used to produce a proof of concept derived data set code generating system that saves the programmer the time of hand-coding the commonly used algorithms. The system dynamically generates SAS code specific to a selected Oracle Clinical database view. A series of HTML forms is used to collect user requirements for standardized derived data sets. This information is used to create SAS code that can be modified further by the programmer as needed. During the process of developing the system we dealt with several problems, including timeouts, password authentication on Unix, and keeping track of the many user-specified selections collected during the application’s execution. We found SAS/Intrnet to be a useful tool for creating such a system.

REFERENCES
CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors at:

Luai Alzoubi
Work Phone: (734) 622-2168
Email: Luai.Alzoubi@pfizer.com

Kent Nassen
Work Phone: (734) 622-2048
Email: Kent.Nassen@pfizer.com

Subrahmanyam Pilli
Work Phone: (734) 622-3265
Email: Subra.Pilli@pfizer.com

Address:
Pfizer Global Research and Development
Ann Arbor Laboratories
2800 Plymouth Road
Ann Arbor, MI 48105

Web: www.pfizer.com
Login Screen

Screen 1: Select Study

Step 1: Select Study

This request took 0.28 seconds of real time (v8.2 build 1391).
Screen 1.5: Select Oracle Clinical View

Step 1.5: Select View

[Screen with selection options]

This request took 0.28 seconds of real time (v8.2 build 1391).

Screen 2: Select Datasets

Step 2: Select Datasets from release1865mar19

(DEMO and NORMLAB2 probably most often!)

[Screen with dataset selection options]

This request took 14.65 seconds of real time (v8.2 build 1391).
Screen 3: Select Variables from Previously Selected Datasets

**Step 3a: Select Variables for DEMO**

**Step 3b: Select Variables for NORMLAB2**

Screen 3: Select Variables, continued.

**Step 3b: Select Lipid Variables**

*This request took 26.93 seconds of real time (v8.2 build 1391).*
Screen 4: Select Number of Study Weeks

Step 4: Select Weeks

Enter # of Weeks: 

Continue

Reset

dataset: DEMO_normals2

This request took 0.06 seconds of real time (v8.2 build 1091).

Screen 5: Set Time Windows for Study Weeks

Step 5: Enter Start and End of each Week

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>0.1</td>
<td>0.9</td>
<td>0</td>
</tr>
</tbody>
</table>

This request took 0.08 seconds of real time (v8.2 build 1091).
Screen 6: Select Baseline and Last Observation Carried Forward (LOCF) Weeks

Step 6: Select the Baseline Weeks for NORMLAB2

Step 7: Select the LOCF Week

Final Screen: Location of Output and Review of Generated SAS Code

Generating SAS code for s0000_00$release186Smar19..
See your output in file /var/tmp/dds_s0000_00.sas
Review your output here...

```sas
Obs   LINE
1  **********************************************************************
2  * dds_s0000_00.sas *
3  **
4  * This program creates the derived efficacy databases *
5  * for cardiovascular lipid data for 0000_00 with *
6  * Oracle Clinical view release186Smar19 *
7  **
8  * Macro calls: %weekkg Computes weeks and is included *
9  * in this program. *
10  * RebuildDB Create the derived DS code *
11  **
12  * Generated by the Wonderful Cardiovascular Derivd *
```