Paper 051-2011

Stacking the Deck: Dynamically Filtering Prompt Values for Stored Processes

Alesia Pehl, Blue Cross and Blue Shield of Minnesota, Eagan, MN
Susan Trelstad, Blue Cross and Blue Shield of Minnesota, Eagan, MN
Kyle Koenig, Blue Cross and Blue Shield of Minnesota, Eagan, MN

ABSTRACT

In self-service applications that access sensitive information, accurate data filtering cannot be left open to chance. Companies must fulfill legal obligations to restrict access to data or they might need to protect data to meet business requirements. Integrating security and other business rules to filter table-driven prompt values can be a significant challenge, especially if the source database does not accommodate row-level security. SAS® 9.2 has new capabilities for presenting user-appropriate content in report-selection prompts. This paper offers two solutions for implementing dynamically filtered parameters: one for off-the-shelf SAS® applications and the other for custom applications. The off-the-shelf solution uses SAS® Information Maps to source dynamic prompts for SAS Stored Processes®. The custom solution uses stored processes to populate parameter values in a Java Web application. Having implemented both at Blue Cross and Blue Shield of Minnesota, we share our lessons learned while comparing the two solutions. Never gamble on filtered content again.

INTRODUCTION

Dynamically filtering parameter prompt values is a critical and valuable functionality for both SAS and custom web applications. Filtered parameter prompt values provide a level of control that supports a precise report generation configuration.

Prompt values can be limited to meet specific requirements for various considerations. User security can be used as data filtering criteria when generating a list of prompt values. Security is a major driving factor when filtering allowable prompt values for a user. Application specific security roles not only allow users to log in to the application and access specific reporting objects, they can also be used to filter a prompted parameter list when the information is available in a retrievable format.

Business rules can dictate the need for filtered prompt values. The business rules function as fine-grained security to provide access to only particular rows in a data source. Prompt values that are not meaningful, based on business usage, are excluded from specific reports. Situations such as these define prompted content in terms of the report's functional criteria. The applications highlighted in this paper have specific requirements for security and usage that require the use of dynamically filtered parameters.

The off-the-shelf Blue Cross SAS application delivers employer group reports to external insurance agents and internal underwriting and sales users. The parameters are dynamically filtered to display only the appropriate content based on the user's security role and their association with the employer group(s). As an example, multiple users might have access to the same report because they have the same security role, but see different values in the dynamic pick-list due to their unique business responsibilities.

The custom Blue Cross application delivers employer group reports to external insurance agents, employer groups, and internal sales, underwriting, finance, and informatics users. The client list is dynamically filtered based on the security role, association with the employer group(s), and type of report. For example, a user requests security to a specific role and employer group. When the user logs into the application, only those reports which correspond to the user's security role are viewable. When the user selects parameters for a report, the list of available employer groups is pre-filtered to display only the employer group(s) to which the user has a business connection.

Both applications are highly visible, servicing thousands of external users. Delivering sensitive information to external entities needs to be without error. Health companies such as Blue Cross and Blue Shield of Minnesota are legally required by the Health Insurance Portability and Accountability Act (HIPAA) to enable security to protect the confidentiality of patients' medical information. Violation of this law carries significant fines.

The implementation of dynamically filtered prompt values is procedurally similar between the off-the-shelf SAS and the custom Java applications, but the technology used is different. Figure 1 shows the architecture of the two solutions. The key similarity to both approaches is that they are data-driven. An underlying data structure contains the business information necessary to drive decisions as to which values to display. The SAS application uses data stored in SAS data sets and Oracle tables compared to the custom application, which uses a series of SAS data sets. The tools used to query these table structures are quite different. The off-the-shelf SAS solution uses an information

map to access the data; the custom solution uses a stored process to retrieve data in XML format. In the off-the-shelf SAS solution, the values are used to populate a dynamic prompt in a stored process while the custom application feeds the XML data into a Java web application to create a parameter.

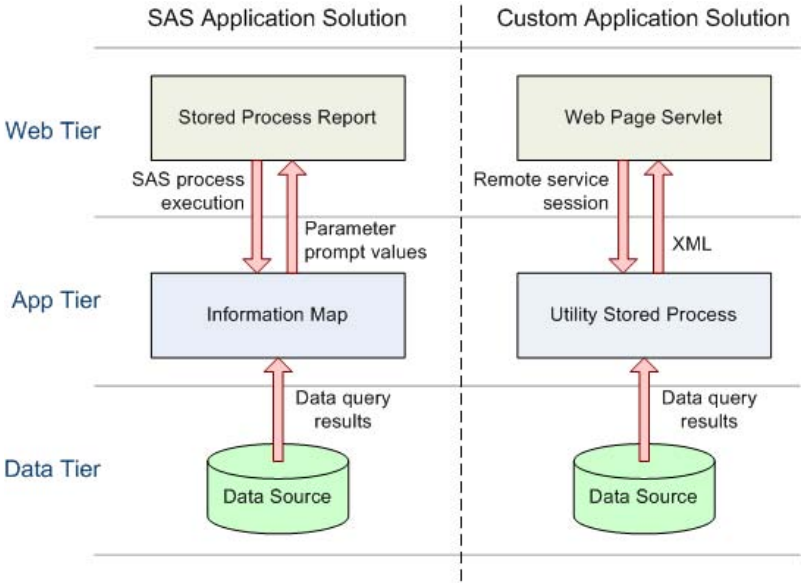

Figure 1. Architectural Comparison between Off-the-Shelf SAS and Custom Solutions

DYNAMICALLY FILTERING PROMPT VALUES USING SAS INFORMATION MAPS

The SAS Information Delivery Portal provides a user-friendly interface from which to launch stored processes. Stored processes are just one of the various objects that can be accessed via the Information Delivery Portal, but they have the benefit of producing highly customized output. Using SAS 9.2, they become an even more powerful way of generating reports. Now, developers can create dynamic stored process prompts sourced from SAS Information Maps. Because information maps can be assigned pre-filters, including authorization-based filters, stored process parameter prompt values can also be filtered. Another benefit coming out of SAS 9.2 is that dependencies can be applied to prompts, based on previously selected prompt values, so parameter prompt values can be cascading.

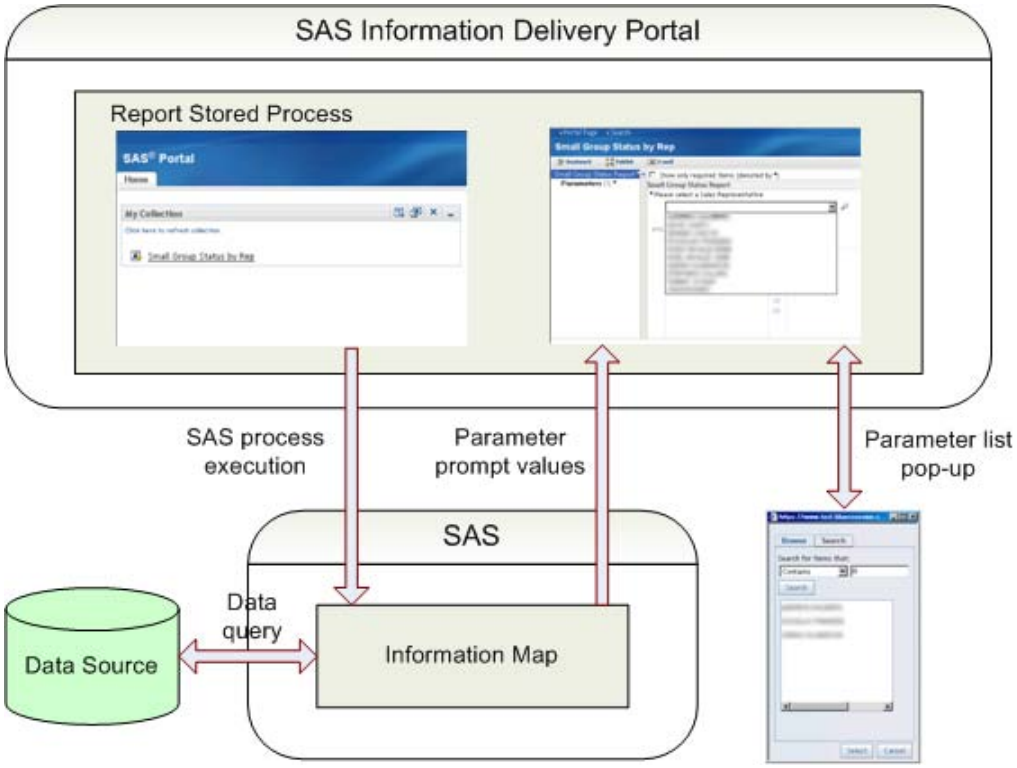

Figure 2. Dynamically Filtered Parameters Using SAS Information Maps

CREATING THE INFORMATION MAP

The information map is the key to selecting filtered data and returning the results to the stored process dynamic prompt. The information map must have certain features in order for it to populate the parameter prompts accurately and simulate row-level security. First, a table is required, containing a list of user IDs and the data that each user ID is authorized to access. For example, if sales representatives are only supposed to access the accounts that they have sold, the table would contain sales representatives' user IDs and their corresponding sales rep identifiers. This look-up table is then joined to other table(s) containing the sales rep identifier and detailed information about their accounts, as shown in figure 3. Configuring the look-up table as a "Required Table" in the information map properties ensures that the table is queried each time the information map is accessed.

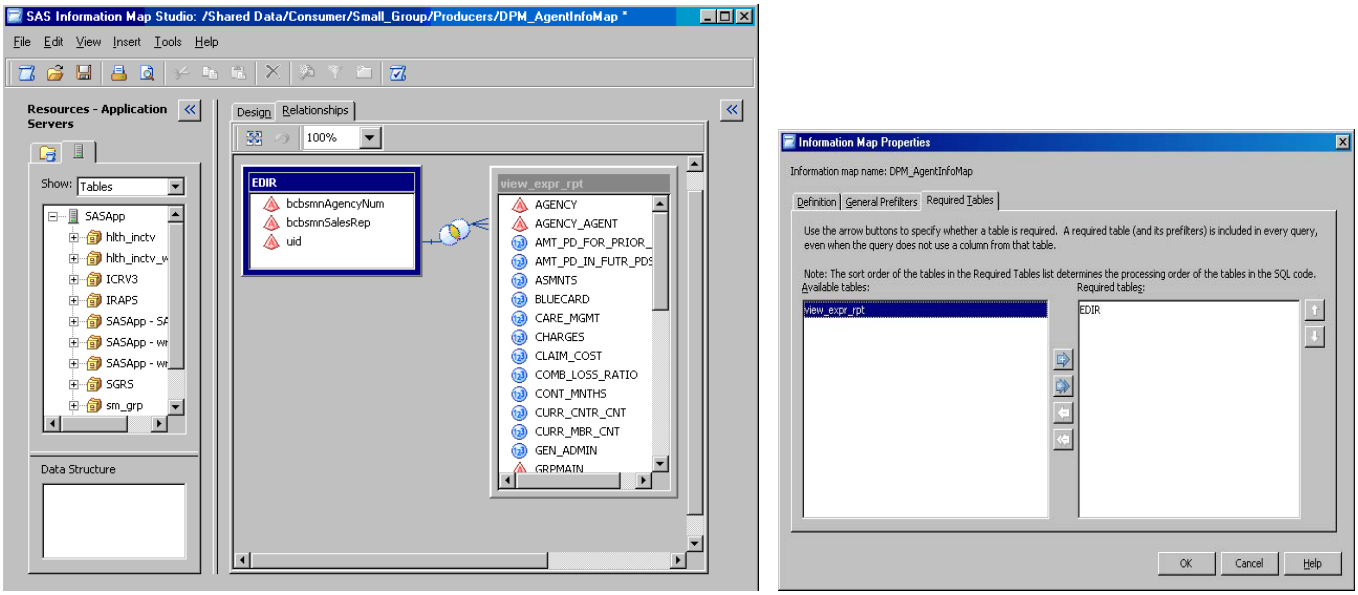

Figure 3. Information Map Security Table

After configuring the data, the information map filter(s) can be defined. SAS metadata identity-driven values, such as &SAS.Userid, can be used to simulate row-level permissions for the authenticated user ID. Figure 4 illustrates how to define a filter to ensure that the user ID in the security table is equal to the &SAS.Userid. The filter must then be defined in the information map properties as a "General Pre-Filter".

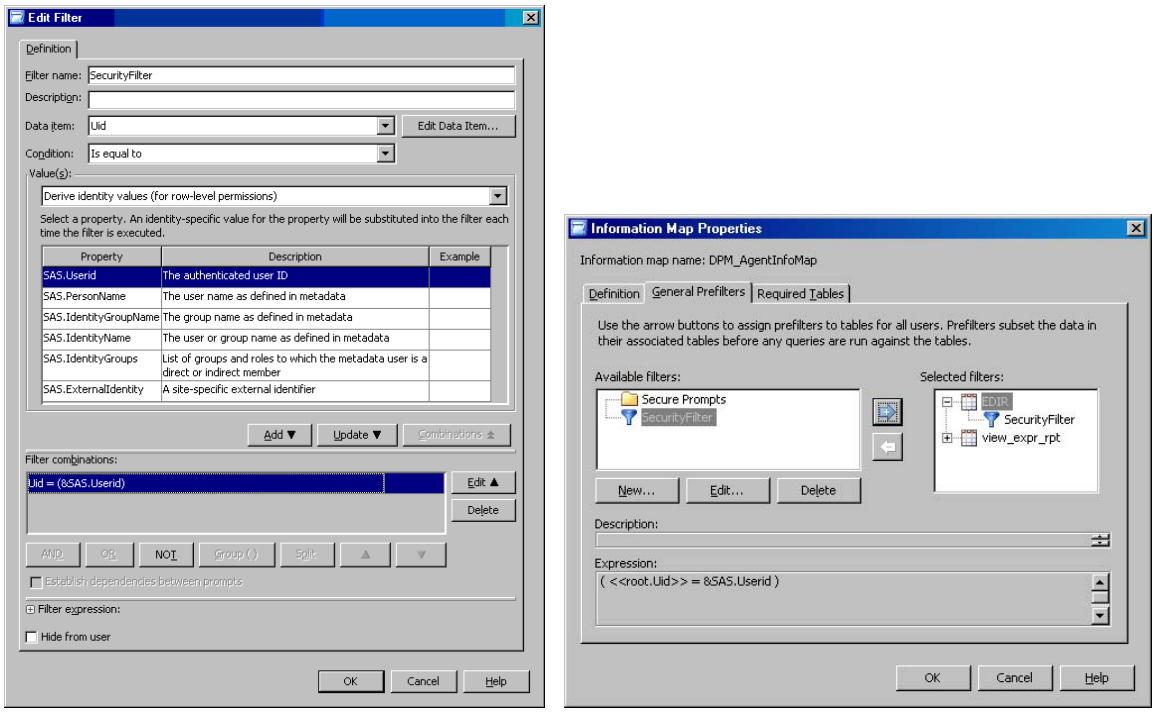

Figure 4. Information Map Security Filter

SETTING UP THE STORED PROCESS PROMPTS

SAS Management Console 9.2 is used to set up stored process parameter prompts that are dynamically populated using an information map. The prompts can be added as shared parameters, if there are multiple stored processes that require the same prompt. The current version of Enterprise Guide (version 4.3) does not allow the creation of shared prompts or the sourcing of dynamic prompts from information maps. However, if the parameter is created as a shared prompt in Management Console 9.2, it can be added to other stored processes using Enterprise Guide (version 4.2 or higher).

The steps to create the dynamically generated parameter prompts, as shown in Figure 5, are as follows:

1. Create a new prompt and navigate to the "Prompt Type and Values" tab at the top.
2. Select a dynamic list as the method for populating the prompt.
3. Browse to the location of the information map that contains the security filter and data item to populate the prompt.
4. Choose the data item to populate the dynamic prompt list. One data item can be used for the code value (unformatted), to be passed to the stored process, and another for the displayed value (formatted), to populate the pick list. The unformatted values can be appended to the formatted values within the pick list, so that the user can see both.

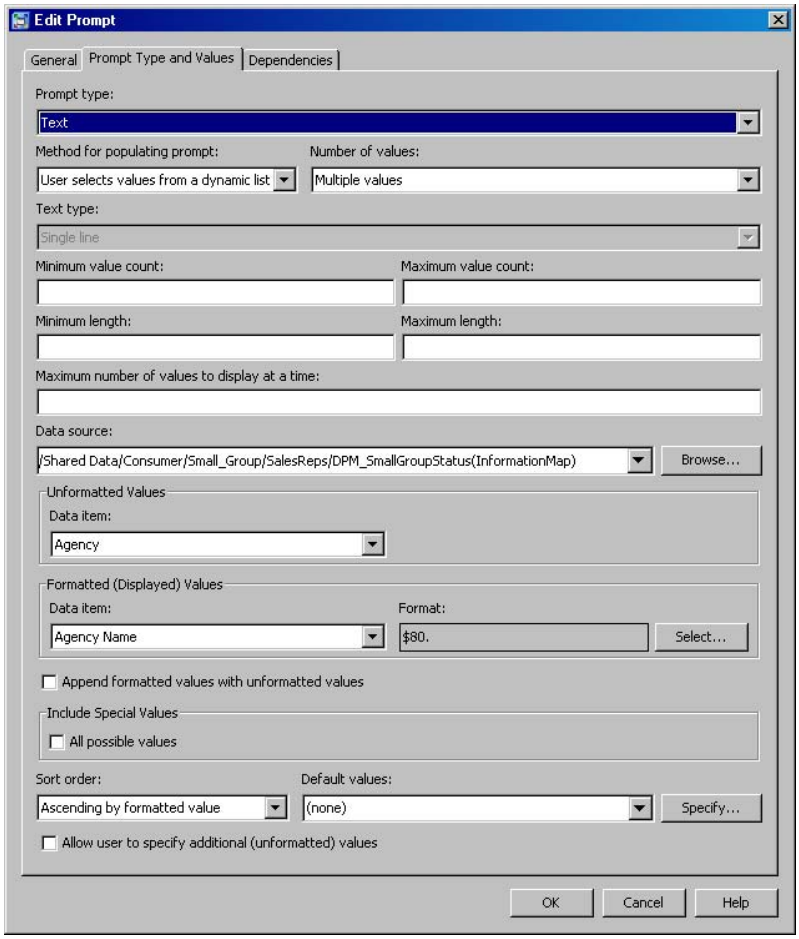

Figure 5. Stored Process Parameter Prompt

After filtered data is received from the information map for the initial prompt, the stored process can populate any dependent dynamic prompts based on the selected value (or default hidden value) of previous prompts. This is achieved through setting up prompt dependencies, located on the "Dependencies" tab in Figure 5.

When the prompt allows multiple values to be selected, as in Figure 5, additional logic is required in the stored process SAS code. A SAS data set, populated with the macro variable values, can be built as in the following example:

```
data agencyagent;
    length agency_agent $10.;
    %if &agencyagent_count >1 %then
    %do i=1 %to &agencyagent_count;
       agency_agent=resolve("&&agencyagent&i");
       output;
    %end;
run;
```

If the potential list of values is longer than a few items, the data set created above can be merged with the report query result set, keeping only the selected values. Otherwise, a comma-delimited string of values can be created from the data set, made into a macro variable, and included in the query criteria. Of course, if the count of selected values is just one, that single value can be used in the report query filter criteria.

LAUNCHING THE STORED PROCESS

Once the stored process is created, it can be accessed via the SAS Information Delivery Portal. When the stored process is kicked off, the information map generates the query to return values into the first dynamic prompt. Subsequent prompts that are dependent upon the value(s) selected in previous prompts are grayed out until a value is selected.

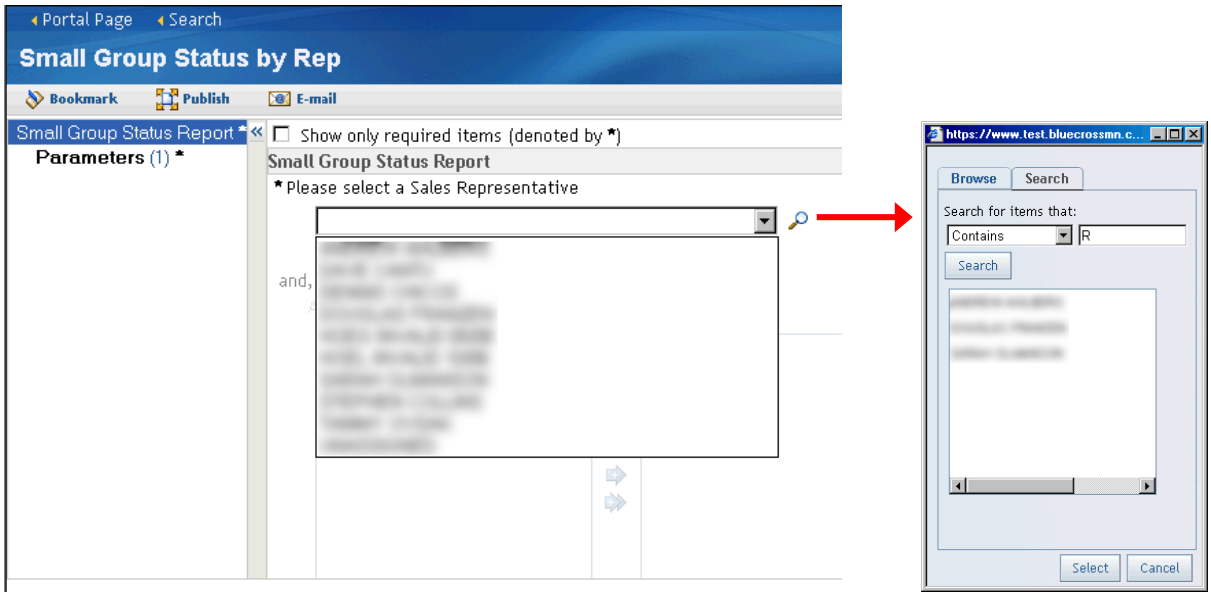

Figure 6. Stored Process Parameter Page

Upon clicking on the magnifying glass icon next to the drop down list, as seen in figure 6, users can enter search criteria. This is particularly useful when the list of values is long, or when the list is, for example, sorted by last name, but only the first name is known, etc. The operators that can be selected for the search criteria are: contains, exactly matches, starts with, ends with, and matches pattern.

The SAS 9.2 release provides a quick and easy solution for generating dynamic secure prompt values, and for cascading prompts. Previously, this was only possible with a custom web application. One drawback, however, to the out-of-the-box solution is that drop down lists are empty when no values are found. Unfortunately, there is no message to the user to indicate that they do not have any values corresponding to their business function. This is one scenario where a custom web application might still be beneficial.

DYNAMICALLY FILTERING PROMPT VALUES FOR CUSTOM WEB APPLICATIONS

Dynamically filtering user prompt values is equally important and useful for custom web applications as it is for off-the-shelf SAS applications. One way to query, filter, retrieve and display user prompt values on a custom web application page is to combine the functionality of a Java-based web servlet with a SAS Stored Process.

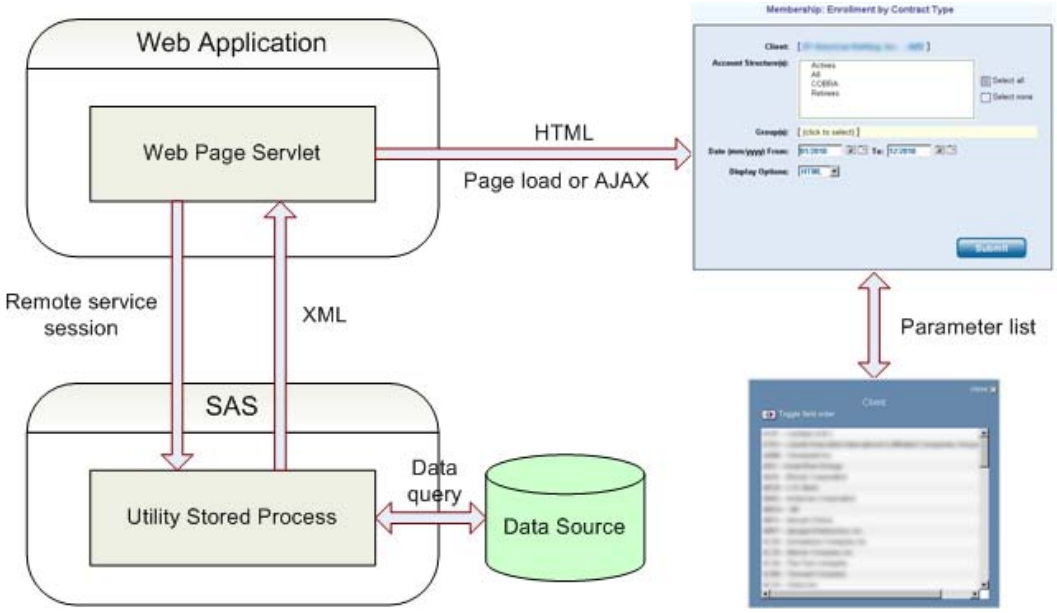

Figure 7. Dynamically Filtered Parameters for Custom Web Applications

INVOKING THE WEB PAGE APPLICATION SERVLET

A web application can utilize a Java servlet to connect to and access SAS services. Using the Java libraries provided by SAS, the servlet can create a stored process instance, pass query filter criteria to it as parameters and execute it.

A Java-based web servlet is used to generate the user interface and manage the display, functionality and content of the parameter prompts on a web page. It utilizes the Java libraries provided by SAS to instantiate various objects required to access SAS services. The Java libraries provide a vast array of capabilities and functionality. The logic needed to access and utilize the SAS services and resources through a Java application can be extensive and is outside of the scope of this paper. The online SAS JavaDocs documentation is listed in the References section of this paper.

The servlet must contain specific logic to connect to SAS Remote Services and locate the stored process service. The SAS services are used to access the stored process functionality rather than using the /SASStoredProcess web application directly. This provides greater control of the process and allows for passing parameters through the service connection instead of appending them to the stored process web URL.

After the servlet connects to the stored process service, it must pass the query filter criteria as a list of parameter values to the stored process. The values used as query filter criteria to the stored process could come from user input from web page forms, embedded values within the servlet itself or from data returned by a different stored process, all possibly representing different filtering criteria. User-specific security information, report criteria or other conditions at run-time could be used as query filtering values to be passed to the stored process.

The Java servlet then runs the data query stored process. The stored process execution can be in either a synchronous or asynchronous mode. Executing the stored process synchronously would require less Java code, but will block input and output for the servlet until the stored process has completed. For short running queries this should not be a problem. Utilizing an asynchronous call of a stored process requires more Java coding to run the stored process in its own executable thread and requires logic to monitor the progress of the query process. This is best used for queries that might take longer than a few seconds to complete.

Java servlet code fragment showing how a stored process object is created and executed:

```
// Create the stored process
PathUrl path
   = new PathUrl("sbip://" + metadataRepositoryPath + stpName + "(StoredProcess)");
com.sas.services.storedprocess.metadata.StoredProcessInterface metadataSPI =
(com.sas.services.storedprocess.metadata.StoredProcessInterface)privUserRepository.
getObjectByPath(path);
StoredProcessOptions spo = new StoredProcessOptions();
StoredProcess2Interface storedProcess
 = (StoredProcess2Interface)storedProcessService.newStoredProcess(metadataSPI,spo);

// Set stored process parameters
HashMap<String,String> params = new HashMap<String,String>();
params.put("SOMEPARM","some value");
stp.setParameterValues(params);

// Execute the stored process
Execution2Interface stpExecution = stp.execute(false,null,false,null);

// Read in the results
BufferedReader stpOutputStream = null;
com.sas.io.InputStreamHeaderInterface header
    = stpExecution.getInputStreamHeader("_WEBOUT");
stpOutputStream = new BufferedReader(
    new InputStreamReader(stpExecution.getInputStream("_WEBOUT")));
```

EXECUTING THE UTILITY STORED PROCESS

The stored process called by the Java servlet executes base SAS code to access one or more data sources and returns the filtered data. It is designed to return data to the calling process instead of creating and displaying a user readable report. In this configuration the stored process is referred to as a “utility” stored process instead of a user-consumable, report-generating stored process.

A stored process session is used to execute the query logic instead of a workspace session since a stored process session is already initiated and provides a quicker response, which is important for web applications.

The stored process uses the passed filter criteria to query data from one or more sources. It can access any data that's available to SAS and has the capabilities of any base SAS program. The stored process can contain any DATA step or procedure to retrieve data. However, since the results of the query will not be used for display purposes, no PROC REPORT, ODS or other page formatting code should be used in a utility stored process.

The stored process must be coded to return the filtered results in XML format. The utility stored process can optionally be coded to return multiple XML data sections in a single execution. The code example below offers more information on the design and logic of a stored process that returns XML data.

Utility stored process code example:

```
filename xmldata temp lrecl=200 mod;
libname xmldata xml;

%macro build_xml(data_table);
data xmldata.&data_table;
  set &data_table;
run;
%mend build_xml;

data example_1;
  set some_data_source;
run;

%build_xml(example_1)
```

```
data _null_;
  infile xmldata length=L end=eof;
  file _webout;

  input @;
  input @1 key $6. @1 line $VARYING32767. L
        ;
  if (_n_ gt 2) and (eof = 0) then do;
    if key in ('<?xml ', '<TABLE', '</TABL') then delete;
  end;

  label line='00'x;
  put line;
run;
```

POPULATING THE PARAMETER LIST

The servlet must detect when the stored process query has completed and should include logic to verify that it has completed successfully. SAS provides a variety of return codes and conditions that can be used to determine the success or failure of a stored process executed from Java code. If the execution of the stored process is successful, the servlet must retrieve and process the results of the query. The Java servlet must be able to handle the condition when no valid values are found by the query stored process and no result XML is returned.

The servlet reads the filtered data from the returned XML stream and uses the information to dynamically generate HTML and other content needed to populate a web page. If utilizing AJAX capabilities, the information can be used to update specific areas of a web page. The data received from the stored process could be displayed on the web page as static text, a single value parameter, a parameter list or a hidden support parameter. There are several options to display and combine filtered and other parameters, including selection drop-downs and scrollable selection lists.

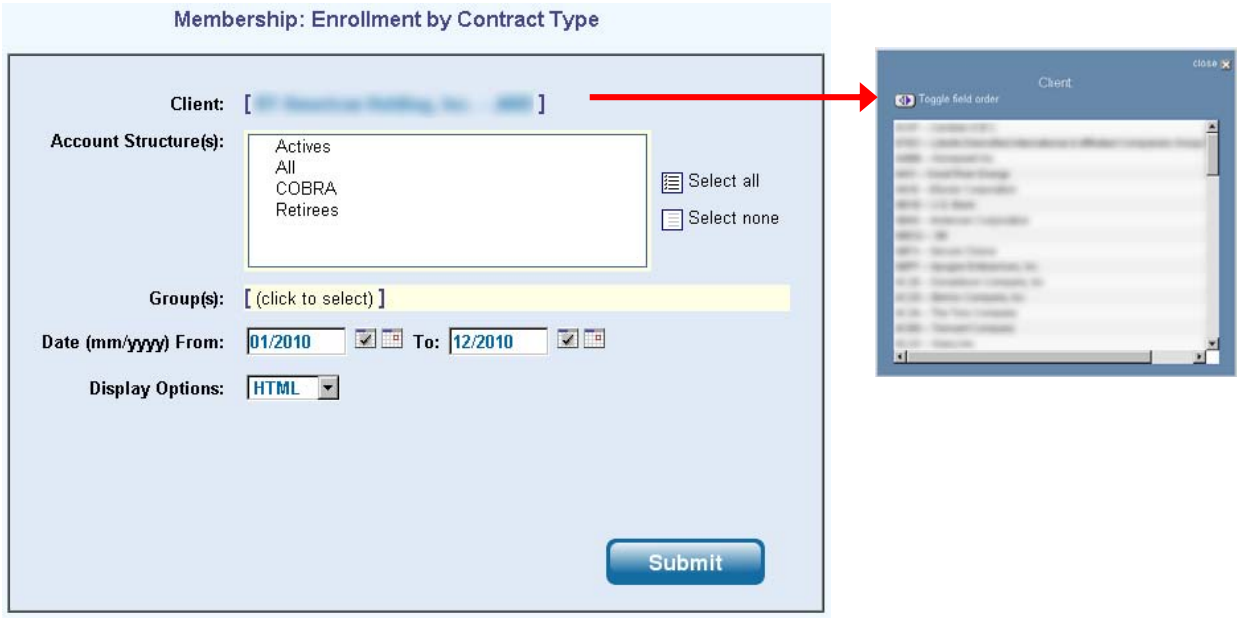

Figure 8. Dynamically Generated Prompt and Associated Pop-up Values List

Instead of a list of values, the utility stored process could be coded to return a single, non-selectable value. This value could be displayed on the page as static text or could be retained as a hidden value not shown to the user.

Since XML supports multiple data sections in a single file, the results from a single utility stored process call could be used to populate multiple displayed or hidden parameters.

Other user input, such as parameter value entry or selection, could be passed to the utility stored process as filter criteria. The output value of a data filtering utility stored process could also be used to as input to another utility stored process. Multiple cascaded utility stored processes could be linked to support a chain of parameter dependencies.

CONCLUSION

The off-the-shelf SAS solution offers significant time and cost savings as it requires only SAS Information Map Studio, SAS Management Console, and Enterprise Guide to create the necessary information map and stored process objects. A developer who is familiar with these tools is able to incorporate this functionality with general knowledge of the corporate security model and business rules. Using the off-the-shelf technology and home-grown experience contributes to the ease and speed of development making this solution an attractive choice. To further promote efficiency, the stored process prompts can be shared among other stored processes, saving time and money for future projects.

The custom solution offers the flexibility of a customizable user interface which can be especially important when a particular look and feel is needed for branding purposes or incorporating stand-alone or web-based Java applications. The custom solution marries the excellent data handling capabilities of SAS with the more robust and refined user-interface of Java. In addition, it provides the opportunity for custom messaging to clearly communicate the dynamic nature of the data and status of the filtering process. Various data sources can be used to drive the parameter content filtering. Any source that is accessible by base SAS code can be leveraged by the utility stored process. This provides the ability to access disparate data sources, expanding the data source options beyond that which can be consumed directly by an information map. This utility stored process can then be shared by many reports to promote reusability.

REFERENCES

SAS Institute Inc. (2010), “Fine-Grained Controls for Data”:
http://support.sas.com/documentation/cdl/en/bisecag/61133/HTML/default/viewer.htm#a003250936.htm

SAS Institute Inc. (2010), “How to Implement BI Row-Level Permissions”:
http://support.sas.com/documentation/cdl/en/bisecag/61133/HTML/default/viewer.htm#a003086341.htm

SAS 9.2 JavaDocs: http://support.sas.com/rnd/gendoc/bi92/api/index.html

ACKNOWLEDGMENTS

Special thanks to our colleague, Mike Wesley, for his contributions to the SAS solution.

CONTACT INFORMATION

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

Name: Alesia Pehl
Enterprise: Blue Cross and Blue Shield of Minnesota
Address: 3535 Blue Cross Road
City, State ZIP: Eagan, MN 55122
Work Phone: 651-662-1679
E-mail: alesia_pehl@bluecrossmn.com

Name: Susan Trelstad
Enterprise: Blue Cross and Blue Shield of Minnesota
Address: 3535 Blue Cross Road
City, State ZIP: Eagan, MN 55122
Work Phone: 651-662-7070
E-mail: susan_trelstad@bluecrossmn.com

Name: Kyle Koenig
Enterprise: Blue Cross and Blue Shield of Minnesota
Address: 3535 Blue Cross Road
City, State ZIP: Eagan, MN 55122
Work Phone: 651-662-8790
E-mail: kyle_n_koenig@bluecrossmn.com