Paper 1488-2014

Custom BI Tools Using SAS® Stored Processes

Scott Hutchison, US Department of Health and Human Services Office of Inspector General;
John Venturini, Piper Enterprise Solutions

ABSTRACT

Business Intelligence platforms provide a bridge between expert data analysts and decision-makers and other end-users. But what do you do when you can identify no system that meets both your needs and your budget? If you are the Consolidated Data Analysis Center in the HHS Office of Inspector General, you use SAS® Enterprise BI Server and the SAS® Stored Process Web Application to build your own. This paper covers the inception, design, and implementation of the PAYment by Geographic Area (PAYGAR) system, which uses only SAS® Enterprise BI tools, namely the Stored Process Web Application, PROC GMAP, and HTML/JAVA embedded in a DATA step, to create an interactive platform for presenting and exploring data that has a geographic component. In particular, the paper reviews how we created a system of chained stored processes to enable a user to select the data to be presented, navigate through different geographic levels, and display companion reports related to the current data and geographic selections. It also covers the creation of the HTML front-end that sits over and manages the system. Throughout, the paper emphasizes the scalability of PAYGAR, which the Stored Process Web Application facilitates.

INTRODUCTION

This paper gives an overview of how one Federal government agency, with limited staffing and budget, used SAS tools to create the PAYment by Geographic Area (PAYGAR) system, a custom Business Intelligence application for visualizing and exploring data with a geographic component. PAYGAR is a joint effort between the Consolidated Data Analysis Center (CDAC), a division of the Department of Health and Human Services Office of Inspector General (OIG), and the Centers for Medicare and Medicaid Services (CMS). CMS provides the physical servers and SAS® Enterprise Business Intelligence platform on which PAYGAR was constructed and currently operates. Developers from Maricom, under contract to CMS, wrote nearly all the code for the first operational version of PAYGAR with direction from CDAC. Using the knowledge gained from working with Maricom during this period, CDAC has taken over ongoing development with support from eDaptive (which succeeded Maricom as the CMS contractor for SAS® Enterprise Business Intelligence).

Disclaimer: The views and opinions in this paper reflect solely those of the authors and are not official statements or positions of the Office of Inspector General for the Department of Health and Human Services.

THE BUSINESS NEED

Analyzing immense datasets to identify geographic patterns of potential fraud is crucial to the work of the CDAC. However, the skills needed to efficiently discover patterns among billions of Medicare claims and those needed to evaluate whether those patterns should spur investigative activity do not always coexist in the same individuals. Historically, CDAC analysts distributed analytic information to customers (primarily OIG criminal investigators) through adhoc reports and spreadsheets, but this arrangement makes the customers dependent on CDAC's very small staff, which may have other priorities at a given moment. Furthermore, reports and spreadsheets distributed via email are static objects and may not be based on the most current data. With these factors in mind, CDAC sought to build a bridge between the technical skill and subject matter expertise by developing a platform by which customers could easily explore geographic metrics created by CDAC analysts.

CDAC defined the following as critical elements of a successful solution:

- A three-pane presentation consisting of:
    1. a fully navigable map in the center pane,
    2. a series of drop-down menus in the left-hand pane, and
    3. a space reserved for 'companion reports' in the right-hand pane.
- User selection of metric, time period, subgroup, and geography parameters drives the display of the map and companion reports
- Scalable design to allow CDAC to add new parameters with minimal effort.

PLATFORM OVERVIEW

CDAC analysts collectively have many years of SAS programming experience, so we strongly preferred a SAS-based solution. In an example of unexpected synergy, the CMS had invested in SAS® Enterprise Business Intelligence as an analysis and development platform for their new Integrated Data Repository. In consultation with OIG and CMS management, CDAC decided to leverage this platform, including contract support from Maricom, to house PAYGAR.

The SAS Enterprise BI server is used at CMS in three different environments: development, validation and production. Following CMS lifecycle, the application was developed in a Development environment and then passed to the business owner (CDAC) for validation in Validation environment. Following sign-off, the application was certified and deployed in Production. CDAC continues to conceptualize new features of the application in Validation, while the locked-down version sits in Production. If at any point, CDAC wants to promote any new feature, they work with administrators to promote the new version in Production. This ensures that enhancements to the application are promoted when they are thoroughly tested in all environments. Figure 1 illustrates the CMS software development lifecycle.

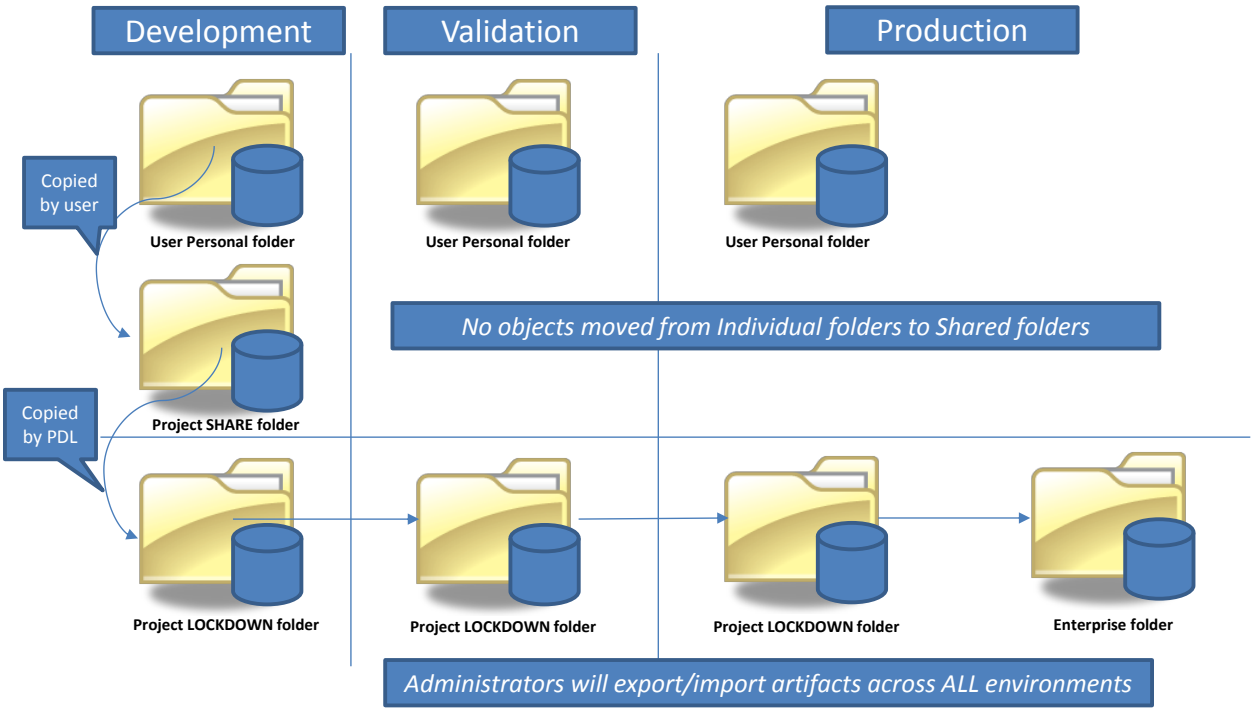

Figure 1: CMS Software Development Lifecycle

Within the SAS Enterprise BI Server toolset are Enterprise Guide, Enterprise Miner, and most notably, SAS Integration Technologies. Of the SAS Integration Technologies suite, the most important feature for PAYGAR is the SAS® Stored Process Web Application service. PAYGAR utilizes the Stored Process Web Application service to chain several reporting stored processes together. The Stored Process Web Application service also allows users to run stored processes that are written in SAS, but may call features such as Cascading Style Sheets (CSS) or Javascript within the confines of the SAS code. Details of how PAYGAR uses these features of the Stored Process Web Application follow in later sections of this paper.

APPLICATION OVERVIEW

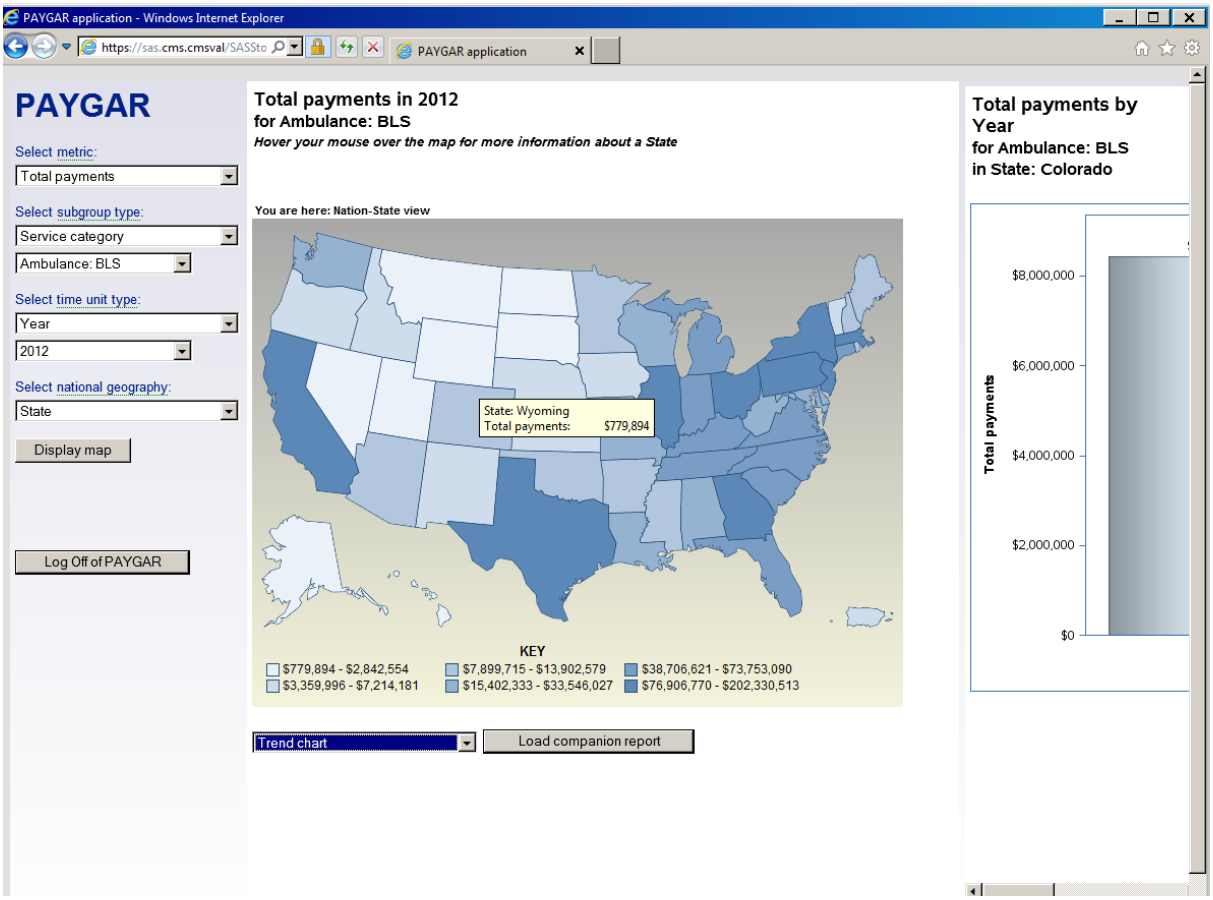

Figure 2: PAYGAR screen showing sub-menus, State map, companion report, and tooltip

PAYGAR allows users to explore geographic metrics produced by CDAC analysts according to their own interests and on their own schedule. To access PAYGAR, the user simply navigates to the URL and then enters their system credentials.

Once logged in, the PAYGAR website presents the user with three panels. Four drop-down menus and two buttons, 'Display Map' and 'Log Off of PAYGAR', comprise the left-hand panel. Initially, CDAC logos appear in the middle and right-hand panels. The user selects a desired metric, subgroup type, time unit type, and national geography from the left-hand panel. The subgroup type and time unit type menus each create a sub-menu that permits specification of the exact subgroup level or time frame when the user selects the related type.

After making the desired selections, the user clicks 'Display Map' and a color-coded map of the United States that reflects their selections replaces the CDAC logo in the center pane. Titles above the map echo the parameters the user selected. The geographic boundaries shown on the map reflect the user's choice of national geography (e.g., if the user selects 'State', the map will show State boundaries).

The central map incorporates tooltips and is user-navigable. Hovering the mouse over a geographic unit generates a tooltip containing detailed information about that geographic unit. Clicking on the geographic unit will have one of two effects. If a smaller geographic unit is available, a map of just the selected geographic unit, showing the smaller unit boundaries, will replace the U.S. map (for example, if the user clicks on New York State, a county-level map of New York replaces the U.S. map). If no further drill down is possible, a trend chart showing the history of the currently-selected metric in the selected geography appears. A breadcrumb directly above the map or trend chart allows the user to return to higher geographic levels.

Below the central map, a drop-down menu invites the user to select a desired companion report using the accompanying launch button. When the user chooses a report from the drop-down and clicks the button, the requested report replaces the CDAC logo in the right-hand panel. The user ends the PAYGAR session by clicking the log off button in the left-hand pane.

APPLICATION DESIGN

Modularity is the organizing principle of PAYGAR's design. We implemented this principle by ensuring that every element of the system can be modified or augmented without modifying any application code. Instead, a series of control and data tables provide all information to stored processes that generate the user interface and produce the center and right-hand panel results. We can add new metrics to the system by simply importing the dataset with the metric values into the PAYGAR application folder and adding a row to the appropriate control table. We can add new options to the other drop-down menus, or new companion reports to be shown in the right-hand frame, just as easily. The process is always the same: ensure the asset needed exists in the application folder, and then add one or more rows to the appropriate control table(s).

DATA ARCHITECTURE

Storing information that controls application behavior is the key to PAYGAR's modularity. There are two types of datasets that PAYGAR uses: control tables and data tables. The control tables determine:

- The metrics that are available for selection
- Which subgroup types, time types, national geographies, and companion report types are available for each metric
- The name of the map file used to plot each national geography
- The drill path from one geographic level to another
- The report to show when the user reaches the final level of the drill path
- The name of the stored process associated with each companion report name

Data tables provide:

- The values for each metric
- The levels associated with each subgroup type
- Crosswalks that map one geographic level to another
- The geospatial information for displaying a given map

Figure 3 (next page) shows the overall database model for PAYGAR. The rest of this subsection explains in detail how the tables work together to provide all the information needed to run the application.

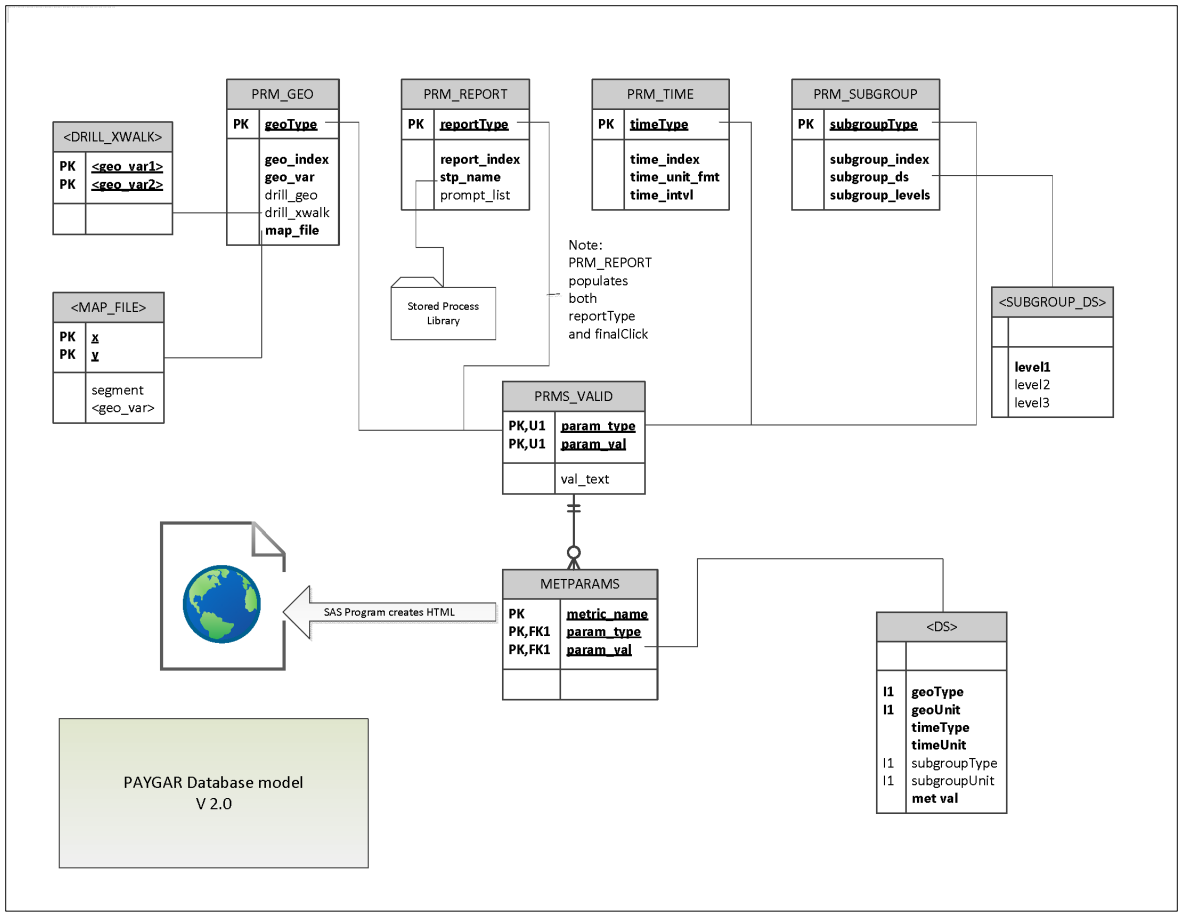

Figure 3: PAYGAR Database Model

PAYGAR tables

The control table METPARAMS is the master control table for PAYGAR. It links each metric with all the parameters available for that metric—that is, which subgroup types, time unit types, geographies, and companion reports the user can select when viewing a particular metric. Because each metric can have multiple values for any one of these parameters (e.g., a metric can have both the 'State' and 'County' geographic levels), METPARAMS is designed as a narrow table with three variables: metricName, param_type, and param_val. Each param_type relates to different aspect of PAYGAR, while param_val provides details of how the aspect is implemented. Table 1 shows the possible values of param_type and the aspects of PAYGAR each affects. Param_type values shown in italics indicate that each metricName may have only one record with that param_type; otherwise, multiple records with that param_type are permitted.

PARAM_TYPE value	**Purpose**
DS	The name of the dataset that contains the data values for the metric
Format	Specifies the format to be used when outputting metric statistics
subgroupType	Identifies a subgroup type available for a metric. Values appear as options in the associated drop-down menu.
timeType	Identifies a time unit type available for a metric. Values appear as options in the associated drop-down menu.
time_min	Identifies the earliest time unit available for a metric as a SAS date
time_max	Identifies the last time unit available for a metric as a SAS date

PARAM_TYPE value	Purpose
geoType	Identifies a geographic level available for a given metric. Values appear as options in the associated drop-down menu.
finalClick	Specifies which report to run and display in the center panel when the user reaches the end of the geographic drill path
reportType	Identifies a companion report available to the metric. Values appear as options in the associated drop-down menu.

Table 1: PARAM_TYPE Values and Purpose

The param_type values that drive drop-down menus (subgroupType, timeType, geoType, reportType) link to associated parameter control tables with additional instructions on how PAYGAR uses each parameter. The value of param_type always appears as a variable name in the associated parameter control table (e.g., the control table PRM_SUBGROUP has a variable named 'subgroup_type.') Table 2 lists the additional variables in each table and how PAYGAR employs them.

PARAM_TYPE	Control table name	Variable name	Purpose
subgroupType	PRM_SUBGROUP	subgroup_ds	The name of the dataset that specifies the subgroup levels
timeType	PRM_TIME	time_intvl	The interval between individual time units (must be valid as an interval for the SAS function INTNX)
		time_unit_fmt	The format used to write out the time unit values
geoType	PRM_GEO	map_file	The dataset that contains the geospatial information for drawing the U.S. map with the boundaries of the geographic level
		geo_var	The variable name used in the metric value dataset and map dataset that identifies the geographic unit
		drill_geo	The next level in the drill path down from this geographic level (e.g., if geoType='State', drill_geo might be 'County').
		drill_xwalk	The name of the dataset that provides the mapping from geoType to drill_geo
reportType[1]	PRM_REPORT	stp_name	The name of the stored process to run when the user selects this report from the drop-down menu

Table 2: Parameter Tables; Variables and Purpose

Four types of data table exist, each corresponding to a parameter in either METPARAMS or one of the parameter control tables. Table 3 shows the different types of data table, the parameter to which each corresponds, and its function in the system.

Data table type	Related parameter	Purpose
METRIC DATASET	Value of param_val where param_type="DS" in METPARAMS	Contains values for a metric. One record per combination of time type, time unit, subgroup type, subgroup level, geography type, and geography level.
MAP DATASET	Value of map_file in PRM_GEO	Map dataset to be used by PROC GMAP. Must contain projected x/y coordinates and a geographic identifier.
MAP CROSSWALK	Value of drill_xwalk in PRM_GEO	Defines how to move between two geographic levels (e.g., State to County)
SUBGROUP DATASET	Value of subgroup_ds in PRM_SUBGROUP	Names the levels available for a particular subgroup type. For example, the subgroup dataset set for the 'Medicare Part' subgroup might contain records *Part A, Part B, Part D*.

Table 3: Data Tables; Related Parameters and Purpose

[1] finalClick also references the PRM_REPORT parameter control table to obtain the stored process name.

A set of stored processes handle all changes to the PAGYAR database. Each parameter table in the PAYGAR database has a corresponding stored process that we use to manage all modifications to that table. Collectively, these load utilities enforce database integrity rules to ensure that only valid data is available to PAYGAR. For instance, when adding a new geographic level to PAYGAR, the load utility ensures that related map file and crosswalk table (if applicable) exist in the application folder and that the map file has appropriately-named variables. Another stored process collects all values from the parameter tables into a single dataset PRMS_VALID. We use yet another stored process to choose parameters from PRMS_VALID to associate with metrics in METPARAMS.

CONSTRUCTING THE USER INTERFACE

A detailed discussion of web design exceeds the scope of this paper, but we will explain how we use SAS to create the web code behind the user interface. PAYGAR relies on the _webout destination in the Stored Process Web Application to create the user interface. The _webout destination captures streaming output from a stored process and presents it as a web page. Hence, the web page is recreated each time the stored process runs. The instructions for creating the web page are contained in a SAS dataset called DS_PGAR_HTMLCODE. This dataset has a single variable *line*, which generally contains one HTML, CSS, or Javascript statement per record. The following code, saved as the stored process SP_PGAR_MAIN, uses this dataset to create the webpage when the user navigates to the stored process URL (see the Stored Process section for more details on stored process URLs).

```
libname pgar_shr base '/sasdata/projects/oig/paygar/share';

*ProcessBody;

data _null_;
set pgar_shr.ds_pgar_htmlcode;
file _webout;
put line;
run;

%stpbegin;
%stpend;
```

Note that the stored process macros appear after the DATA _NULL_ step. The %stpbegin macro ties the _webout destination to ODS, so calling it as the argument of a file statement after %stpbegin will generate the error message **ERROR: File is in use, _WEBOUT.**

There are two components to the web code in DS_PGAR_HTMLCODE, framework and data. The framework consists of HTML and Javascript statements that create the layout and style of the page, and Javascript functions that provide the functionality. The framework remains static within a given version of PAYGAR and hence is stored in a permanent location. The data, on the other hand, changes any time we add a new element to PAYGAR. Therefore, we generate these statements dynamically from the PAYGAR database (see Appendix A for details on this process).

STORED PROCESS ARCHITECTURE

PAYGAR's functionality comes from the ability to chain stored processes together in the Stored Process Web Application. Chaining stored processes means simply passing parameters from one stored process to another. The 'receiving' stored process must have prompts defined to accept the parameters from the 'sending' process. Figure 5 shows how PAYGAR chains stored processes to create the full application.

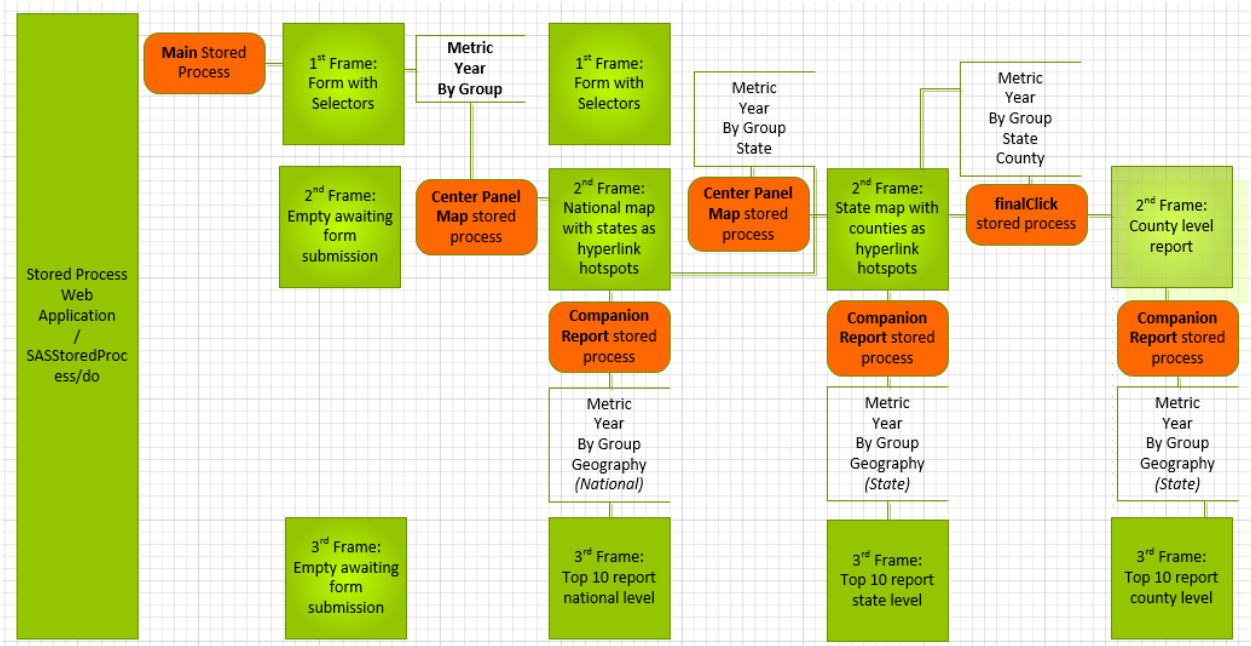

Figure 4: Architecture of chained stored processes in PAYGAR

Chaining stored processes takes places in two ways in PAYGAR. The left-hand menu and the companion report menu both use an HTML form to gather parameters from the user to and initiate the chained stored process via a submit button. The center panel stored process, on the other hand, associates a character variable containing the appropriate parameters with each selectable geographic region via the HTMLVAR= options on the CHORO statement of the PROC GMAP. In both cases, the Stored Process Web Application uses the information to build a URL that launches a chained stored process and provides the correct results.

A typical URL in PAYGAR appears below, which is the URL constructed for Georgia on a National Map with State-level geographic boundaries with the following user selections: metric=Total Payments, subgroup type=Service category, subgroup unit=Ambulance: BLS, time unit type=Year, time unit=2012. Question marks and ampersands separate components of the URL, each of which we explain below. Note that we show this URL as an example only; the actual server is at a different URL.
https://webserver.cms.gov/SASStoredProcess/do?_program=SP%20PGAR%20Main&_action=nobanner,notimer&_GOPT_DEVICE=ACTIVEX&geoTypeHistory=Nation,State&geoUnitHistory=Nation,13&metricName=Total%20payments&subgroupType=Service%20category&subgroupUnit=Ambulance%3A%20BLS&timeType=Year&timeUnit=18993&geoType=County

- https://webserver.cms.gov/SASStoredProcess/do? – the web address of the Stored Process Web Application
- _program=SP%20PGAR%20Main – the chained stored process to launch
- &_action=nobanner,notimer – execution parameters for the stored process (see Table 4)
- &_GOPT_DEVICE=ACTIVEX – device the chained stored process will use to display graphical content. This is an environment parameter in all stored processes.
- &geoTypeHistory=Nation,State and remainder – the input parameters to the stored process that reflect the user's menu selections or the selected geography on the map

Whether constructed from a menu or as part of PROC GMAP, the HTML code will also have a target attribute associated with each URL. The target attribute tells PAYGAR where to display the results the Stored Process Web Application provides.

Table 4 shows a selection of action parameters that can be submitted to the Stored Process Web Application. PAYGAR extensively uses the NOBANNER and NOTIMER actions in all of its stored process calls. A full list of Stored Process features is located at: http://support.sas.com/documentation/cdl/en/stpug/64882/HTML/default/viewer.htm#p184mqqbi9w6qjn1q0619x19eg02.htm

_ACTION=	Description
INDEX	Displays a list of stored processes
PROPERTIES	Displays the prompt page for a stored process when selected
EXECUTE	If the selected stored process has no prompts, this will allow it to execute
BACKGROUND	Allows stored process to run in background (for longer than 1 minute)
NOBANNER	Suppresses the SAS SPWA banner at top of page
NOTIMER	Suppresses the SAS timer throughout the stored process
DATA	Displays a summary of stored process data

Table 4: SPWA actions and descriptions

CONCLUSION

SAS® Enterprise Business Intelligence provides a powerful platform for constructing custom Business Intelligence applications. Many intricacies exist that exceed the scope of this paper, but a solid plan for chaining stored processes and creating the data assets to support them is critical to success. Do not be afraid to experiment! Several of the most elegant aspects of PAYGAR's design were developed only after numerous trials (and errors).

ACKNOWLEDGMENTS

The authors would like to acknowledge OIG and CMS management for their support of this project.

RECOMMENDED READING <HEADING 1>

- *Building Business Intelligence Using SAS®: Content Development Examples (Aanderud & Hall)*
- *w3Schools Online Web Tutorial (w3schools.com)*

CONTACT INFORMATION <HEADING 1>

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

Scott Hutchison
Department of Health and Human Services
Office of Inspector General
Consolidated Data Analysis Center (CDAC)
7500 Security Blvd,Mail Stop N2-02-08
Baltimore, MD 21244
Office: 410-786-0221
Fax: 410-786-5305
Email: scott.hutchison@oig.hhs.gov

John L. Venturini
Piper Enterprise Solutions
8801 Fast Park Drive
Suite 111
Raleigh, NC 27617
Office: (919) 659-8255
E-mail: john.venturini@sas.com

APPENDIX A: CONVERTING DATA TO JAVASCRIPT CODE

We rely on Javascript to transform the data in PAYGAR tables into the drop-down menus. The key construct in this process is the Javascript array. Javascript arrays are simply variables that can contain multiple values. They are similar to SAS arrays in some respects, but have some differences. Most importantly for PAYGAR, instead of storing variable names, Javascript arrays store *objects*—which can be data values, variables, functions, user-created objects, or other arrays. Several methods exist to declare Javascript arrays. We employ two different declaration styles in our code (see below). No matter the declaration method, array elements are referenced by the element's index number, which starts at 0. Hence, `myArray[1]` references the *second* element of the array object `myArray`.

- Method 1:

```
var myArray=new Array              //Create a new Array object
myArray[0]=”Value 1”;              //Assign a value to the first array element
myArray[1]=”Value 2”;              //Assign a value to the second array element
```

- Method 2:

```
var myArray=new Array(“Value 1”,   //Create a new Array object with two elements
”Value 2”);
```

Using information from the PAYGAR control and data tables, we define the Javascript function 'createArray' that, when called, creates the array needed to populate a particular drop-down menu. This function takes up to three parameters that specify the array to be constructed: 'arraySource' is the name of the drop-down menu to be populated, 'metricIndex' identifies the metric that the user has selected, and 'typeValue' defines the subgroup or time type that the user has selected. The PAYGAR webpage calls the function with just the arraySource parameter set, with value 'metricName,' when the page first loads. This creates the metric selection drop-down. The page then calls the function again to populate the subgroup type, time type, and geography type drop-downs when the user selects a metric, this time with both the arraySource and metricIndex parameters. When the user selects a subgroup or time type, the function is invoked again, with all parameters specified, to create the subgroup and time unit drop-downs. A simplified version of the createArray function code appears below:

```
function createArray(arraySource,metricIndex,typeValue){
var arr=new Array();
if (arraySource=="metricName") {
    arr[0]="Disproportionate payments-2x";
    arr[1]="Total payments";
    }

if (arraySource=="subgroupType") {
    if (metricIndex=="Total payments") {
           arr[0]="Service category";
           }
    else if (metricIndex=="Disproportionate payments-2x") {
           arr[0]="Medicare part";
           arr[1]="Service category";
           }
    }

if (arraySource=="subgroupUnit") {
    if (typeValue=="Medicare part") {
           arr[0]=”Part A”;
           arr[1]=”Part B”;
           arr[2]=”Part D”;
           }
    if (typeValue=="Service category") {
           arr[0]="Ambulance: BLS";
           arr[1]="Physical therapy";
           }
    }

return arr;
}
```

We use the PAYGAR control and data tables to create the Javascript statements that construct the createArray function. This simplified example uses the subgroup type and subgroup level to explain the process. Four tables, as shown below, contribute to this example.

metricName	param_type	param_val
Disproportionate payments-2x	subgroupType	Service category
Disproportionate payments-2x	subgroupType	Medicare part
Total Payments	subgroupType	Service category

Example table 1: METPARAMS

subgroupType	Subgroup_ds
Medicare part	DSSUB_PGAR_MCARETYPE
Service category	DSSUB_PGAR_AHRQ

Example table 2: PRM_SUBGROUP

level1
Ambulance: BLS
Physical therapy

Example table 3: DSSUB_PGAR_AHRQ

level1
Part A
Part B
Part D

Example table 4: DSSUB_PGAR_MCARETYPE

Several SAS DATA steps transform these values in to records in the DS_PGAR_HTMLCODE datasets that provides the instructions for the web page. The first lines are straightforward and serve to open the function:

```
Data array_code;
length line $300;

line='function createArray(arraySource,metricIndex,typeValue) {';
output;
line='var arr=new Array();';
output;
run;
```

Now we leave the function definition open and query METPARAMS to get the values with which to populate the array:

```
Data newlines(keep=line);
set pgar_shr.metparams end=last;
where param_type="DS"; /*Ensure that we have only one record per metricName*/
by metricName;
length line $300;

if _n_=1 then do;
    line='if (arraySource=="metricName") {';
    output;
    array_index=0;
    end;

if first.metricName then do;
    line='arr['||strip(array_index)|| ']= "'||strip(metricName)|| ' ";';
    array_index+1;
```

```
    output;
    end;

if last then do;
    line='};';
    output;
    end;

run;

Data array_code;
set array_code newlines;
run;
```

Next, we query METPARAMS again to get the subgroupTypes associated with each metricName:

```
Data newlines(keep=line);
set pgar_shr.metparams end=last;
where param_type="subgroupType";
length line $300;
by metricName param_val;
line='if (arraySource=="subgroupType") {';
output;

if first.metricName then do;
    line='if (metricIndex=="'||strip(metricName)||'") {';
    output;
    array_index=0;
    end;

line='arr['||strip(array_index)||']="'||strip(param_val)||'";';
output;
array_index+1;

if last.metricName then do;
    line='}';
    output;
    end;

if last then do;
    line='}';
    output;
    end;
run;

Data array_code;
set array_code newlines;
run;
```

Next, we query the PRM_SUBGROUP dataset to get the names of the subgroup datasets associated with each type, and query those to get the levels to populate the drop-down.

```
Data newlines(keep=line);
set prm_subgroup end=last;
length line $300;

if _n_=1 then do;
    line='if (metricIndex=="subgroupUnit") {';
    output;
    end;

line='if (typeValue=="'||strip(subgroupType)'") {';
output;
```

```
dsid=open(subgroup_ds);              /*Open the subgroup levels dataset*/
if dsid>0 then do;
    obscnt=0;                        /*Initialize observation count*/
    myvar=varnum(dsid,'level1');     /*Get the variable number*/
    do until(thisobs^=0);            /*Cycle through all observations*/
           thisobs=fetch(dsid);
           if thisobs=0 then do;     /*If observation exists, create line*/
                  line='arr['||strip(obscnt)||']="'
                         ||strip(getvarc(dsid,myvar))||'";';
                  output;
                  obscnt+1;          /*Iterate observation count*/
                  end;
           end;
    rc=close(dsid);
    end;

line='}';
output;

if last then do;
    line='}';
    output;
    end;

run;

Data array_code;
set array_code newlines;
run;
```

Finally, we close the function.

```
data array_code;
set array_code end=last;
output;
if last then do;
    line='}';
    output;
    end;
run;
```