Paper 045-2010

Interactive Dashboards: Powered by Flash and the SAS® Programmer

Michael Thomas and Gordon Hirsch, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® BI Dashboard 4.3 introduces a new feature: a stored process can act as a data source for a dashboard indicator. This paper explores how SAS® programmers can use this new feature to provide highly interactive dashboards based on targeted analytics that only SAS can provide. Two examples are explored. First, a stored process is used to trigger an alert based on the shape of example time series data, instead of just a simple threshold. This approach uses SAS to go far beyond basic exception reporting in to analytics-based exception processing that can achieve excellent signal-to-noise ratios and be a credible tool for busy decision makers. Secondly, parameterized stored processes are used as the data sources for a parameterized dashboard. The parameterized dashboard takes advantage of Flash to give the end user a highly interactive experience, while the stored processes deliver real analytical insight and therefore real value.

INTRODUCTION

SAS® BI Dashboard 4.3 introduces the power of Adobe Flash to SAS® Business Analytics. Flash provides a high level of interactivity that makes the presentation of key metrics more intuitive to business users. BI Dashboard 4.3 also introduces new ways of leveraging SAS® Stored Processes, and makes it easy to express the power of SAS through an engaging and easy-to-use user interface. This paper introduces Adobe Flash and explains how BI Dashboard 4.3 makes use of this technology to create highly interactive and responsive dashboards. Then, the SAS Stored Process Data Provider is examined. The SAS Stored Process Data Provider enables stored processes to feed data into a dashboard in such a way that Flash can be used for presentation and interactivity. This paper concludes by examining how stored processes can be used with the alerting functionality of BI Dashboard so that key users know when to examine their dashboards or other presentation of data. Finalized names of new components and features in the production version of BI Dashboard might differ from those in this paper.

ADOBE FLASH AND BUSINESS DATA PRESENTATION

Adobe Flash has been part of the Internet ecosystem for some time. The technology underlies the YouTube video player as well as much Web site content, including games and advertisements. The Flash player is pervasive enough that it can generally be assumed to be on a browser. With the introduction by Adobe of the Flex APIs and Flex Builder, the development of business applications for the Flash player became more reasonable and cost-effective. Many business software vendors have delivered or plan to deliver products that use Flash, and SAS is incorporating Flash technology in many of its products, including BI Dashboard 4.3.

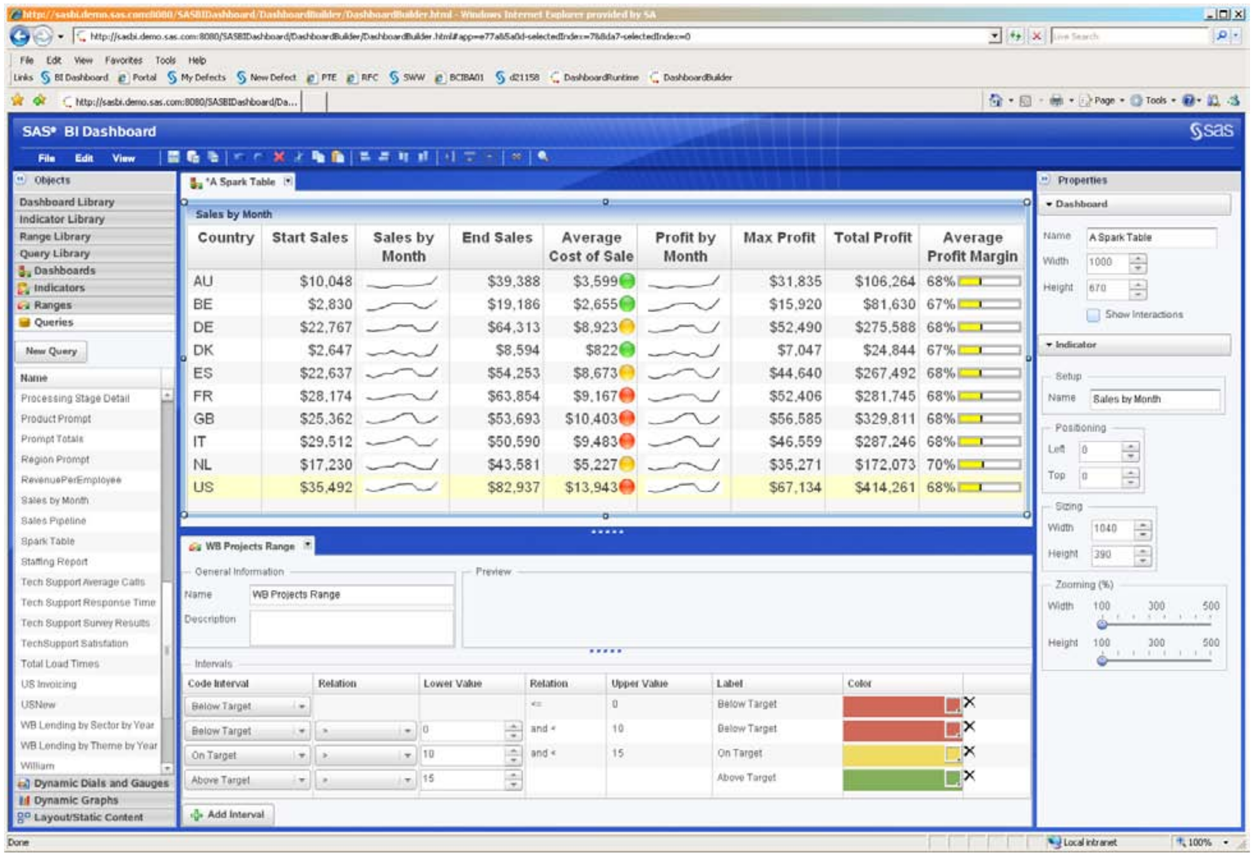
Like many other business software products, BI Dashboard 4.3 leverages Flash in the rich Internet application (RIA) architecture. The RIA architecture has been named and described by Adobe, though the basic principles can and have been realized with other client technologies other than Flash, including AJAX and Java applets. The key value of RIAs is that, unlike a Web page model, the browser has embedded client logic that enhances responsiveness and interactivity while maintaining the administrative advantages of a thin-client architecture. Each of these concepts should be considered separately to understand how they work together to provide the value added by RIAs:

- **Interactivity.** The ability to interact with visual objects on the display in meaningful ways, including hyperlinking and drill-down. Web applications are interactive, even if the response might be slow. PDF documents that are meant to closely emulate paper documents, such as legal contracts, are minimally interactive.
- **Responsiveness.** The speed of interactions in an application. Traditional Web applications typically require a Web page load for interactivity. For example, the user fills out a form and hits submit, the browser waits for a response from the server, and then the response HTML is displayed. All of this could take several seconds, even for relatively trivial interactions. The RIA approach leverages client-side modules to respond to interactions much more quickly.
- **Thin-client architecture.** Responsiveness and interactivity are hallmarks of client/server applications, but the desktop clients are difficult to administer and deploy across the enterprise. Though Flash RIAs aren't strictly zero download — the Flash player is a download to the browser — the Flash player is pervasive enough that Flash RIAs have the characteristics of thin-client architecture.

These concepts combine together to yield rich and responsive applications that express centralized data and can be easily deployed and maintained in a thin-client architecture across the enterprise.

BI DASHBOARD 4.3 & FLASH

BI Dashboard 4.3 uses Flash and the RIA model to create a rich, cutting-edge experience for both dashboard users and dashboard designers. BI Dashboard 4.3 has a Flash Dashboard Display Environment as well as a Dashboard Builder for dashboard designers. The Dashboard Builder is shown in Figure 1 below. It is a what you see is what you get (WYSIWYG) dashboard builder that embeds the Dashboard Display Environment. As with all of BI Dashboard 4.3, the highly interactive environment is available in the context of thin-client architecture. This means that it is easy for any user in the enterprise to build dashboards without application-specific deployment and maintenance — any user with a browser that can watch YouTube videos is also capable of building dashboards.



Sales by Month

Country	Start Sales	Sales by Month	End Sales	Average Cost of Sale	Profit by Month	Max Profit	Total Profit	Average Profit Margin
AU	$10,048		$39,388	$3,599 🟢		$31,835	$106,264	68%
BE	$2,830		$19,186	$2,655 🟢		$15,920	$81,630	67%
DE	$22,767		$64,313	$8,923 🟡		$52,490	$275,588	68%
DK	$2,647		$8,594	$822 🟢		$7,047	$24,844	67%
ES	$22,637		$54,253	$8,673 🟡		$44,640	$267,492	68%
FR	$28,174		$63,854	$9,167 🔴		$52,406	$281,745	68%
GB	$25,362		$53,693	$10,403 🔴		$56,585	$329,811	68%
IT	$29,512		$50,590	$9,483 🔴		$46,559	$287,246	68%
NL	$17,230		$43,581	$5,227 🟡		$35,271	$172,073	70%
US	$35,492		$82,937	$13,943 🔴		$67,134	$414,261	68%

WB Projects Range

General Information
Name: WB Projects Range
Description:

Preview

Intervals

Code Interval	Relation	Lower Value	Relation	Upper Value	Label	Color
Below Target			<=	0	Below Target	🟥
Below Target	>	0	and <	10	Below Target	🟥
On Target	>	10	and <	15	On Target	🟨
Above Target	>	15			Above Target	🟩

Properties
Dashboard
Name: A Spark Table
Width: 1000
Height: 670
Show Interactions

Indicator
Setup
Name: Sales by Month
Positioning
Left: 0
Top: 0
Sizing
Width: 1040
Height: 390
Zooming (%)
Width: 100
Height: 100

Figure 1 Dashboard Builder

The Dashboard Builder embeds the Flash Dashboard Display Environment. There are different Dashboard Display Environments for different devices with different capabilities. Dashboards for all of the Dashboard Display Environments are all built from the Flash Dashboard Builder.

Dashboard designers view dashboards as they are developing them. While an important part of designing dashboards is the visual layout, dashboard designers do a lot more than just layout. They also build the visualizations, describe business rules and map data presentation to data queries, which can include the execution of SAS code in the form of stored processes. So while the discussion of stored processes and BI Dashboard is left for the next section, it's worth noting at this point that designers engage with stored processes via the Dashboard Builder at least as much as dashboard consumers. As the user builds dashboards, they experience the benefits of the Flash RIA platform, including drag-and-drop functionality, desktop feel in editing objects and easy deployment.

The Flash Dashboard Display Environment leverages the Flash RIA platform in many ways, including:

- **Bird's-eye view.** Bird's-eye view allows the designer to stuff the dashboard with more content than can be seen at normal sizes, but still large enough to attract the eye to the most relevant items.

- **Data brushing.** When the user interacts with data values in one visualization or table, the analogous data values in another visualization or table are highlighted, allowing the user to interactively see relationships.
- **Loosely coupled eventing.** No commonality between different interactive elements in a dashboard is required. Data can be from different data sources and different types of data sources. For example, In a single interactive dashboard, different indicators could be from SQL queries against an operational system, SAS Information Maps against a data mart and stored processes that provide analytic data. They all tie together at the dashboard level via the Dashboard Builder, and the designer maps how parameters are passed. For example, region_id from one indicator can be mapped to SalesRegionId in another indicator.
- **Data caching for responsiveness.** The Flash Dashboard Display Environment has an embedded data cache so that the dashboard can be more responsive to interactions.

The screenshot in Figure 2 shows a dashboard that uses data brushing, bird's-eye view and loosely coupled eventing.

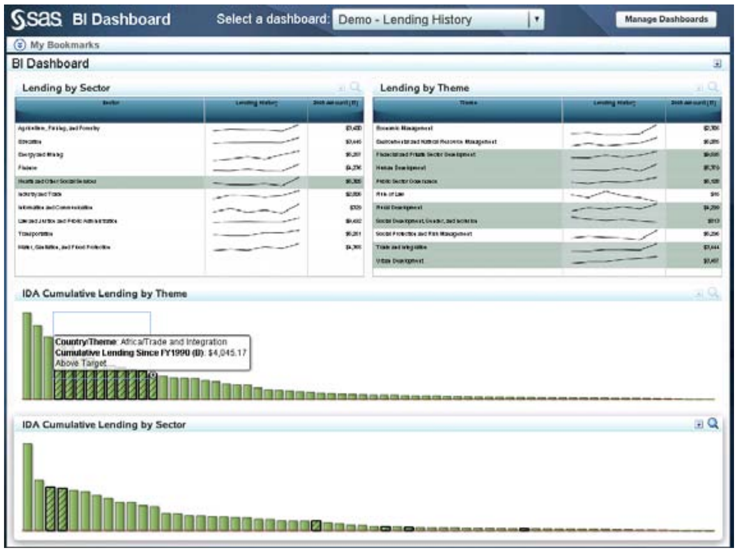

BI Dashboard
Select a dashboard: Demo - Lending History
Manage Dashboards
My Bookmarks
BI Dashboard
Lending by Sector
Lending by Theme
IDA Cumulative Lending by Theme
Country/Theme: Africa/Trade and Integration
Cumulative Lending Since FY1990 (B): $4,045.17
Above Target
IDA Cumulative Lending by Sector

Figure 2 Example Flash Dashboard

STORED PROCESSES AND BI DASHBOARD

Stored processes are a way to invoke SAS code in a manner that fits in with Web application and n-tier architectures. Stored processes can be generated by many products, including SAS® Enterprise Guide®. It is beyond the scope of this paper to fully discuss stored processes, but for the purposes of this paper they can be thought of as a way to wrap SAS code so that it can be invoked by various components and products, including BI Dashboard.

Stored processes have many different types of output. BI Dashboard consumes two different types of stored process output:

- data, in the form of a data set contained in a package
- visualizations, in the form of images in common image formats (png, jpg, gif, and so on)

The next two sections discuss how stored processes can generate these two different types of output. Each different output provides value in different ways. With data output, BI Dashboard can take analytic output and handle all the aspects of presentation, including integration with other visual objects in the dashboards, such as other graphs and prompts. With visualization output, the breadth of visualizations that can be created by SAS can be included in a Flash dashboard.

STORED PROCESSES AS DATA SOURCES

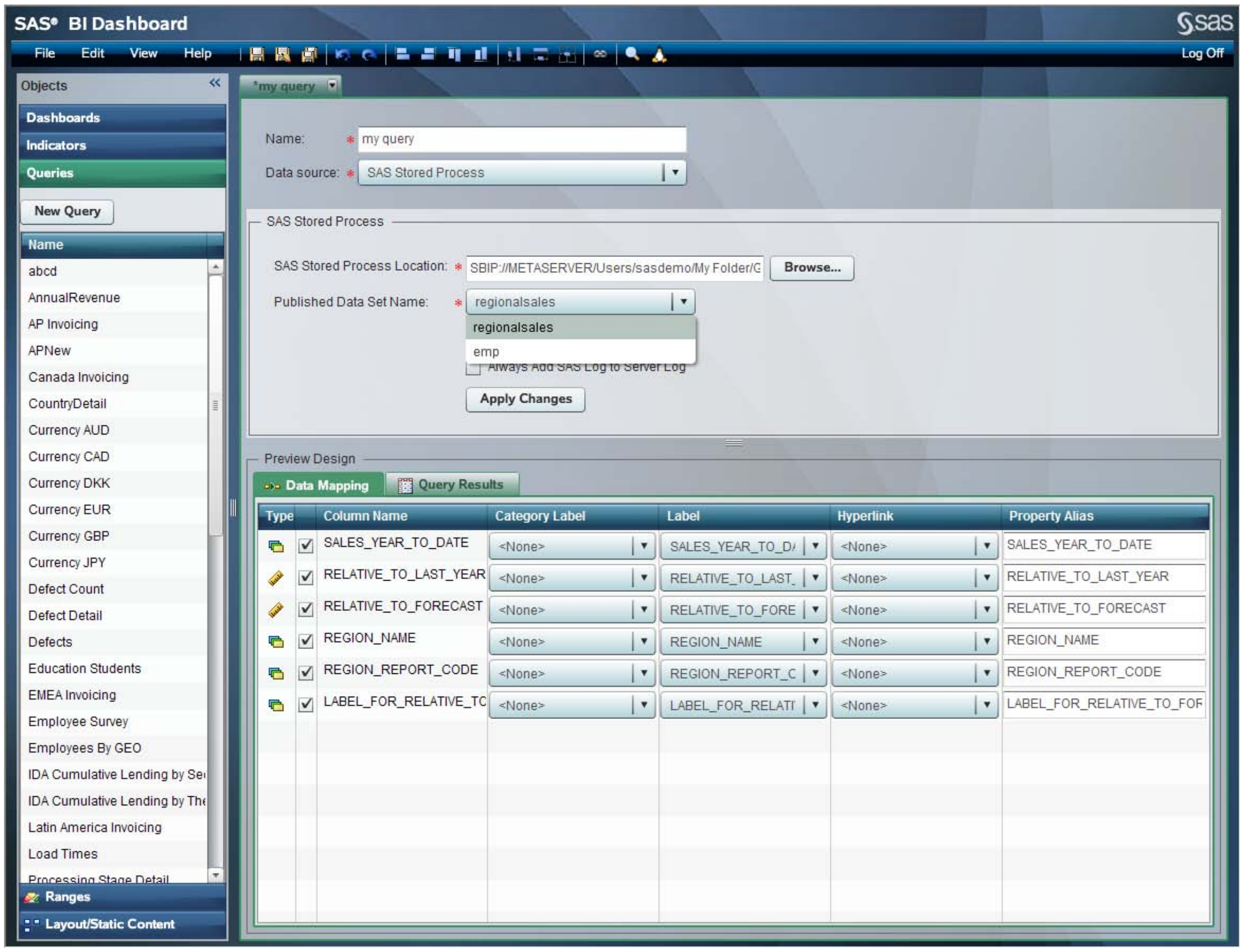
The following two segments of SAS code shows the basics of populating a package from a stored process. This section of code is only for reference — it sets up the stored process and a macro that is used later.

```
%global _ARCHIVE_PATH _ARCHIVE_NAME;
*ProcessBody;

%macro checkrc(text);
   if rc ne 0 then do;
      msg=sysmsg();
      put msg=;
      end;
   else put "&text succeeded";
%mend;
```

For the following code, assume that you have two data sets named work.emp and work.regionalsales. This code creates the package and inserts the data sets.

```
data _null_;
rc    = 0;
pid   = 0;
desc  = "Two data sets in a package";

putlog "&foo";

desc = trim(desc);

Call package_begin(pid, desc, nameV, rc);
%checkrc(Package init);

Call insert_dataset(pid, "WORK", "emp", "22 persons",
                       '', rc);
%checkrc(Sample dataset);

Call insert_dataset(pid, "WORK", "regionalsales", "Regional Sales",
                       '', rc);
%checkrc(Sample dataset);

length fullpath $4096;

Call package_publish(pid, "TO_ARCHIVE", rc,
                        "archive_path, archive_name, archive_fullpath",
                        "&_ARCHIVE_PATH", "&_ARCHIVE_NAME", fullpath);
%checkrc(Publish to archive);
call symput('_ARCHIVE_FULLPATH', fullpath);

Call package_end(pid, rc);
%checkrc(Package term);

run;
```

In BI Dashboard, the dashboard designer creates a data query based on the stored process. The designer creates a new query, picks the Stored Process Data Provider as the data source, and then picks the stored process by navigating the SAS folders to find it. Since this stored process has multiple data sets in the package, the designer chooses the data set. From there, the user can configure different settings of the query. Since the designer can reasonably accept the defaults, the discussion of the settings is left to BI Dashboard documentation.



Name
abcd
AnnualRevenue
AP Invoicing
APNew
Canada Invoicing
CountryDetail
Currency AUD
Currency CAD
Currency DKK
Currency EUR
Currency GBP
Currency JPY
Defect Count
Defect Detail
Defects
Education Students
EMEA Invoicing
Employee Survey
Employees By GEO
IDA Cumulative Lending by Se
IDA Cumulative Lending by The
Latin America Invoicing
Load Times
Processing Stage Detail

Name: * my query

Data source: * SAS Stored Process

SAS Stored Process

SAS Stored Process Location: * SBIP://METASERVER/Users/sasdemo/My Folder/G

Published Data Set Name: * regionalsales (regionalsales, emp)

Always Add SAS Log to Server Log

Preview Design

Type		Column Name	Category Label	Label	Hyperlink	Property Alias
	✓	SALES_YEAR_TO_DATE	<None>	SALES_YEAR_TO_D/	<None>	SALES_YEAR_TO_DATE
	✓	RELATIVE_TO_LAST_YEAR	<None>	RELATIVE_TO_LAST_	<None>	RELATIVE_TO_LAST_YEAR
	✓	RELATIVE_TO_FORECAST	<None>	RELATIVE_TO_FORE	<None>	RELATIVE_TO_FORECAST
	✓	REGION_NAME	<None>	REGION_NAME	<None>	REGION_NAME
	✓	REGION_REPORT_CODE	<None>	REGION_REPORT_C	<None>	REGION_REPORT_CODE
	✓	LABEL_FOR_RELATIVE_TC	<None>	LABEL_FOR_RELATI	<None>	LABEL_FOR_RELATIVE_TO_FOF

Figure 3 Query Based on Stored Process

With the query built, the designer can move on to create an indicator such as the one shown in Figure 4 below: The process for creating an indicator after creating a query is the same regardless of data source — the designer would go through the same steps for stored process output as they would for SQL query or information map output. The details of how to create the indicators and then put them in to dashboards is left to the product documentation of BI Dashboard 4.3. But briefly, the designer assigns data columns to the roles needed by the visualization. If applicable, the designer selects a range to apply to gauge displays. A range is a set of intervals where each interval has some business classification, such as Below Target or Above Quota, and a color, such as red, yellow or green.

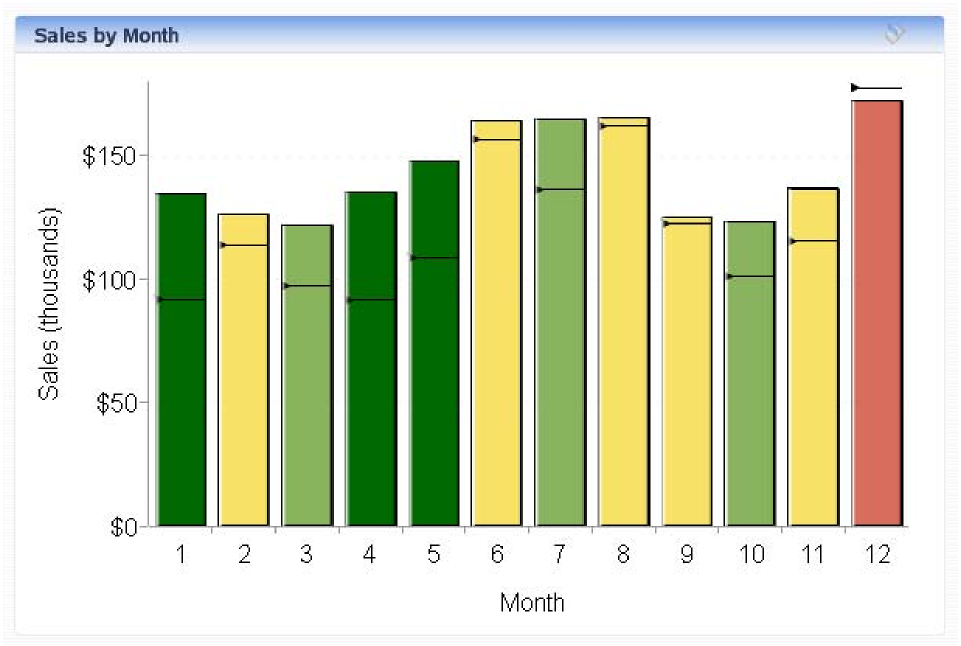
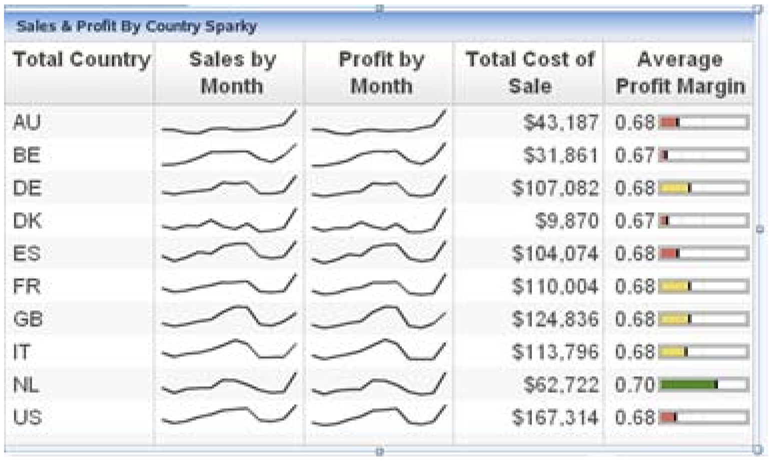


Figure 4 Targeted Bar Indicator

The screenshot in Figure 5 shows another type of indicator display possible with BI Dashboard. It uses sparklines as well as gauges to convey different insight in a compact form. As with any indicator, this indicator can source its data from a stored process.



Sales & Profit By Country Sparky

Total Country	Sales by Month	Profit by Month	Total Cost of Sale	Average Profit Margin
AU			$43,187	0.68
BE			$31,861	0.67
DE			$107,082	0.68
DK			$9,870	0.67
ES			$104,074	0.68
FR			$110,004	0.68
GB			$124,836	0.68
IT			$113,796	0.68
NL			$62,722	0.70
US			$167,314	0.68

Figure 5 Tabular Indicator with Sparklines and Gauges

The next stored process, generated originally by Enterprise Guide but edited for brevity, takes parameters and can be used in conjunction with BI Dashboard loosely coupled eventing. The stored process takes a parameter, Country, and uses it in the WHERE clause of a PROC SQL statement:

```
*ProcessBody;

%macro checkrc(text);
   if rc ne 0 then do;
      msg=sysmsg();
      put msg=;
      end;
   else put "&text succeeded";
%mend;

%global COUNTRY _ARCHIVE_PATH _ARCHIVE_NAME;

/*%STPBEGIN;*/

OPTIONS VALIDVARNAME=ANY;

/*Libname WORLDBNK META  libid=A5976BVY.AY000005;*/

*  End EG generated code (do not edit this line);

/* --- Start of shared macro functions. --- */

/* Conditionally delete set of tables or views, if they exist          */
/* If the member does not exist, then no action is performed   */
%macro _eg_conditional_dropds /parmbuff;
```

(The code for this macro isn't included in order to keep the code brief. It is code generated by Enterprise Guide and is easily recreated.)

```
%mend _eg_conditional_dropds;

%macro _eg_WhereParam( COLUMN, PARM, OPERATOR, TYPE=S, MATCHALL=_ALL_VALUES_,
MATCHALL_CLAUSE=1, MAX= );
```

(The code for this macro isn't included in order to keep the code brief. It is code generated by Enterprise Guide and is easily recreated.)

```
%mend;

%_eg_conditional_dropds(WORK.QUERY_FOR_COUNTRYDETAIL);
```

The PROC SQL statement uses the Country parameter to select only rows for the particular country. If there is no value for Country, then the SQL statement will return no rows. The Dashboard Builder will work better for the dashboard designer when the queries return rows, because the designer can see changes to visuals as they design. To help the dashboard design experience, it's best to set default values for any parameters as part of the stored process set up in metadata.

```
PROC SQL;
   CREATE TABLE WORK.QUERY_FOR_COUNTRYDETAIL AS
   SELECT *
      FROM SASDATA.COUNTRYDETAIL AS t1
      WHERE %_eg_WhereParam( t1.country, Country, EQ, TYPE=S );
QUIT;
```

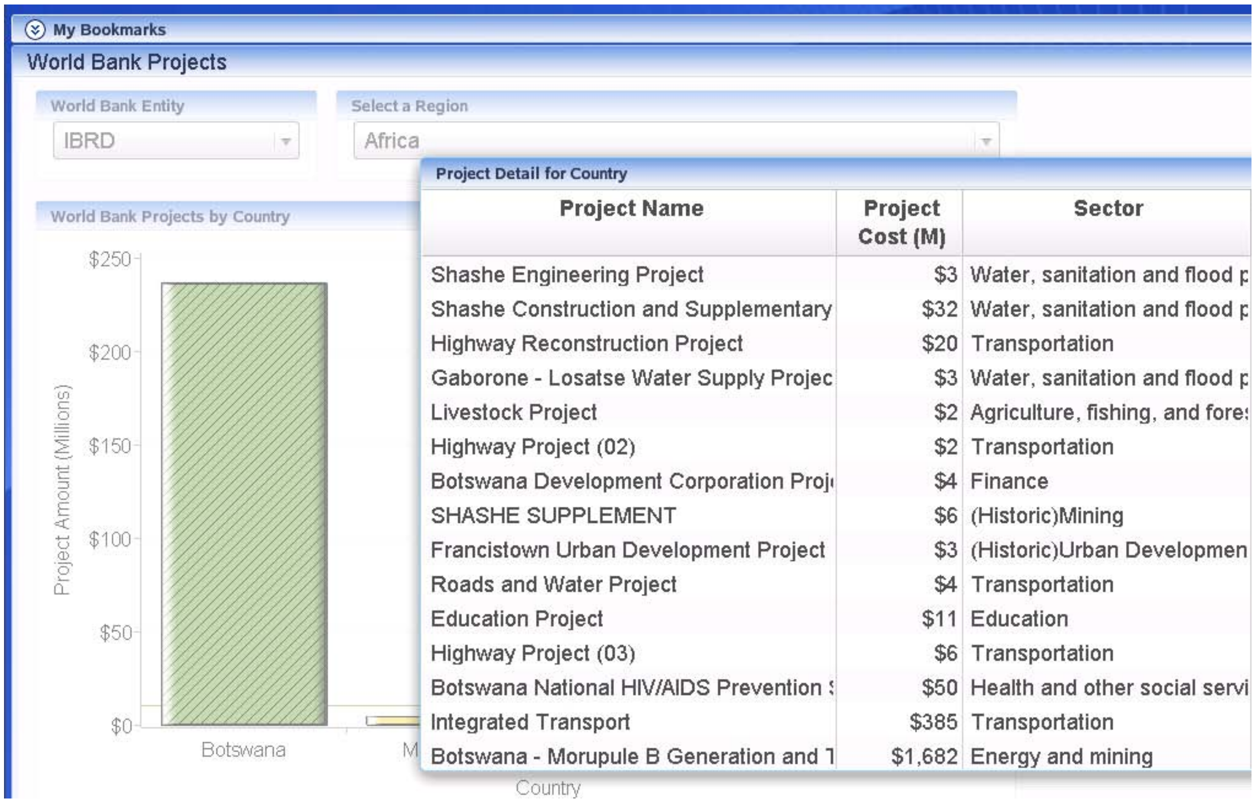
The remaining code takes the data set and adds it to the package.

```
data _null_;
rc    = 0;
pid   = 0;
desc  = "";

call package_begin(pid, desc, nameV, rc);
```

```
%checkrc(Package init);

call insert_dataset(pid, "WORK", "QUERY_FOR_COUNTRYDETAIL", "Country Detail",
'', rc);
%checkrc(Package data set);

length fullpath $4096;

call package_publish(pid, "TO_ARCHIVE", rc,
                        "archive_path, archive_name, archive_fullpath",
                        "&_ARCHIVE_PATH", "&_ARCHIVE_NAME", fullpath);
%checkrc(Package publish);

call symput('_ARCHIVE_FULLPATH', fullpath);

call package_end(pid, rc);
%checkrc(Package term);
run;

/* --- End of code for "Query Builder". --- */

*  Begin EG generated code (do not edit this line);
;*';*";*/;quit;
/*%STPEND;*/

*  End EG generated code (do not edit this line);
```

This stored process only publishes a single data set to the package. The dashboard designer sets up the query to the stored process as before, and then sets up the indicator based on the query. From here, the designer sets up interactions at the dashboard level. The designer has a couple of options with an indicator based on a parameterized stored process:

- Set up an event linkage between one or more prompts and the indicator, so that the indicator content is filtered. For the dashboard consumer, this means that changing a prompt value will immediately trigger a re-query of the indicator, which re-runs the stored process with new values for its parameters.
- Set up a pop-up link from another indicator object to the stored process indicator. When the user clicks on an item in another indicator, such as a bar in a bar chart or a particular gauge, then the stored process indicator is displayed in a pop-up window, based on the relevant parameter. This is shown in Figure 6 below, where the pop-up table is based on the stored process above. The table is displayed after selecting on the bar of the indicator. That indicator was driven by parameters from the two drop-down prompts, which are also considered indicators because they can visualize business rules just like any other type of indicator. The screenshot shows a total of four different indicators, all of which are decoupled from each other. They could all be based on stored processes, or they might all be based on different types of data sources, including SQL queries, information maps or other SAS products, including SAS® Strategy Management and SAS® Human Capital Management. They could all be drawing from the same database, or from entirely different data locations.



World Bank Projects

World Bank Entity: IBRD

Select a Region: Africa

Project Detail for Country

Project Name	Project Cost (M)	Sector
Shashe Engineering Project	$3	Water, sanitation and flood p
Shashe Construction and Supplementary	$32	Water, sanitation and flood p
Highway Reconstruction Project	$20	Transportation
Gaborone - Losatse Water Supply Projec	$3	Water, sanitation and flood p
Livestock Project	$2	Agriculture, fishing, and fore:
Highway Project (02)	$2	Transportation
Botswana Development Corporation Proj	$4	Finance
SHASHE SUPPLEMENT	$6	(Historic)Mining
Francistown Urban Development Project	$3	(Historic)Urban Developmen
Roads and Water Project	$4	Transportation
Education Project	$11	Education
Highway Project (03)	$6	Transportation
Botswana National HIV/AIDS Prevention S	$50	Health and other social servi
Integrated Transport	$385	Transportation
Botswana - Morupule B Generation and T	$1,682	Energy and mining

Figure 6 Parameterized Dashboard with Table Pop-up Sourced from Stored Process

STORED PROCESSES AS VISUALIZATION SOURCES

In addition to outputting data, stored processes can output visual content in the form of images in common formats such as png, jpg and gif. Via the stored process APIs these images can be addressed via URL just like any other images, as long as the user-agent has a current session with the SAS Stored Process Web Application. BI Dashboard has always been able to access any image via URL. For images outputted by stored processes, which require an authenticated session, BI Dashboard brokers a new session with the Stored Process Web Application via standard SAS Web application practices, so that the images can be viewed as part of a dashboard without requiring a second authentication within the dashboard.

When the image is brought into the dashboard, it is brought in as a flat image. Several of the types of interactivity that the Flash Dashboard Display Environment provides require that the indicators have data associated with relevant visualizations. Since only the image is provided and not the underlying data, indicators that display stored processes' visualizations can't participate in local filtering or data brushing.

The principle advantages of using stored processes as visualization sources include:

- reuse of existing stored processes
- availability of more types of visualizations

This approach depends on the stored process streaming back an image in response to the request. To perform this properly, the stored process should not have the stpbegin and stpend macros. The following code creates a chart and streams it back as a png.

```
goptions gsfname=_webout gsfmode=replace;
goptions device=png;
proc gchart data=sashelp.class; vbar age / discrete; run; quit;
```

The URL for such a stored process can be captured by going to the SAS Stored Process Web Application and executing the stored process. Then, right-click on the image and get the URL from the image properties. It should be a URL like this:

http://sasbi.demo.sas.com:8080/SASStoredProcess/do?_action=form,properties,execute,nobanner,newwindow&_program=%2FUsers%2Fsasdemo%2FMy+Folder%2Fstp_image_test&

If the URL of the image itself includes program=_replay, then it will not work with BI Dashboard.

With the URL in hand, simply paste it in to the URL field of either a graph indicator or an image decorator.

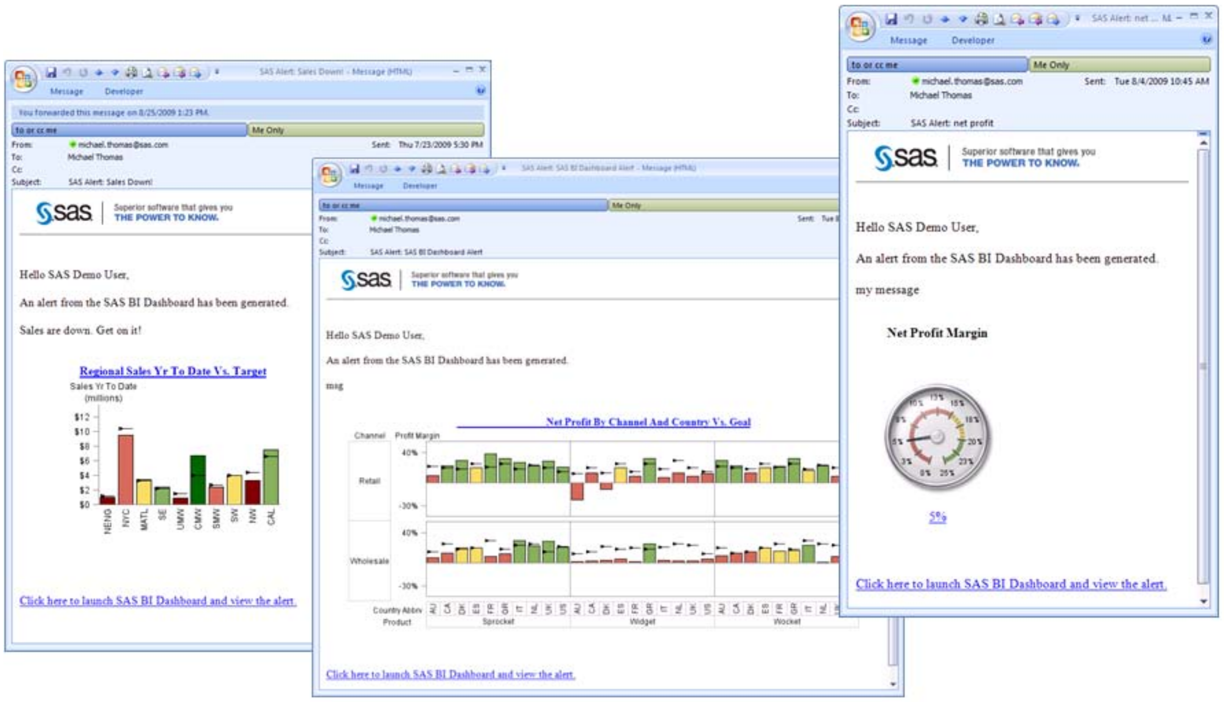
STORED PROCESSES AND ANALYTIC-BASED ALERTING

Stored processes and alerting are a powerful combination. While many BI systems can send alerts based on queries against relational or OLAP sources, the power of the SAS language can be used to generate alerts based on predictive analytics, optimization models and other types of analytics. While it is relatively easy to use alerts to convey bad news that has already occurred, SAS can produce alerts that, if acted upon, can keep bad news from happening. Instead of alerts that only tell of today’s problems, SAS can produce alerts that tell of tomorrow’s opportunities.

But when designing alerts that are based on analytics, signal-to-noise ratio is a key aspect of creating a system that the human decision-makers will heed. The signal-to-noise ratio problem is discussed after reviewing the basics of configuring alerts in BI Dashboard. Following those discussions, the role of stored processes in alerting is reviewed.

BI DASHBOARD ALERTING

Starting with BI Dashboard 4.2, BI Dashboard is able to send alert messages to either e-mail or to the Stored Process Alerts portlet. The Event Generator component of BI Dashboard uses a polling mechanism to invoke an indicator, and then generates an event when a change of interval is detected, where intervals are like below target, on target and above target. The best example of a change of state is when the single gauge of an indicator changes from on target to below target. As soon as Event Generator observes the change to below target, it generates an event that is passed to Alert Services. If an alert registration exists for below target for that particular indicator, then an alert is created, resulting in an e-mail or other communication. An example of such an e-mail alert is shown below in Figure 6, on the right.



SAS Alert: Sales Down! - Message (HTML)

You forwarded this message on 8/25/2009 1:23 PM.

From: michael.thomas@sas.com Sent: Thu 7/23/2009 5:30 PM
To: Michael Thomas
Cc:
Subject: SAS Alert: Sales Down!

Hello SAS Demo User,

An alert from the SAS BI Dashboard has been generated.

Sales are down. Get on it!

Regional Sales Yr To Date Vs. Target

Click here to launch SAS BI Dashboard and view the alert.

SAS Alert: SAS BI Dashboard Alert - Message (HTML)

From: michael.thomas@sas.com
To: Michael Thomas
Cc:
Subject: SAS Alert: SAS BI Dashboard Alert

Hello SAS Demo User,

An alert from the SAS BI Dashboard has been generated.

msg

Net Profit By Channel And Country Vs. Goal

Click here to launch SAS BI Dashboard and view the alert.

SAS Alert: net ...

From: michael.thomas@sas.com Sent: Tue 8/4/2009 10:45 AM
To: Michael Thomas
Cc:
Subject: SAS Alert: net profit

Hello SAS Demo User,

An alert from the SAS BI Dashboard has been generated.

my message

Net Profit Margin

5%

Click here to launch SAS BI Dashboard and view the alert.

Figure 7 Alert E-mail Examples

This simple case is also the most intuitive, because there is a single data value that triggers the alert. BI Dashboard also supports the concept of percentage of gauges in a particular interval. Those alerts are shown in the two other screenshots in Figure 6. If an indicator is a bar chart, where each bar in the bar chart is either below target, on target or above target, then an alert can be triggered if (for example) 30% or more of the bars are below target. Likewise, for a KPI indicator based on a result that has 10 rows of data and thus shows 10 gauges, an alert can be triggered if a given percentage of gauges is in a particular interval. A common request is that the alert should only trigger if one of the gauges of a given indicator is in a particular interval. For that case, a new indicator should be set up where the underlying data query only returns a row for the particular data value of interest.

The details of setting up alerts in BI Dashboard are left to the product documentation because the user interface was not finalized at the time of this writing.

ALERTING AND SIGNAL-TO-NOISE RATIO

After getting past the basics of raising an alert in response to an event, the real value in alerting is in having a good signal-to-noise ratio. With historical data, it is possible to get a perfect signal-to-noise ratio for some particular criteria. For example, the data says that sales were down at a store last week. It is easy to set up an alert that triggers because sales were down at that store last week. The signal-to-noise ratio is perfect. If sales were down 20 weeks in a year, then it is easy to achieve 0% noise and 100% signal — the alert just has to be triggered for each of those 20 weeks.

The more difficult problem is sending an alert because sales are likely to be down *next* week. One approach would be to always send an alert, since there is usually some probability that sales are going to be down. The e-mail alert could link to a detailed treatment of the probabilities for next week. But from the perspective of the recipient of the e-mail, who might not be inclined to absorb the statistical detail, the signal to noise ratio is 20/52 = .38. Another approach would be to never send an e-mail unless it is almost certain that sales will be down. This approach tends to eliminate noise, but also eliminates signal.

To illustrate further, consider a type of alerting with a bad signal-to-noise ratio: the car alarm. When a car alarm goes off, it's usually a false alarm, and most people are annoyed at the car owner. Few if any people actually react to a car alarm by assuming that a crime is being committed and calling the police. In this case, there is too much noise and not enough signal. The car alarm has no credibility.

On the other side, consider the “lap watch dog.” Lots of people joke about their supposed watch dogs that would more likely wag their tails and kiss a burglar than bark. Such a lap watch dog might be more tolerable than a dog that barks all night at the slightest disturbance, but the dog lacks not only credibility but purpose. At least, such a dog doesn’t serve the purpose of protecting the house from burglars.

Another very real world example is faced routinely by the National Weather Service and local authorities — to evacuate or not to evacuate, to cancel school to keep people off early-morning icy roads, or not. Regardless of the sophistication of the underlying statistical models, the decision to evacuate is largely binary. Either the alarm is sounded or it is not. Failure to sound the alarm leads to casualties very directly, but sounding the alarm too often leads to a loss of credibility that can also lead to casualties.

Like a lot of examples in alerting, these examples consider events of negative impact. But analytical models could also be brought to bear on divining opportunity as well. For example, alerts could be sent because it is likely that particular items should sell well in a particular geography next week. Such upside alerts face the same signal-to-noise ratio issues of credibility and purpose as downside alerts.

Most analytical models that can produce an alert of any value also have a lot of nuance and fuzziness. For example, a model that only says yes or no to whether sales will be down next week is inferior to a model that provides probabilities that can be traced through to root assumptions. But when it comes to alerting on such a model, the decision the system must make about whether to trigger a particular alert is always yes or no.

For a given organization, a given set of recipients and a given model, it might make the most sense to always send all of the detail. But such a detailed communication is more like a daily or weekly report on the model than really being an alert. The consumers farther away from both the domain and from analytics in general are less likely to engage such a report every time it comes across the in-box. While sending a detailed report might be safer in some ways, it might also limit the potential impact of the underlying analytical model on the organization.

Regardless of how much precision the underlying model is capable of in determining probabilities of events, the decision of sending out any particular alert is binary — the alert is either sent or not sent to a particular user or group of users. The underlying detail is more likely to be studied, and the overall model is more likely to be heeded, if credibility and purpose are enhanced by achieving an optimal signal-to-noise ratio over time.

STORED PROCESSES AND BI DASHBOARD ALERTING

As described above, BI Dashboard 4.3 can source data from a stored process. Also, BI Dashboard alerts are driven by the Event Generator component, which polls BI Dashboard repeatedly and fetches the indicator. An indicator used for an alert can be displayed as part of a regular dashboard, but it doesn't have to be. When it isn't, the indicator can differ from display indicators in both responsiveness and complexity.

A stored process used for an alert doesn't need to be particularly fast. When an indicator is based on a stored process, then either the stored process is executed or the result of a previous execution is fetched from a cache. Unlike a user's request for an indicator, the Event Generator's request isn't particularly time sensitive. The stored process could take a comparatively long time to execute, such as a minute, and no one would know because it occurs on a background thread. The recipient of the alert doesn't know how long it took to run, whereas the dashboard consumer knows very well how responsive the live dashboard is.

Also, an indicator used for alerting can benefit from simplicity. As discussed above, an alert must either be triggered or not. As such, a simple indicator that displays a stoplight functions well as an alerting indicator. While a red and green stoplight isn't particularly interesting visually, it is highly analogous to alerting itself. In addition to modeling the simple yes/no behavior that an alert must exhibit, a simple stoplight benefits from being purely qualitative and not quantitative. With a stoplight, the stored process code can distill the output down to 0 and 1 — red or green, on or off, trigger or don't trigger. Freed from the burden of communicating any numeric information, such as a quantification of risk, the underlying stored process can combine different inputs synthetically to the single quantity of being either interesting or not. On the other hand, a gauge like a slider, dial or speedometer can communicate more numeric information along with the quality of relevance. If the underlying model can be boiled down to a single dimension, then this might be useful.

In any of these cases, the gauge itself is just the lead-in to the main story. The gauge arrives in the e-mail and interests the user, who then clicks on it. Upon clicking, the user can be led to detail of any level of richness. One option is to open a Flash dashboard that provides both advanced visualization and interactivity. But the alert could also go to a simple departmentally hosted Web page that explains why the alert was sent and what to do about it.

CONCLUSION

Stored processes can provide great value, but the output needs to be presented in a compelling way to the right audience to realize that value. Interactive Flash dashboards can be very enticing, but without solid insight from the data the dashboards are just pretty eye candy. When BI Dashboard 4.3 and stored processes are used with the techniques described in this paper, the power of analytics can reach deeper in to the enterprise with exciting, interactive Flash interfaces.

ACKNOWLEDGEMENTS

Thanks to Vincent DelGobbo for help with the technical content of this paper.

CONTACT INFORMATION

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

Name: Michael Thomas
Enterprise: SAS
E-mail: michael.thomas@sas.com
Web: http://www.sas.com