Adventures in Building Web Applications: A Tutorial on Techniques for Real-World Applications

Jason Gordon, GE Capital Card Services, Stamford Connecticut
Michael Davis, Bassett Consulting Services, North Haven, Connecticut
George Sharrard, GPS Corporation, Rowayton, Connecticut

ABSTRACT

The authors recently completed their first web-based application, a Dashboard reporting system for the Card Services a GE Capital Company. Along the way, they discovered how SAS/IntrNet® software operates differently than they originally thought. They also developed some tools and not so obvious strategies to build and maintain a web site with well over a hundred pages. This presentation is an attempt to share those experiences. Among the topics covered:

- maintaining a test site using automated string replacement
- automating web site updates through block replacement
- validating user input without Javascript or Java
- logging what users do without hit counters or server logs
- using PUT statements to _webout -- not what you thought
- returning SAS/GRAPH® output to a web browser

The presentation will feature some on-line demonstrations to illustrate how these techniques are used in real-world applications.

INTRODUCTION

GE Capital, with assets of more than US$345 billion, is a global, diversified financial services company with 28 specialized businesses. GE Capital, based in Stamford, Connecticut, provides equipment management, mid-market and specialized financing, specialty insurance and a variety of consumer services, such as car leasing, home mortgages and credit cards, to businesses and individuals around the world.

Not long ago, the Risk Management Group of GE Card Services identified the following unmet needs:

- Standardized set of portfolio reporting and measures on account management (authorization and credit line) strategies
- Friendly way for users with little programming knowledge to obtain data and information on portfolio diagnostics
- Easy mechanism for sharing data and reporting across a multi-location business
- Fast way to get custom reports and graphics for analysis and presentations

BUSINESS OBJECTIVES

The Process Control Dashboard was created to facilitate the following business objectives:

- Simplify and distribute account management reporting and measures to increase profitable sales and reduce loss on high risk portfolio segments
- To reduce creation time for portfolio review and analysis
- To provide flexible reporting and data on a variety of account management metrics
- Standardize definitions and reporting across business

MAINTAINING A TEST SITE

The Dashboard web site was prototyped on a laptop computer running SAS software® and Microsoft Personal Web Manager under Microsoft Windows NT 4.0 Workstation. When the prototype was ready to be promoted to the Sun web server, the HTML pages, graphics, and other files were transferred by FTP and SAS/CONNECT® to a test directory on the Sun web server. The Sun web server also serves as the SAS application server for the Dashboard application.

After testing on the Sun web server, the HTML pages and graphics would be promoted to the directories on the web server to which the user hyperlinks were pointed. The strength of this approach is that changes can be introduced to the web site for testing while access to the production areas is maintained.

However, since both the test and production areas of the web site use the same directory of SAS programs and the same SAS data libraries, some care has to be exercised so that users of the production portion of the web site do not access features that could give erroneous results or worse, crash the socket service.

AUTOMATED STRING REPLACEMENT

To understand the need for automating string replacement, consider the following example portion of an HTML form, drawn from one of the application web pages:

```
<form action=
"http://hounddog/scripts/brokerv6.exe"
method="get" target="_blank">

<input type=hidden name="_service"
value="newsas_v6">
```

For those not familiar with HTML forms, the first statement is an HTML Form tag that directs the web browser to call the Version 6 broker and to pass the parameters entered via the HTML form to the SAS broker, which observes the CGI (Common Gateway Interface) standard.

In this example, we have renamed the Windows version of the SAS/IntrNet Version 6 Application Broker, brokerv6.exe, to prevent confusion with the Version 8 Application Broker, which resides in the same virtual directory (/houndog/scripts). The broker call also specifies that the parameters are to be passed by the "GET" method and that the results of the call are to be returned in a new browser window.

The second statement passes along a hidden parameter required by SAS/IntrNet, _service, which directs the Application Broker to launch a new SAS session to handle the broker request instead of sending it to already running default (socket) service. This launch service is useful during testing in that if something goes wrong, only the launched SAS session will fail, leaving the socket service intact.

By contrast, when an HTML page is promoted to the production area of the Sun web server, those two statements might be replaced by examples that look similar to the following statements:

```
<form action=
"http://Sunweb01/cgi-bin/brokerv6"
method="get" target="_blank">

<input type=hidden name="_service"
value="default">
```

If only one or two pages needed to be changed, it would be a trivial task, one that would probably be performed by hand. However, each Dashboard measure is launched by its own HTML page and the Dashboard site launches over a hundred measures.

Since this is a SAS application, it begat a SAS solution. An SCL (Screen Control Language or SAS Component Language) program was written to scan each HTML page in the Dashboard directory, swap the Microsoft Windows broker calls for the Sun Solaris versions, and to write the updated HTML files to a second output directory. The working portion of the global replacement program appears below:

```
/* global replacement program */
/* open directory of input files */
rc=filename('tempdir', <source directory>;

/* open directory to make list of files */
dir_id= dopen('tempdir') ;

/* make a list of all HTML files */
dir_lst = makelist() ;
file_cnt = dnum(dir_id) ;

do i= 1 to file_cnt ;
   filenm = dread(dir_id, i) ;
   if upcase(scan(filenm, 2, '.')) eq
   'HTML' then
   rc = insertc(dir_lst, filenm, -1) ;

end ; /* do 1 to file_cnt */

/* close temporary filerefs */
rc= filename('tempdir', '') ;

/* create the blank Find, Replace lists */
find_lst= makelist() ;
replace_lst= makelist() ;

/* open the G_REPLACE data set */
dsid_gr=
 open(<global replacement data set>,'i') ;
call set(dsid_gr) ;
nobs= attrn(dsid_gr, 'nobs') ;

/* loop through this section for all
   G_REPLACE observations */
do i = 1 to nobs ;
   rc= fetch(dsid_gr) ;

   /* insert the Find and Replace strings
   into the lists */
   rc = insertc(find_lst, find_str, -1) ;
   rc = insertc(replace_lst, replace,-1) ;

end ; /* loop through all G_REPLACE obs */
```

```
/* close the G_REPLACE data set */
dsid_gr = close(dsid_gr) ;

/* loop through each HTML file */
html_lst_len = listlen(dir_lst) ;
do i = 1 to html_lst_len ;

   /* open the next HTML file for input */
   in_f_nm = getitemc(dir_lst, i) ;
   in_f_nm = <source directory> ||
    "\" || in_f_nm ;
   rc = filename('in_html', in_f_nm) ;
   fid_in = fopen('in_html', 'i') ;

   /* open the output HTML file */
   ot_f_nm = getitemc(dir_lst, i) ;
   ot_f_nm = <output directory> || "\" ||
    ot_f_nm ;
   rc = filename('out_html', ot_f_nm) ;
   fid_out= fopen('out_html', 'o') ;

   /* loop through input HTML file */
   if (fid_in gt 0) then
   do while(fread(fid_in) = 0);
      rc = fget(fid_in, buffer, 200);

      /* if statement is to be replaced */
      replace_lst_len = listlen(find_lst);
      do j = 1 to replace_lst_len ;
         find_str= getitemc(find_lst, j) ;
         if upcase(left(buffer)) eq
          upcase(left(find_str)) then do ;
            buffer=getitemc(replace_lst,j)
            ;
            j = replace_lst_len ;
         end ;
      end ;

      /* write buffer to output file */
      rc = fput(fid_out, buffer) ;
      rc = fwrite(fid_out) ;

   end; /* loop through input HTML file */

   /* close output HTML file */
   fid_out = fclose(fid_out) ;
   rc = filename('out_html', '') ;

   /* close input HTML file */
   fid_in = fclose(fid_in) ;
   rc = filename('in_html', '') ;

end ; /* loop through each HTML file */
```

The global replacement data is not reproduced here. It consists of two character variables, *find_str* and *replace*. A few additional points regarding this routine should be noted:

- statements to be replaced must appear by themselves on their own line to be replaced
- to eliminate problems when the Dashboard pages are ported to the Sun web server, the HTML file extension rather than the HTM extension is always used
- non-HTML files are ported separately although GIF and JPEG files could be copied via SCL using the SASHELP.FSP.IMGDAT Class

BLOCK REPLACEMENT

It is necessary to periodically update the selections that can be made from Dashboard website pages. Features requiring periodic updates include:

- adding to clients from the pull-down lists
- changing the years shown as radio stations
- showing months for which reports are available

In addition, when items on the Dashboard website are changed, the change is noted in the “News” link on the main Help page.

In many situations, such updating is accomplished by using htmSQL, a CGI application also distributed with SAS/IntrNet. htmSQL uses a variation of Structured Query Language (SQL) to replace database queries embedded in an HTML page with the results of the query. It is a good technique to use when selections of pull-down lists and radio boxes change daily or even more frequently

However, it seemed wasteful to make a database call when the selections were update monthly or less frequently. So the decision was made to update these features with a SCL program, launched from a web page interface.

How does it work? The portion of each HTML page that needs to be updated is bracketed by a part of tags. The tags appear as comments to the browser and are ignored when displaying the page. However, when a program similar to the global replacement program is run, the block of HTML code bracketed by the tags is replaced.

As an example consider the following HTML code, an example of a client selection pull-down list block:

```
<!-- client select start -->
   <p><select name="client" size="5">
   <option value='00111'>Client 111
   <option value='00222'>Client 222
   <option value='00333'>Client 333
   </select></p>
<!-- client select end -->
```

When the SCL program parses the page and encounters the Client Select Start tag, it ignores subsequent lines until it encounters the Client Select End tag. Then the SCL program inserts the

replacement block where the two tags were encountered.

Before the block replacement program is run, other programs are launched that read tables that contain the current client list or build new month or year lists based on the current date. The tags and other HTML statements are written with SCL FPUT and FWRITE statements.

The glue that pulls the block replacement feature together is the *b_rplace* control table that links the start and end tags with the filename of the replacement block. In the case of the Dashboard application, the *b_rplace* table contains some additional columns (variables) that can be used to further restrict which HTML pages are updated.

As an example of how that feature works, each measure falls into a category. One of those categories relate to portfolio performance measures. The name of those pages are prefixed with "pp_". If the row in the *b_rplace* control table has a "Y" in the *Perf* column, the block replacement program parses it for the tags associated with that particular block.

For better performance, the block replacement program reads the *b_rplace* into SCL lists. As it checks each line in the HTML page, it cycles through all items in the SCL lists for a match before moving on to the next line in the HTML page.

In addition to speeding up replacements, the ability to apply block changes by category increases the ease of selectively updating the web site. For those who wish to follow the approach used to maintain the Dashboard, the authors recommend setting up an naming convention and keeping the web site with a single directory over the commonly used approach of setting up several levels of directories.

VALIDATING USER INPUT

Many, if not most web application designers validate user input using JavaScript. There is much to recommend this approach. However, the developers of the Dashboard decided to use a SAS-based approach.

The decisive reason was the ability to alter the validation parameters programmatically through changing the controlling SAS data set. As part of a SAS application, it is easier to change a observation in a SAS data set than to search and alter a script embedded in a web page.

The SCL program module used to validate user input is known as the Validator. The Validator is executed via a PROC DISPLAY and returns two macro variables to the SAS program that calls it. The macro variable *err_flag* contains "0" if all validation checks pass. If one of the checks fail, the macro variable *err_msg* contains a short text description of the failed check.

If the check fails, the program that executed Validator writes the contents of *err_msg* as part of an HTML page sent to the web browser. The rest of the SAS program is aborted.

The Validator evaluates macro variables that are created by SAS/IntrNet Application Broker from an HTML form. The Validator checks are driven by a control data set with the following structure:

Variable	***Description***
chktype	check type
chkvar	check variable
compvar	comparison variable
comp_op	comparison operator
constant	comparison constant
fail_msg	failure or warning message
num_char	numeric or character
severity	severity of comparison
valpage	HTML page(s) to validate
vformat	variable format

The *chktype* variable tells the Validator whether check is against another macro variable passed from the HTML form or with the constant from the control data set. The name of macro variable to be checked is comes from *chkvar*. If the *chktype* variable contains "constant", the comparision is made against the contents of the *constant* variable in the control data set. Otherwise, the comparison is made against the contents of the data set variable named in *compvar*.

The variable *num_char* specifies whether the variable to be check is numeric or character. The *vformat* variable defines the SAS format used to transform the value of macro variable variable constant before the comparison is made.

The *valpage* variable specifies the name of the HTML pages for which the check applies. To make the Validator easier to program and maintain, *valpage* can specify a page prefix instead of the complete page name. As an example, "pp_" would apply the specified check to any of the performance portfolio measure pages.

The *severity* variable allows a failed check to be specified as a warning instead of an error that would cause the SAS program to be aborted. The text contained in the *fail_msg* variable is return to the SAS program in the *err_msg* macro variable.

LOGGING WITHOUT HIT COUNTERS

Many websites measure activity using hit counters or web server logs. These are good and useful techniques but suffer from the following limitations:

Hit counters measure where visitors go on a web site but not what they do. The visitors may have browsed a page in error or as a curiosity

Web logs show more information about visitors but they may not show what the visitors selected on the page. The downside is that server logs can be intimidating to analyze. Additional software is sometimes required to perform this analysis.

As part of the Dashboard application, the SAS program launched from each measure page logs all of the parameters generated by the HTML form. These parameters include:

- measures
- clients
- time period
- graph types

Each log entry is written out as an observation containing the date, time, and information from the HTML form. Then the log entry is appended to the log file using the APPEND procedure as follows:

```
proc append
 base=logger.logger data=logger force ;
run ;
```

From time to time, the Dashboard administrator downloads the logs for analysis. The focus of the analysis is to determine which are the most popular measures, clients and graph types. Another focus of the analysis is to determine which features of the Dashboard are not used. Those features may be obsolete or the users may require additional training to use those features.

LET'S MAKE SOME OUTPUT

The reserved filename _webout has got to be one of the coolest features of the SAS/IntrNet software. While it's not ODS, it certainly becomes the Output Delivery System for a web enabled SAS application. If you're familiar with using PUT's to write reports to external files in a DATA _NULL_ step, then you have the idea of how one uses _webout. But, as with all things new, sending output to _webout isn't exactly like using PUT's to write to an external file. In fact the first time you try, other than the syntax, nothing seems to be the same!

SAS/IntrNet and SAS/GRAPH

Using SAS/GRAPH through a Web interface requires you learn a few new tricks. Everything works the same, all your knowledge about the graph procedures is directly transferable. There are a couple of different device drivers to know about and some Version 6 limitations that have to be worked around. Most appear to be resolved in Version 8.

The following discussion is based on our experience using a Unix web server.

DRIVERS

SAS/GRAPH generic drivers allow you to route graphical output to any device defined to your operating system. In Windows you can specify:

```
goptions device=winprtm ;
goptions device=winprtg ;
goptions device=winprtc ;
goptions device=winplot ;
```

allowing the Windows print drivers installed on your system to control the bit-stream sent to the available output devices. SAS does not need to know what make or model laser printer you have, so long as Windows knows. When it comes time to print a graph just make sure you have selected the device you want from the FILE - PRINT SETUP pull down box. (The same applies to the OS2PRT*x* drivers.)

This same idea applies to graphs displayed via a web browser interface. All that gets displayed in a web browser is a GIF or JPG image. What it will be printed on is not your concern. The web browser is in-charge of printing and having the proper drivers available. The GOPTIONS statement for the "*generic*" print driver for web output is:

```
goptions device=gif ;
```

However, unlike the metafile SAS sends to the generic Windows print file, GIF is a file format not under SAS software control. GIF is in fact, a very low resolution format best suited to displaying little pictures on a web page. You may be very

disappointed when you see your first graph displayed using the GIF or JPG drivers.

WRITING LINES TO AN EXTERNAL FILE

In a data step you can write to an external file using a FILE statement and PUT statement(s).

```
/* example 1 */
data _null_ ;
   set old ;
file '<filename>'
 noprint notitles ;
put
 @10 dob ddmmyy10.
 age 2.
 grade 2.
 ;
run ;
```

For each observation in "OLD" one line will be written to the file.

Sending html encapsulated text or graphics to a browser is pretty much the same thing. A special fileref called *_webout* is provided by SAS/IntrNet for this purpose.

```
/* example 2 */
data _null_;
  file _webout ;
  put 'Content-type: text/html';
  put;
........

run ;
```

Whatever you are going to send to _webout, you must use the first two PUT statements in the example above. BUT, unlike the PUT statements shown in Example 1, there are a few restrictions.

The most important restriction is that you get to write to _webout only once! Everything you need to send back to the requesting browser has to be ready to go as soon as your program addresses _webout.

Normally, with PUT statements, you can write to the same external file several times in the same data step or over several data steps. PUTs can modify an existing external file (append records to the end) or re-write the entire file (after, say, the original file has been sent to a printer).

NOT SO WITH _WEBOUT !!

Our original attempts at using _webout mimicked what we knew about using PUTs to write to external files. We used GREPLAY procedure to load multiple graphs onto one page, having three composite pages that we wanted returned to the requesting browser.

```
/* example 3 – THIS DOES NOT WORK */
data _null_;
  file _webout ;
  put 'Content-type: image/gif';
  put;
run;

goptions device=gif gsfname=_webout
gsfmode=append  rotate=landscape display ;

%include "&path2./replay1.sas" ;

goptions device=gif gsfname=_webout
gsfmode=append  rotate=landscape display ;

%include "&path2./replay2.sas" ;

goptions device=gif gsfname=_webout
gsfmode=append  rotate=landscape display ;

%include "&path2./replay3.sas" ;
```

Best we could get was the first composite page. No errors or warnings for pages two and three. What you need to do is to write the graphical images to disk (all 3 in this case) as GIF or JPG files. Then you write html to _webout linking to the disk files.

Note that in the following example, Example 4, the reference to the macro variable *fiz* is a unique randomly generated SAS name used to identify the gif images associated with a particular Dashboard request.

```
/* example 4 */
filename graf01ga "<path>/&fiz.a.gif" ;
goptions device=gif gsfname=graf01ga
gsfmode=replace display
         rotate=landscape ;
%include "<path>/replay_03a.sas" ;

filename graf01gb "<path>/&fiz.b.gif" ;
goptions device=gif gsfname=graf01gb
gsfmode=replace display
         rotate=landscape ;
%include "<path>/replay_03b.sas" ;


filename graf01gc "<path>/&fiz.c.gif" ;
goptions device=gif gsfname=graf01gc
gsfmode=replace display
         rotate=landscape ;
%include "<path>/replay_03c.sas" ;

data _null_;
  file _webout ;
  put 'Content-type: text/html';
  put;

  put '<html>' /
      '<body>' /
```

```
      '<p>'    /
      "<img src=
http://<URL to graph directory>
      /&fiz.a.gif"
      " width='&width' height='&height'>"
/
      '</p>'   /
      '<p>'    /
      "<img src=
http://<URL to graph directory>
      /&fiz.b.gif"
      " width='&width' height='&height'>"
/
      '</p>'   /
      '<p>'    /
      "<img src=
http://<URL to graph directory>
      /&fiz.c.gif"
      " width='&width' height='&height'>"
/
      '</p>'   /
      '</body>' /
      '</html>' ;
run ;
```

FONTS

Another great disappointment concerns the text associated with your SAS/GRAPH image. A GIF or JPG graphic created on a UNIX box does not access to True Type fonts. Unlike HTML, the GIF image is a finished product when it is returned to your browser. What this means for graphic output is that there is a LOT of pixelation on the screen image. Small text (in legends, axis labels and tick marks) is unreadable - even if you zoom in . Using GREPLAY to load several graphs on one page only makes things worse.

The work around for this is to always generate multiple versions of the same graph using different device drivers. Generate one graph (to disk) as a GIF or JPG file. This is what gets displayed in your browser. A second disk file is created as a PS (PostScript®) file. The "phasr540" driver works well for this. This image cannot be displayed as a graphic element in a composite HTML page.

However, if the if a PostScript plug-in such as Ghostview has been installed into the browser being used, the PostScript image can be display using the following technique. Set-up a link to the PostScript version of the graphic at the bottom of your composite HTML graphic page. After reviewing the GIF image, users can "Click here for a high resolution PostScript file". A download dialog box opens up and the PostScript file is moved from the web server to the browsing computer.

The code used to generate the PostScript graphic and to insert the link on the composite HTML graphic page follows:

```
filename graf01pa "<path>/&fiz.a.ps" ;

goptions device=phasr540 cback=white
gsfname=graf01pa display gsfmode=replace
rotate=landscape ;

%inc "<path2>/replay_03a.sas" ;

filename graf01ga "<path>/&fiz.a.gif" ;

goptions device=gif gsfname=graf01ga
gsfmode=replace display rotate=landscape ;

%inc "<path2>/replay_03a.sas" ;

data _null_;
  file _webout ;
  put 'Content-type: text/html';
  put;
put '<html>' /
      '<body>' /
      '<p>'    /
      "<img src=
http://<URL to graph directory>
      /&fiz.a.gif"
" width='&width' height='&height'>" /
      '</p>'   /
      "<a href=http://
<URL to graph directory>/&fi.a.ps>"
      'Click Here for High Resolution
PostScript Graph </a>' /
'<p>' /
;
run ;
```

VERSION 8

The Dashboard was constructed using Version 6 because the web server used to host the application was running a version of Sun Solaris not capable of supporting Version 8. Recently, the web server was upgraded and plans are being made to upgrade the Dashboard application to run under Version 8.

Part of the plan are to change the high resolution graphics from PostScript to Adobe Acrobat® (PDF) files because Acrobat files are more universally supported as a plug-in on the client browsers. Other plans include taking advantage of hardware fonts to avoid the “pixelation” problem.

The biggest advantage of web-enabling applications with Version 8 is probably the ability to generate the HTML format tags automatically using the Output Delivery System (ODS). Developers using Version 8 should also look into using the Load Manager and a Pool Service to better handle the

issue of scaling up web-enabled applications for high-demand levels.

CONCLUSION

The Dashboard application illustrates how web-enabling a SAS application can make user-selected (dynamic) graphical information and tables available. The techniques used build upon one's existing SAS skills. This paper attempts to provide some of the supplemental information required to put SAS applications on the web.

Please note that the latest documentation for SAS/IntrNet software is available from the SAS Institute web site, http://www.sas.com. The following bibliography may be helpful to persons who are not familiar with SCL and SCL lists.

BIBLIOGRAPHY

Davis, Michael (1998), "SCL for the Rest of Us: Nonvisual Uses of Screen Control Language," *Proceedings of the Twenty-Third Annual SAS Users Group International Conference,* 23, 193-202.

Horwitz, Lisa Ann (1998), "Harnessing the Power of SCL Lists," *Proceedings of the Twenty-Third Annual SAS Users Group International Conference,* 23, 48-56.

ACKNOWLEDGEMENTS

SAS, SAS/CONNECT, SAS/GRAPH, SAS/IntrNet, and SAS/QC are registered trademarks of SAS Institute Inc. Acrobat and PostScript are registered trademarks of Adobe Systems Inc.

CONTACT INFORMATION

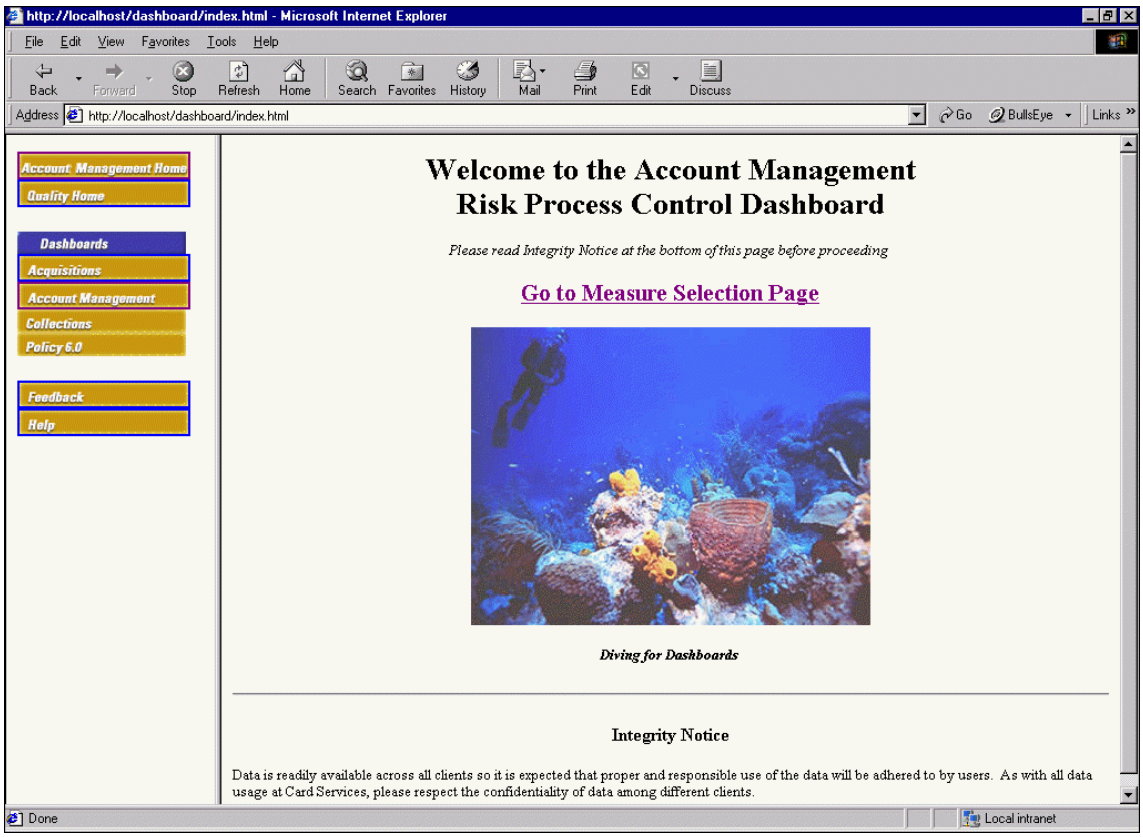
The authors may be contacted as follows:

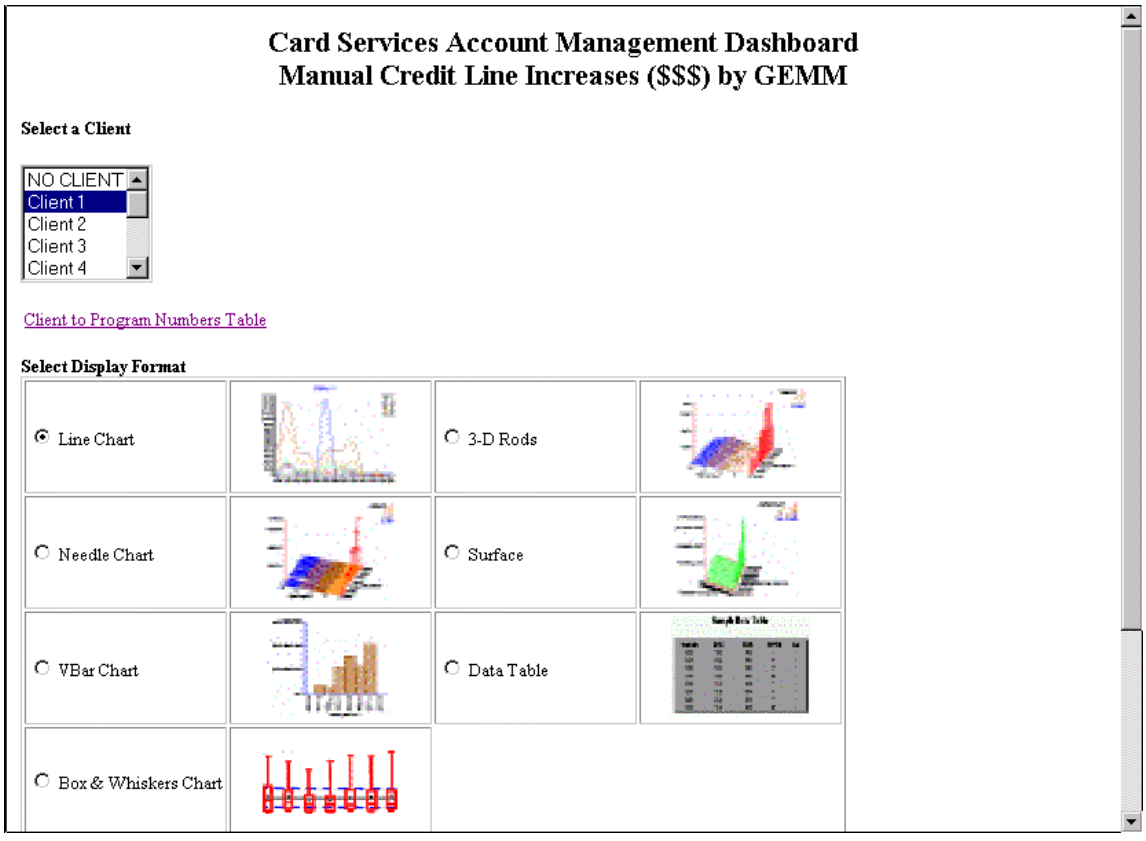
Jason T. Gordon
GE Capital Card Services
1600 Summer Street
Stamford, CT 06927
E-Mail: Jason.Gordon@gecapital.com
Telephone: (203) 357-3350
Facsimile: (203) 357-4281

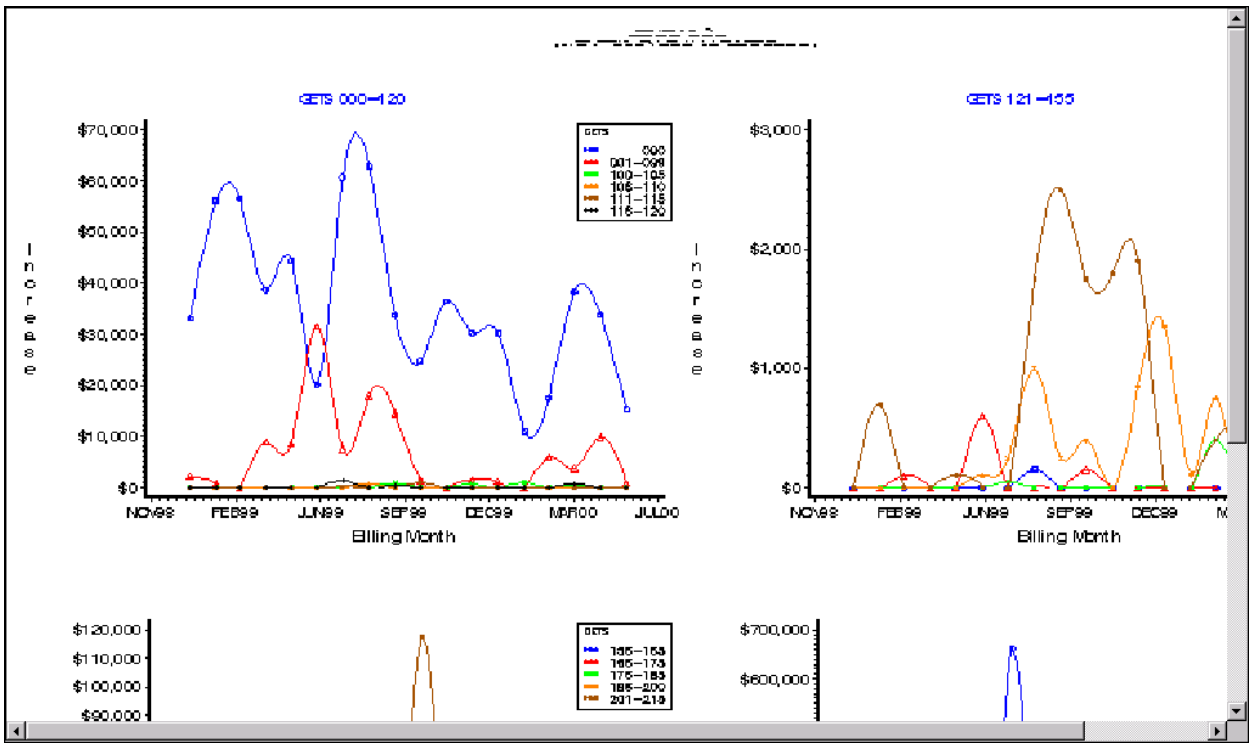
Michael L. Davis
Bassett Consulting Services, Inc.
10 Pleasant Drive
North Haven CT 06473-3712
E-Mail: michael@bassettconsulting.com
Web: http://www.bassettconsulting.com
Telephone: (203) 562-0640
Facsimile: (203) 498-1414

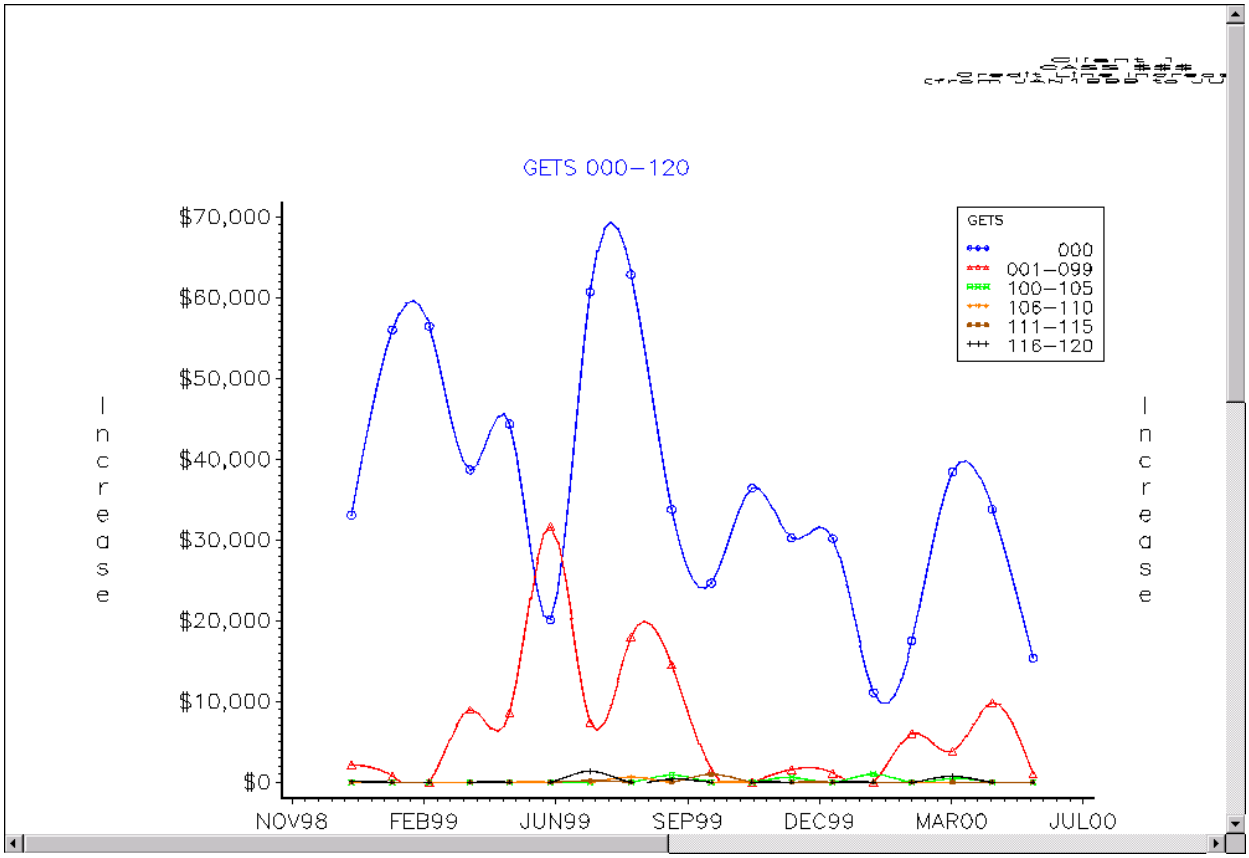
George P. Sharrard, Ph.D.
14 Sunwich Road
Rowayton CT 06853
E-Mail: georges00z@aol.com
Telephone: 203-838-7248
Facsimile: 203-838-7216

Please note that the code supplied in this paper is designed only to illustrate the concepts being discussed and will need to be modified to work in other applications. Modified code is not supported by the authors.


http://localhost/dashboard/index.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address http://localhost/dashboard/index.html
Go BullsEye Links
Account Management Home
Quality Home
Dashboards
Acquisitions
Account Management
Collections
Policy 6.0
Feedback
Help
Welcome to the Account Management
Risk Process Control Dashboard
Please read Integrity Notice at the bottom of this page before proceeding
Go to Measure Selection Page
Diving for Dashboards
Integrity Notice
Data is readily available across all clients so it is expected that proper and responsible use of the data will be adhered to by users. As with all data usage at Card Services, please respect the confidentiality of data among different clients.
Done
Local intranet


Card Services Account Management Dashboard
Manual Credit Line Increases ($$$) by GEMM
Select a Client
NO CLIENT
Client 1
Client 2
Client 3
Client 4
Client to Program Numbers Table
Select Display Format
Line Chart
3-D Rods
Needle Chart
Surface
VBar Chart
Data Table
Box & Whiskers Chart


GETS 000-120
GETS 121-135
Increase
Billing Month
NOV98
FEB99
JUN99
SEP99
DEC99
MAR00
JUL00
GETS
000
001-099
100-105
106-110
111-115
116-120
121-130
131-135
136-145
146-155
156-165
166-175
176-185
186-200
201-215


GETS 000-120
Increase
$70,000
$60,000
$50,000
$40,000
$30,000
$20,000
$10,000
$0
NOV98
FEB99
JUN99
SEP99
DEC99
MAR00
JUL00
GETS
000
001-099
100-105
106-110
111-115
116-120