Paper 182-27

Connecting the SAS® System to the Web: A Hands-on Introduction to SAS/IntrNet® Application Dispatcher

Vincent Timbers, Penn State, University Park, PA

ABSTRACT

There are several methods for accessing the SAS System through the Web to provide Web-enabled information-delivery solutions. The Application Dispatcher is a SAS System server that executes SAS programs from a Web browser, which is easily implemented by those with knowledge of the SAS DATA Step and Procedures. This makes it a good method for moving existing SAS programs to the Web.

This paper will provide a hands-on introduction, including setup of the Application Dispatcher and the development of Web pages and SAS programs that connect the SAS System to the Web.[1]

INTRODUCTION

SAS/IntrNet provides several methods of accessing the SAS System from the Web. These methods can be broken into two main categories 1) data services, which allow users to update and query data from a Web browser with SQL-type queries, and 2) compute services, which allow users to execute SAS programs from a Web browser.

SAS/IntrNet Application Dispatcher is a compute service that allows users to pass parameter selections from a Web page to the appropriated SAS program that executes on a server, sending the results back to the user's Web browser. Because the program is executed on the server, end users only need a Web browser. There is no need for SAS on their workstations. Application Dispatcher can execute four types of programs that take advantage of all data access, manipulation, reporting and analysis capabilities of the SAS System: SAS programs stored in external files, source entries, SCL entries, and macro entries.

Application Dispatcher is an excellent choice for those who have SAS programming experience and little or no experience with CGI programs or other methods of Web application development. Many SAS programs can be adapted to work with Application Dispatcher with few coding changes.

APPLICATION DISPATCHER OVERVIEW

Application Dispatcher consists of two components. A Common Gateway Interface (CGI) program called the Application Broker resides on a Web server. The second component is the Application Server that is simply a SAS session running on a server that executes the SAS programs.

---

[1] The information presented in this paper is specific to SAS Release 8.2 for the Windows operating system.

An Application Dispatcher program usually starts with a Web page where users make selections or enter data on a form. When the user submits the form the items are passed to the broker. The name of the Application Server that is to process the request and the name of the program to run is also sent from the Web form to the broker. These two items are usually hidden fields on the form that the user is unaware of. Using a configuration file that contains the locations of Application Servers, the broker directs the user selections along with the program name to the appropriate Application Server. The SAS program receives the parameters selected or entered on the Web form as SAS macro variables that are resolved in the SAS program. The program is executed and the results are sent back to the user's browser. It is that simple!

The sample application below is a sales report that can be produced for different regions. The user can select the region from a list on the Web form. When the form is submitted the broker will send the selection to the Application Server along with the program name where the program will be executed. The program will receive the region selected as a macro that will be resolved in a WHERE statement to restrict the report to the region selected. ODS will create HTML output from a PROC REPORT procedure and direct it back to the browser.

APPLICATION DISPATCHER INSTALLATION

Installation of the Application Dispatcher includes installing the Application Broker on a Web server and installing the Application Server on a SAS server.

APPLICATION SERVER INSTALLATION / SETUP

The installation of the Application Server begins by including SAS/IntrNet in the installation of the SAS System on the server. This will install the necessary components on the SAS Server. However, at this point the Application Server has not been created or configured.

"Application Server" is a general term referring to the SAS session that processes the request. A specific instance of an Application Service like the one created in the next section is called a "service." In the remainder of this paper the term "*service*" refers to the Application Server.

An Application Server is created by selecting **Programs**, **The SAS System**, **IntrNet** and **Create a New IntrNet Service** from the **Start** button. From the Welcome window that appears (not shown here), click the **Next** button and the Select Configuration Task window will appear as shown in Figure 1.

Figure 1: Select Configuration Task window

There are several types of services (Application Servers). A socket service is an Application Server that remains running at all times waiting for requests. A launch service is an Application Server that starts when a request is received from the Application Broker and stops when the request is completed. Poll services are started when a request is received and remains running until a specified time of inactivity has passed before stopping. Pool services can also be defined to run multiple Application Servers. If one server is busy when a new request is received, the pool service can start a second Application Server. The Load Manager is a program that manages requests and servers by sending requests to servers that are idle and starting servers for pool services as needed. For this paper, a socket service will be used. See the SAS/IntrNet documentation for more detail on all types of services and the load manager.

With socket service selected, click **Next** and the Enter Service Name window will appear as shown in Figure 2.
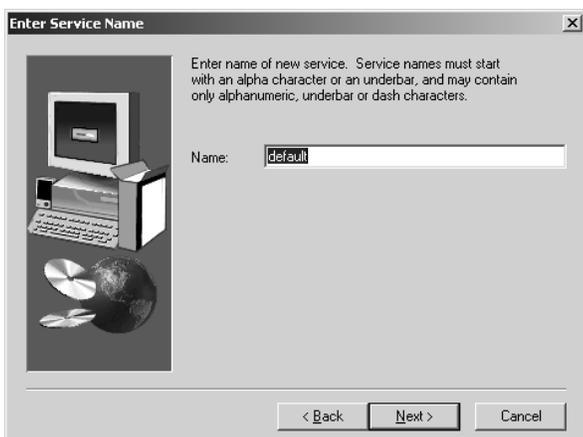

Figure 2 – Enter Service Name window

Any name can be used for the service name. However, "default" will be used in this paper. It is a good idea to use "default" as the service name when getting started with Application Dispatcher because the sample applications provided with the software are written to use the "default" service.

Click **Next** to record the service name and the Choose Service Directory window will open as shown in Figure 3.

Figure 3: Choose Service Directory window

The service directory is the subdirectory where files for the service are located. These files include a SAS program file containing the commands for starting the service and log files. To change the default location of the service directory, click on the **Browse** button to open the Choose Folder window (not shown). Click **Next** to record the service directory and open the Enter Service Ports window shown in Figure 4.
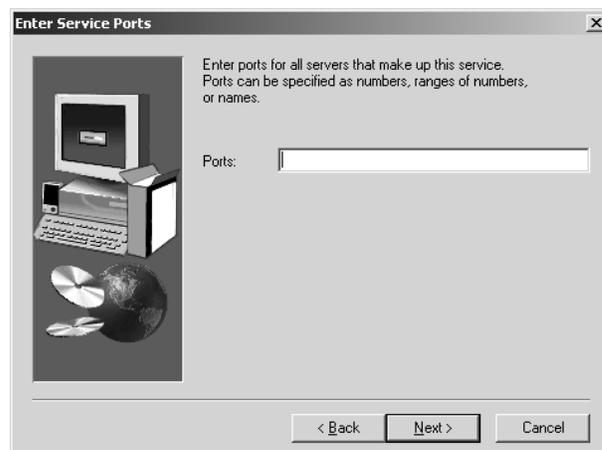

Figure 4: Enter Service Ports window

The port number entered here is the TCP-IP port the broker and the service use to communicate. A port number not in use by another application should be used. For this example enter 5001. After clicking the **Next** button the Enter Admin Password window will open (not shown). The password entered will be required when performing administrative tasks from a Web browser such as stopping the service. If a password is not entered, administrative tasks may be performed without a password. Click **Next** to save the administrative password and the Create Service window will open with a summary of all selections for review, as shown in Figure 5.
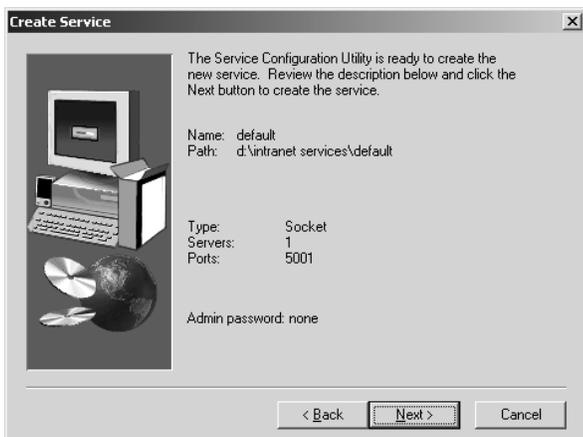
Figure 5: Create Service window

After reviewing the information, click the **Next** button in this window and the next three windows to complete the creation of the socket service.

BROKER INSTALLATION AND CONFIGURATION
The Application Broker is installed on a Web server from the Client Side Components CD. The broker is typically installed in the cgi-bin directory that is in the Web server root directory. The sasweb directory containing SAS/IntrNet samples will also be created in the Web server root directory.

After the broker is installed, it will need to be configured by editing the broker.cfg file found in the cgi-bin directory. This file contains several configuration options with documentation including sections for all three types of services: socket, launch and pool.

The section of the broker.cfg file below is the configuration for the socket service named "default" located on the SAS server at the address appsrv.yourcomp.com and communicating on port 5001.

```
SocketService default "Reuse existing session"
ServiceDescription "Pages reference this
generic server when they don't care which
service is used."
ServiceAdmin "Your Name"
ServiceAdminMail "yourname@yoursite"
Server appsrv.yourcomp.com
Port 5001
#  Remove the following line for any servers
before V8.1
FullDuplex  True
```

In the configuration of the "default" service, enter the information specific to your site for each item. For "Server" replace appsrv.yourcomp.com with the DSN or IP address for the "default" Application Server service previously created.

It should be noted that a broker.cfg file could contain configurations for multiple services. This is done by copying the configuration for a service and making the required changes.

After the Application Server and Application Broker have been installed and the broker.cfg file has been modified the Application Dispatcher is ready to be tested.

TESTING APPLICATION DISPATCHER
Before testing, the "default" service must be started. Select **Programs**, **The SAS System**, **IntrNet**, **Default Service** and **Start Interactively** from the **Start** menu. With the service started, testing can begin.

To test the broker, enter the address for the Web server where the broker is installed followed by the path the broker.exe file in the address field of a Web browser. For example, http://localhost/cgi-bin/broker.exe, where "localhost" is the address for the Web server. Replace "localhost" with the address of the Web server where the broker is installed. If the broker is working properly the Web page shown in Figure 6 will be returned.
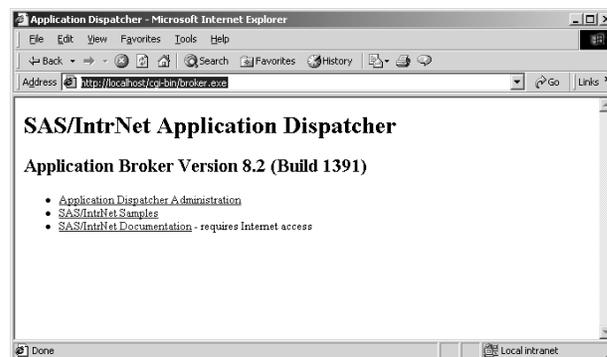

Figure 6: Application Broker

With the Broker working properly, the "default" service can be tested by entering the following address in the Web browser.

```
http://localhost/cgi-bin/broker.exe
?_SERVICE=default&_PROGRAM=sample.webhello.sas
```

This Web address illustrates the _SERVICE and _PROGRAM parameters that are passed to the broker. The broker will send the request to the "default" service which will execute the webhello.sas program. The results are returned to the browser as shown in Figure 7.

Figure 7: Hello World!

3

With the broker and "default" service installed and tested, the Application Dispatcher is ready for use.

SAMPLE APPLICATION

APPSTART.SAS PROGRAM
Before building the sample application, the program and data file locations for the application must be defined for the "default" service. The service is started with the SAS program named appstart.sas. This file is located in the service directory entered during creation of the "default" service. This directory can be opened by selecting **Programs, The SAS System**, **IntrNet**, **Default Service and Service Directory** from the **Start** menu.

The appstart.sas program contains a PROC APPSRV procedure which starts the service. The ALLOCATE FILE statement associates a SAS file reference with an external file or directory containing SAS programs while the ALLOCATE LIBRARY statement associates a SAS library reference to a SAS data library. For the service to use the file and library allocations they must be explicitly listed in PROGLIBS and DATALIBS statements.

The statements below will create the file and library allocations for the example that follows and should be placed directly above the RUN statement in the appstart.sas file

```
allocate library sugidata 'c:\sugi27\data';
allocate file sugicode 'c:\sugi27\code';
proglibs sugicode;
datalibs sugidata;
```

SAS programs and data set files for the sample application are located in the subdirectories as shown in the allocation statements above. The SAS data set "shoes" used in the sample application is a sample data set that comes with the SAS System and can be found in the SASHELP library.

ODS OUTPUT
The sample application below is a sales report that can be produced for different regions. A region can be selected from a list on the Web form. When the form is submitted, the broker will send the selection to the Application Server along with the program name where the program will be executed. The program will receive the region selected as a macro variable that will be resolved in a WHERE statement to restrict the report to the region selected. ODS will create HTML output from a PROC REPORT procedure and direct it back to the browser.

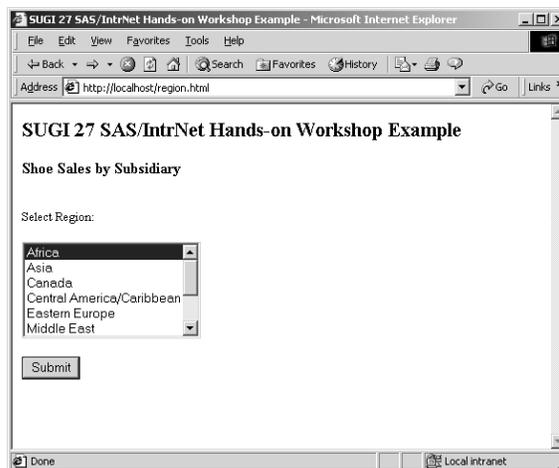The Web page containing the region selection is shown in Figure 8 with the HTML code following.

Figure 8: Shoe Sales by Subsidiary

```
<HTML>
  <HEAD>
    <TITLE>SUGI 27 SAS/IntrNet Hands-on Workshop
    Example</TITLE>
  </HEAD>
  <BODY>
    <FORM ACTION="/cgi-bin/broker.exe"
    METHOD="POST">
      <INPUT TYPE="HIDDEN" NAME="_service"
      VALUE="default">
      <INPUT TYPE="HIDDEN" NAME="_program"
      VALUE="sugicode.sales_by_sub.sas">
      <H2>SUGI 27 SAS/IntrNet Hands-on Workshop
      Example</H2>
      <H3>Shoe Sales by Subsidiary</H3>
      <P><BR>Select Region:</P>
      <P>
      <SELECT NAME="region" SIZE="6">
        <OPTION SELECTED>Africa</OPTION>
        <OPTION>Asia</OPTION>
        <OPTION>Canada</OPTION>
        <OPTION>Central America/Caribbean</OPTION>
        <OPTION>Eastern Europe</OPTION>
        <OPTION>Middle East</OPTION>
        <OPTION>Pacific</OPTION>
        <OPTION>South America</OPTION>
        <OPTION>United States</OPTION>
        <OPTION>Western Europe</OPTION>
      </SELECT>
      </P>
      <INPUT TYPE="SUBMIT" NAME="Submit"
      VALUE="Submit">
    </FORM>
    </BODY>
</HTML>
```

The most important part of this Web page is the form. The action attribute of the FORM tag contains the path to the broker.exe file. Following the FORM tag are two INPUT tags with the type attribute of "HIDDEN" which keeps them from being displayed on the Web page. The first INPUT tag with "_services" as the name attribute and "default" as the value attribute tells the broker that the "default" service is to receive the request. The second INPUT tag with "_program" as the name attribute contains the name of the program to be executed by the "default" service. Notice that the program has a three-part name. The first part, sugicode, is the program library that was allocated in the appstart.sas file. This tells the service where to find the program sales_by_sub.sas.

The SELECTION tag defines the selection list named "region." This item is passed to the "default" service by the broker and is received into the program as a macro variable named "region" and contains the value selected on the Web page.
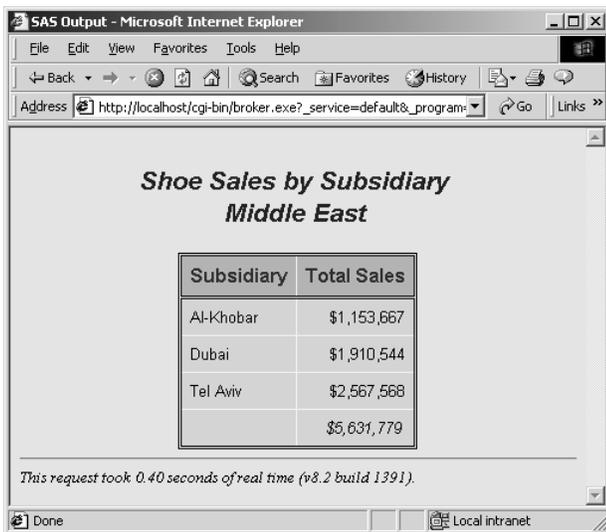

Figure 9: Asia Shoe Sales Report

The report in Figure 9 was produced when the "default" service executed the sales_by_sub.sas program listed below.

```
ods html body=_webout (dynamic) rs=none;

title1 'Shoe Sales by Subsidiary';
title2 "&region";

proc report data=sugidata.shoes;
  where region="&region";
  column subsidiary sales;
  define subsidiary / group;
  define sales / analysis sum;
  rbreak after  / ol summarize suppress skip;
run;

ods html close;
```

The program sales_by_sub.sas uses ODS to create HTML output from the PROC REPORT procedure. The BODY option on the ODS statement directs ODS to send the output to the "_webout" destination. "_webout" is a predefined file reference that sends the output back to the Web server and on to the Web browser. The WHERE statement in the PROC REPORT procedure resolves the macro variable "region" to restrict the report to the region selected.

Adding a graph to the report is quite simple. The code shown below includes additional parameters on the ODS statement, a GOPTIONS statement and a PROC GCHART procedure. The PATH and URL parameters allow the graphic to be added to the ODS output properly. The GOPTIONS statement sets graphic options for the production of the graph as a .GIF file. With these additions, the PROC GCHART procedure will add a graph to the report as shown in Figure 10.

```
ods html body=_webout (dynamic) path=&_tmpcat
(url=&_replay) rs=none;

title1 'Shoe Sales by Subsidiary';
title2 "&region";

proc report data=sugidata.shoes;
  where region="&region";
  column Subsidiary sales;
  define Subsidiary / group;
  define sales / analysis sum;
  rbreak after  / ol summarize suppress skip;
run;

goptions device=gif570
  ftext=swissl transparency
  noninterlaced htitle=1.5;

proc qchart data=sugidata.shoes;
  where region="&region";
  hbar3d region / sumvar=sales sum
  subgroup=product
  shape=cylinder
  patternid=subgroup
  legend=legend1;
  label product='Shoe Style';
run;

ods html close;
```


Figure 10: Shoe Sales Report With Graph

CUSTOM OUTPUT
There are times when information from a data set needs to be included in a Web page that is not generated by ODS. An example is the first Web page in this application where a region is selected for the shoe sales report. It is possible that the regions change, and it would be best to have the list populated dynamically from the data.

5

This custom Web page can be dynamically generated using a _NULL_ DATA Step with PUT statements to write the HTML to the "_webout" file definition which directs the HTML back to the browser. To illustrate this technique, the code from shoe_sales_report.sas listed below dynamically produces the first Web page in the example above from which a region is selected for the shoe sales report.

```
data regions;
  set sugidata.shoes (keep=region);
  proc sort nodupkey;
    by region;
run;

data _null_ ;
  set regions end=eof;
  file   webout;
  if _n_=1 then put
    /'<HTML>'
    /' <HEAD>'
    /' <TITLE>SUGI 27 SAS/IntrNet Hands-on
      Workshop Example</TITLE>'
    /' </HEAD>'
    /' <BODY>'
    /' <FORM NAME="form" ACTION="' "&_url" '"
      METHOD="POST">'
    /' <INPUT TYPE="HIDDEN" NAME="_service"
      VALUE="' "&_service" '">'
    /' <INPUT TYPE="HIDDEN" NAME="_program"
      VALUE="' "&_pgmlib" '.sales_by_sub.sas">'
    /' <H2>SUGI 27 SAS/IntrNet Hands-on Workshop
      Example</H2>'
    /'   <H3>Shoe Sales by Subsidiary</H3>'
    /'   <P><BR>Select Region:</P>'
    /'   <P><SELECT NAME="region" SIZE="6">';
  put
    / '      <OPTION>' region '</OPTION>';
  if eof then put
    /'      </SELECT>'
    /'      </P>'
    /' <INPUT TYPE="SUBMIT" NAME="Submit"
      VALUE="Submit">'
    /'    </FORM>'
    /' </BODY>'
    / '</HTML>';
run;
```

The first DATA step produces an unduplicated data set containing the variable region. This data set is read into the _NULL_ DATA Step that uses a PUT statement to write all the HTML down to the SELECT tag. This PUT statement only executes on the first observation in the data set , if _n_=1. The second PUT statement is executed for every observation and writes an HTML OPTION tag for each region. The last PUT statement is executed only for the last observation in the data set and writes the remainder of the HTML for the page.

Portions of the FORM and INPUT tags in this Web page are also dynamically generated. As mentioned above, when the broker sends the request to the service the HTML form items are received as macro variables in the SAS program. The SAS program also has reserved and special variables available as macro variables. Most of these variables contain information about the request being processed. These variables include the DSN or IP addresses of the Web server and Application Server, the path for the broker.exe, the program library, and the name of the service.

In the FORM tag the _URL macro variable is resolved to write the path for the broker. This will result in the following FORM tag.

```
<FORM NAME="form" ACTION="/cgi-bin/broker.exe"
METHOD="POST">
```

The first INPUT tag includes the macro variable _SERVICE that will write the name of the service to produce the INPUT tag below.

```
<INPUT TYPE="HIDDEN" NAME="_service"
VALUE="default">
```

In the second INPUT tag the macro variable _PGMLIB will write the name of the program library to produce the following INPUT tag.

```
<INPUT TYPE="HIDDEN" NAME="_program"
VALUE="sugicode.sales_by_sub.sas">
```

Using the macro variables to insert these items can make the application more flexible. For example, if the application needs to be moved to a different Application Dispatcher service or the name of the program library needs to be changed, the SAS code will not need to be modified.

Because this program does not receive any user selected parameters it does not need to be called from an HTML form. It can be called by entering the Web address with the parameters _SERVICE and _PROGRAM in the Web browser or link on a Web page as shown below.

```
http://localhost/cgi-bin/broker.exe
?_service=default
&_program=sugicode.sales_report.sas
```

Entering this address in the browser will send the _SERVICE and _PROGRAM parameters to the broker which will send the _PROGRAM parameter to the default service where the program will execute and send the Web page back to the browser as shown in Figure 8 above.

SESSIONS

The sample application above has illustrated the basics of developing a Web-enabled information delivery solution with Application Dispatcher. While these techniques can be used to develop more complex applications, the addition of "sessions" expands the possibilities for application development with Application Dispatcher.

When sessions are not used, requests to the Application Server are independent from one another. There is no easy way to have one request use information from another request. Sessions make it possible for one request to save macro variables and data sets that can be used by future requests. This is often referred to as "persistence" because the macro variables and data sets "persist" from one request to another. This is also known as "maintaining state."

A session is created with a call to the APPSERV_SESSION function as shown in the following code.

In macro:
```
%let rc=%sysfunc(appsrv_session(create));
```

In data step or SCL:
```
rc=appsrv_session('create');
```
When the session is created, the automatic variables _THISSESSION, _REPLAY, _SESSIONID, and _SESSIONCOOKIE are set to reflect the session id for the session. These variables can be used to create URLs or HTML pages that make new requests to the same session.

Macro variables to be saved in a session must begin with SAVE_. For example, the statement

```
%let save_region=Europe;
```

would create the macro variable "save_region" with the value "Europe" that would be available to future requests. Data sets saved in the library named "save" will be saved in the session for future use.

To access the macro variables and data sets in a session, the "_session" parameter with the session number must be passed to the broker which passes it on to the Application Server.

Sessions have a timeout setting which will cause the session to expire and all session data to be deleted. The timeout can be set with the following call to the APPSRVSET function with the timeout entered in seconds .

In macro:
```
%let rc=%sysfunc(appsrvset(session
timeout,300));
```

In DATA step or SCL:
```
rc=appsrvset('session timeout',300);
```

Sessions can be explicitly deleted with the following function call.

In macro:
```
%let rc=%sysfunc(appsrv_session(delete));
```

In DATA step or SCL:
```
rc=appsrv_session('delete');
```

To illustrate the use of sessions, the sales_by_sub.sas program has been modified below to create a session that will be used to save the region selection and data set for use in a report of sales by product for a selected subsidiary.

```
odds html body=_webout (dynamic) rs=none;

%let rc=%sysfunc(appsrv session(create));
%let save_region=&region;

data save.shoes;
  length lsubsidiary $300;
  set suqidata.shoes
  (where=(region="&region"));

  lsubsidiary = '<A HREF="' ||
  "%superq( thissession) "   ||
  '& program=' || "& pgmlib" ||
  '.sales by prod.sas&subsidiary=' ||
  urlencode(trim(subsidiary)) || '">' ||
  htmlencode(trim(subsidiary)) || '</A>';

run;

title1 'Shoe Sales by Subsidiary';
title2 "&region";

proc report data=save.shoes;
  column lsubsidiary sales;
  define lsubsidiary / qroup "Subsidiary";
  define sales / analysis sum;
  rbreak after  / ol summarize suppress skip;
run;

ods html close;
```

The first change is to create the session with the APPSRV_SESSON function call. The second %LET statement sets the session macro variable "save_region" to the value of the macro variable "region" passed from the HTML form. "save_region" will be available to the second report.

The DATA Step added to the program reads the shoes data base with a WHERE option to include only the region selected. The variable "lsubsidiary" is a hypertext link that calls the second report. This is a technique often referred to as "Drill Down." The necessary HTML with required parameters are added to the values of the variable subsidiary to created links that will call the next report. The "_thissession" macro variable resolves to provide the path for the broker and all parameters to identify the Application Server service, port and session, as show below.

```
/cgi-bin/broker.exe?_service=default
&_server=127.0.0.1&_port=5001
&_sessionid=5job.B0lH52
```

The shoes data set containing records for the selected region written to the "save" library will be saved in the session.

The new version of sales_by_sub.sas will produce the Web page shown in Figure 11 with each subsidiary being a hypertext link to the sales by product report for the subsidiary selected.

Figure 11: Shoe Sales by Subsidiary

When a link in the Shoe Sales by Subsidiary report is selected, the Shoe Sales by Product as shown in Figure 12 is produced for the subsidiary selected.

Figure 12: Shoe Sales by Product

The sales_by_pord.sas program listed below is quite simple. Because the program is executed in the session created by the previous report the saved macro variable and data set are available. The save_region session macro variable is resolved in the second TITLE statement. The PROC REPORT procedure uses the save.shoes data set that was saved in the session from the previous report.

```
ods html body=_webout (dynamic) rs=none;

title1 "Shoe Sales by Product";
title2 "&save_region";
title3 "&subsidiary";

proc report data=save.shoes;
  where subsidiary="&subsidiary";
  column product sales;
  define product / group;
  define sales / analysis sum;
  rbreak after  / ol summarize suppress skip;
run;

ods html close;
```

This example illustrates how macro variables and data sets can be saved in a session and accessed by subsequent requests. This can greatly increases possibilities for application development.

CONCLUSION

SAS/IntrNet Application Dispatcher is one of several choices for developing Web-enabled information-delivery solutions. It is an excellent choice for those with SAS programming experience and little or no experience with Web application development. It is also an excellent choice for connecting existing SAS applications to the Web. While the techniques used in the sample application are very basic, they provide the basis for developing more complex applications that connect the SAS System to the Web.

CONTACT INFORMATION
The author may be contacted at:

Vince Timbers
Penn State
101 Shields Building
University Park, PA 16802
Phone: (814) 865-4253
e-mail: vlt@psu.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brands and product names are registered trademarks or trademarks of their respective companies.