

SAS® on the Web: How do I get There from Here?

Carol Martell, UNC Highway Safety Research Center, Chapel Hill, NC
Ruth Marinshaw, UNC Academic Technology and Networking, Chapel Hill, NC
Eric Rodgman, UNC Highway Safety Research Center, Chapel Hill, NC

ABSTRACT

Do you ever wonder what it would take to start using those SAS® skills in a web application? Several pieces have to be in place if you want to play. This tutorial will tell you what those pieces are and how to either ask for them or implement them yourself. Your system and web server administrators may not be SAS users. Being informed will help you make friends while asking for what you need!

INTRODUCTION

SAS/IntrNet® software, one of the SAS web technology products, is a set of CGI tools. We examine two of the available tools, Application Dispatcher and htmSQL, in the Solaris environment. For each tool there must be a machine functioning as the Web Server and a machine function as the SAS Server. The Web Server and SAS Server may be on one machine or two different machines. Configuration options are plentiful. For this paper we simply choose one option for each tool and describe the installation and configuration steps. You should read the documentation available on the SAS Web site:
<http://www.sas.com/rnd/web/intnet/doc>.

APPLICATION DISPATCHER

The Application Dispatcher is, in a sense, exactly what its name implies. A request to run an application is dispatched, or sent, from a browser window through a broker to a SAS session and the results are returned to the browser window. Configurations tell the broker where the SAS session(s) and application libraries are. In some situations, the SAS session is not yet running and must be launched. In others, SAS is already running and session communication is via a port. These various modalities of sessions are defined as named services in the broker configuration file, and the desired service name is included in the web request. Each service is started, whether by request or in advance, by SAS code that includes specification of the program library locations.

Using the Application Dispatcher allows SAS to be the back end for an HTML-based interface that returns the results from a SAS job to the browser window. For example, you might present in the browser window an HTML page that displays the names of all variables from a SAS data set. The end user might select the variables of interest to him, then select procedures of interest, and submit those choices. This input is sent to a CGI program (broker) on a web server. The broker takes this information and passes it on to a SAS Application Server. The Application Server is running SAS; it takes the input passed from the Broker and invokes the requested SAS program. SAS output is then passed back to the broker and streamed to the user's browser window. The end user does not have to know how to use SAS; indeed, the

end user does not need to know that SAS was run. What the end user sees is that he requested information, and it was displayed for him.

The Application Dispatcher has three basic component requirements: the browser; a CGI program, referred to as the broker; and a SAS Application Server. **To use the Application Dispatcher, you must have a web server available, and you must also have SAS running on a server.** For the purposes of this paper, we are using SAS 8.2 and an Apache web server, both running in a Solaris environment.

INSTALLATION AND CONFIGURATION STEPS

BRIEFLY

SAS Server:

1. Install SAS/IntrNet components from the Version 8 installation media.
2. Add a service to the /etc/services file
3. Configure and start the service, using inetcfg.pl

Web Server:

1. Install the Application Broker, contained in the CGI Tools, downloadable from the SAS Web site.
2. Add the service definition to the broker.cfg file

IN-DEPTH

The steps to configuring Application Dispatcher Services as a socket service follow:

1. Install SAS 8.2 on your SAS server. Be sure to install the SAS/IntrNet components with this installation.
2. Make sure that you have a web server running and available for use. You will need to know the URL for accessing files on the web server. From or with your web server administrator, determine where the web server's cgi scripts are stored. For installation of SAS/IntrNet components on the web server, the person doing the installation will need to have write permission to that cgi script directory. On many servers, this will be /cgi-bin in the directory where the web server software is installed. In addition, determine what the document root is on your web server (typically, /htdocs in the directory where the web server software is installed).
3. On your SAS server, download the latest CGI tools for Release 8.2 of SAS/IntrNet software from <http://www.sas.com/rnd/web/intnet/index.html>. You will be instructed to download them to a temporary directory on your SAS server.
4. Extract the files from the compressed download file and run the INSTALL script that was included in the download. As noted earlier, the user running this script must have write access to directories on the web server and the SAS server. As part of the INSTALL process, you will specify a location for sample files, the path for cgi scripts, and the URL for those. The INSTALL process puts the Application Broker and broker configuration files

in the cgi-bin directory on the web server.

5. On the SAS server, run the **inetcfg.pl** utility in `ISASROOT/utilities/bin`. The script will prompt you for a location to store SAS/IntrNet service files (for example, `/local/pkg/default`). Select the **socket** type of service, and use the service name of **default**. For the most basic installation, only create one service to run on one server. Since you are setting up a socket service, you will be prompted to identify a particular port to use. After consulting with your system administrator, identify a port number.

6. Ask your system administrator to create an entry in `/etc/services` that identifies the port number you referenced in `inetcfg.pl` and also names the service (as above, this name is **default**). For example, if you specified port 5228 for service default, a line such as the following would be added to `/etc/services` on your SAS server:

```
default      5228/tcp      #SAS socket srvc
```

7. After running **inetcfg.pl** and modifying `/etc/services`, your web server administrator should modify the **broker.cfg** file in the `/cgi-bin` directory on the web server. Several different parameters in this file will need to be changed. First, you will need to modify the socket service entry to reflect the appropriate parameters for your system. For example, your entry might look like this if you specified to use Port 5228 on the SAS server named `myserver.unc.edu`:

```
SocketService default "Reuse existing
session"
ServiceDescription "Pages reference this
generic server when they don't care which
service is used."
ServiceAdmin "Your Name"
ServiceAdminMail your\_name@wherever
Server myserver.unc.edu
Port 5228
# Remove the following line for any
servers before V8.1
FullDuplex True
```

Also modify **broker.cfg** to change the `SelfURL` entry to the URL for your web server and broker. For example, if I used `webserver.mywebserver.unc.edu` and my CGI scripts are stored in the `cgi-bin` directory, my entry would be

```
SelfURL http://mywebserver.unc.edu/cgi-bin/broker
```

You can also change the `Administrator` and `AdministratorMail` entries in **broker.cfg** to identify the person responsible for the service.

You should also modify the entry `DefaultService` to show the service you defined (for us, default):

```
DefaultService default
```

Many other settings are configurable in this **broker.cfg** file. You and your administrators should read the document <http://www.sas.com/rnd/web/intrnet/doc/dispatch.pdf> to discuss what other options to modify. But the options shown above are the most basic.

Start your socket service by running the **start.pl** script on the SAS server. This script was created when you ran `inetcfg.pl`. The **start.pl** script will be in the directory that

you identified in `inetcfg.pl` as the root directory for SAS/IntrNet services. (In my example above, I used `/local/pkg/default` for my root location, so I would find **start.pl** in `/local/pkg/default`.)

Test whether your service is available by pointing your browser to <http://yourwebserver/cgi-bin/broker>? (Note that the question mark is part of the URL!!)

8. At this point, you are not finished with your configuration. In that same root directory for SAS/IntrNet services, `inetcfg.pl` created a file, **appstart.sas**. You now need to configure that file to include the librefs and associated paths for programs that will be run through the Application Dispatcher. The way the Application Dispatcher works is that the end user displays an HTML-coded page. The end-user uses check boxes, text entry fields, or other features to select information that SAS will process. The information is passed to the broker (on the web server). Coded in the information sent to the broker is a reference to the appropriate SAS program to run on the SAS server. That reference typically is `libref.program.sas`. In the **appstart.sas** file, you must have those librefs defined to SAS will know where to find the requested program. We recommend that you define program libraries here, not data libraries. It is a better practice to define data libraries with librefs in the SAS programs that run for a given request, rather than defining them in the `appstart.sas` file. The `appstart.sas` file can be modified at any time, but if you are running a socket service, you should stop and restart your service when you modify `appstart.sas`.

9. You will want to work with your system administrator to ensure that the `start.pl` script, which starts the Application Dispatcher services, is run whenever the system is rebooted. Your system administrator can automate this process. You should also discuss ways to verify that the service is running normally. SAS provides some utilities for administrative functions like this.

htmlSQL

With `htmlSQL`, the program and results are together in one file. A query selects items to display. Query execution is triggered with a browser request and results are displayed in the format dictated by the developer. Input files contain a mix of HTML and `htmlSQL` directives. When a browser requests one of these input files, the `htmlSQL` directives are processed, yielding results that are displayed in the browser window. Viewing the page source will not reveal the `htmlSQL` directive code, since only the results are sent to the browser window. Access to data is through a SAS/SHARE® server.

INSTALLATION AND CONFIGURATION STEPS

SAS Server installation and configuration steps:

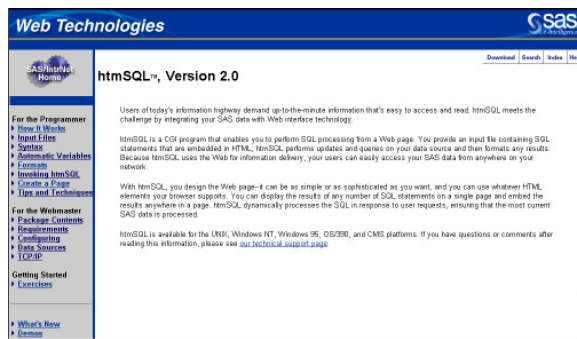
1. Define a service on the SAS application server
2. Start a SAS/SHARE server

Web Server installation and configuration steps:

1. Install the `htmlSQL` CGI tool on the Web server
2. Configure the installation by modifying `htmlSQL.cfg`
2. If the SAS Server is a different machine, define a service identical to that defined on the SAS Server

Documentation for `htmlSQL` is on the SAS Web site:

<http://www.sas.com/rnd/web/intrnet/htmSQL>



IN-DEPTH

THE SAS SERVER AND htmSQL

Data is accessed in htmSQL through a SAS/SHARE® server. SAS/SHARE is a sophisticated product not covered in the scope of this paper. We describe a minimal configuration in the following three steps and encourage the reader to further investigate SAS/SHARE capabilities.

1. Each SAS/SHARE server uses a defined service on the SAS application server. In the Solaris environment, a service is defined by adding a line to the `/etc/services` file (your UNIX administrator has 'write' permission)

```
myshare    5555/tcp    # SAS/SHARE server
```

The `/etc/services` line above defines a service named 'myshare' on port 5555, an arbitrarily chosen unused port. (When the Web server is a different machine, the same entry should appear in `/etc/services` on both machines)

2. The SAS/SHARE server references the defined service. Submit the following SAS code to start a SHARE server named 'myshare'. For our example, we choose to define a library when we start the server by preceding PROC SERVER with the LIBNAME statement.

```
LIBNAME mylib 'path to my library';
PROC SERVER ID=myshare AUTHENTICATE=OPT
COMAMID=TCP;
```

AUTHENTICATE=OPT prevents us from having to provide authentication with each htmSQL page. The COMAMID parameter sets the communication access method for the SHARE server.

3. Because the SHARE server must be running to respond to data requests, some form of automatic resubmission of the PROC SERVER is a good idea. Suppose the PROC SERVER job is named `servit.sas`. A script, scheduled to run periodically, could look for the process named `servit` and resubmit the job if that process did not exist:

```
#!/bin/sh
# looking for job named servit
if (ps -ef | grep servit | grep -vc grep)
then
  echo "need not run job"
  exit 1
else
  echo "must run job"
```

```
/bin/sas /path/servit.sas
exit 1
fi
```

If the preceding script were stored in the executable file named `restartit`, the crontab entry to check every minute would be:

```
* * * * * /.../restartit > /dev/null 2>&1
```

THE WEB SERVER AND htmSQL

Files to be installed on the Web server may be downloaded from the SAS Web site. All of the CGI Tools for SAS/IntrNet are included in a single tar file named `websrv.tar`.

The executable file, named `htmSQL`, belongs in the Web server directory for CGI executables. The Web server administrator will know the location.

The execution of htmSQL is configured by the file `htmSQL.cfg`, which belongs in `cgi-bin` directory. A default template named `htmSQL.cfg_v8` is included in `websrv.tar`. Copy the template to `htmSQL.cfg`. Helpful documentation comments are included in the file, which can be modified with a text editor. Most options can remain at the default setting. The DATASRCFILE option points to a file of predefined data sources and associated data libraries. This data source file is not required and we do not use it for our example, since we define a libref in the PROC SERVER code. We ensure that the DATASRCFILE option is commented out in `htmSQL.cfg`.

It is convenient to have the Web server automatically recognize htmSQL files by the filename extension. The Web server administrator can do this by adding two lines to `httpd.conf` for an Apache Web server. The first line causes files with extension `.hsql` to be processed with htmSQL. The second line provides the path to the htmSQL executable:

```
AddHandler htmSQL .hsql
Action htmSQL /exepath/htmSQL
```

When the Web server is not the same machine as the SAS server, ask the Web server administrator to duplicate (exactly) the service definition in `/etc/services`.

USING htmSQL

Directives are enclosed in curly braces. Most function in pairs to delimit sections of code: `{query}...{query}`. A `{query}` section contains one or more `{sql}` sections. Each `{sql}` section can have one associated `{eachrow}` section to format the results and one `{norows}` section for different action if the query results are empty. While the `{query}` section is for read access, the `{update}` section allows write access. Each `{update}` section contains one or more `{sql}` sections, each of which may have an associated `{success}` and `{error}` section.

Variables are preceded by `&` and enclosed in curly braces: `{&var1}`. Variables may be passed in as name-value pairs with the browser request or come into existence in `{sql}` sections. There are automatic variables as well.

OTHER INSTALLATION POSSIBILITIES

SAS provides you with three types of Application Dispatcher services: `launch` (a new SAS session is begun whenever a request is made), `socket` (SAS is running all the time, waiting for a request) and `pool` (multiple SAS servers are available). You should read about these in detail before configuring your installation. The choice of service type may be influenced by how your web and SAS services are structured. For example, a `launch`

service requires that the web server and SAS server be on the same machine. For a pool service, the web and SAS server may or may not be the same machine.

The most basic htmSQL setup does not include the use of a data source file. Consider the following issues to decide your requirements. Using the file can provide extra security in several ways by hiding the server name and port, the SHARE server user access password, or the userid and password for a secure mode server from the developer. The server name, port or service, and authentication elements can be placed directly into htmSQL code, sidestepping the need for the data source file. If authentication is required, it is more convenient to have a data source file and avoid having to authenticate for each query or update. Another consideration is that htmSQL referencing a predefined data source need not be updated if the mapping behind that data source changes. One would create a data source file use the dsdef script in websrv.tar.

SAMPLE CODE

We begin with a simple listing of a variable from the first 10 observations of a table.

Application Dispatcher code:

```
libname cps 'path/to/data';
data _null_ ;
set cps.countyofservice(obs=10) ;
file _webout ;
if _n_ =1 then do ;
  put '<br>' ;
  put '<br>'
  'List of first 10 CPS Service Counties'
  '<br><br>' ;
end ;
put servicecounty '<br>' ;
run ;
```

Having saved this code in a file named ex1.sas in the program library 'mylib', we can invoke this from a browser window by entering (substituting the web server name and path for ...):

```
http://.../broker8?_program=mylib.ex1.sas
```

This could be placed on a web page as a hyperlink:

```
<a href=
http://.../broker8?_program=mylib.ex1.sas>
run the job</a>
```

It could be the action for a web form:

```
<form action=http://.../broker8 method="post">
<input type="hidden" name="_program"
value="mylib.ex1.sas">
```

The results come to the browser window and look like this:

List of first 10 CPS Service Counties

```
Alamance
Ashe
Bertie
Brunswick
Buncombe
Cabarrus
Caldwell
Carteret
Caswell
Chowan
```

The htmSQL code to do the same thing:

```
List of first 10 CPS Service Counties<br><br>

{query server="myshareserver"}

{sql}
  select servicecounty
    from mylib.countyofservice(obs=10)
{/sql}

{eachrow}
  {&servicecounty} <br>
{/eachrow}

{/query}
```

This code, placed in a file named 'ex1.hsql' in a web server directory and displayed in a browser window, yields identical results:

List of first 10 CPS Service Counties

```
Alamance
Ashe
Bertie
Brunswick
Buncombe
Cabarrus
Caldwell
Carteret
Caswell
Chowan
```

One obvious advantage to htmSQL is that put statements do not come into play. Suppose your results needed to have a hyperlink to a file and the name of that file is found in a variable named var1. Also suppose that you want the visible text for the link to be var1. In the Application Dispatcher coding put statements can get complicated:

```
Put
  '<a href="pathtodirectory/' var1 '>' var1
  '</a>';
```

The equivalent code in htmSQL would be:

```
<a href="pathtodirectory/{&var1}">{&var1}</a>
```

SAS procedure output can be dynamically produced using the Application Dispatcher but not htmSQL.

DATA step coding can be used in the Application Dispatcher, but only SQL queries may be used in htmSQL.

ODS HTML can be used with the Application Dispatcher but not with htmSQL.

These sorts of differences will come under consideration as you plan your web applications.

CONCLUSION

Given a programming goal, SAS provides numerous pathways to that goal. Environmental circumstances can bring your goal closer or make it a little harder to reach. We have discussed installation, configuration and use of two SAS/IntrNet CGI tools. Your programming skills and your web and SAS server environment can influence the product mix you choose for your web application.

CONTACT INFORMATION

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

Carol Martell
UNC Highway Safety Research Center
730 Airport Rd, CB# 3430
Chapel Hill NC 27599-3430
Work Phone: 919-962-8713
Fax: 919-962-8710
Email: carol_martell@unc.edu
Web: www.hsrc.unc.edu

Ruth Marinshaw
Applications Support Group
Academic Technology & Networks
38 Phillips Hall, CB #3455
Chapel Hill NC 27599-3455
Work Phone: 919-962-4314
Fax: 919-962-5664
Email: ruth@unc.edu
Web: <http://help.unc.edu>

Eric Rodgman
UNC Highway Safety Research Center
730 Airport Rd, CB# 3430
Chapel Hill NC 27599-3430
Work Phone: 919-962-8709
Fax: 919-962-8710
Email: eric_rodgman@unc.edu
Web: <http://www.hsrc.unc.edu>