The SAS/IntrNet™ Application Dispatcher

Donald J. Henderson, SAS Institute Inc.
Edmund Burnette, SAS Institute Inc.
Vincent DelGobbo, SAS Institute Inc.
John Leveille, SAS Institute Inc.

ABSTRACT

The Application Dispatcher component of SAS/IntrNet Software allows a user to run a specified SAS® Application program **on demand** and return the generated *Internet Content* to a Web Browser.

The Application Dispatcher is one of the compute services of SAS/IntrNet software and is composed of the following components:

1. The Application Broker CGI
   The link between the Web Browser and the compute services. Resides on the Web Server and forwards requests for processing to the Application Server(s).

2. The Application Server
   An SCL program that listens for requests from the Dispatcher on a defined TCP/IP socket. The Application Server runs the specified SAS Application.

3. SAS Applications
   SAS programs that perform specified processing. They generate the results (*Internet Content*) which are sent back to the Web Browser by the Application Server via the Dispatcher. SAS/IntrNet software includes several Dispatcher Applications. The majority are customer supplied.

This paper will provide an overview of the SAS/IntrNet Application Dispatcher and will include several demos of SAS/IntrNet Applications and samples available from SAS Institute.

NOTE: *At the time this paper was finalized the SAS/IntrNet Application Dispatcher was about to be released in alpha form. It is expected that feedback and comments from the alpha release will likely change the exact functionality of the Application Dispatcher. For more up-to-date information on the current feature set and capabilities of the SAS/IntrNet Application Dispatcher, including complete documentation, please visit the web site for the Institute's Web Tools:*

http://www.sas.com/rnd/web

and follow the links for products, specifically the Application Dispatcher.

INTRODUCTION

The Application Dispatcher is a general-purpose gateway that allows a user to invoke SAS code on the Web server machine or another machine and get the results back in HTML or other formats. An already running SAS session performs the desired processing. (Note that the capability to launch a new SAS session for each request is planned for the next scheduled release of the SAS/IntrNet Application Dispatcher.)

The Broker CGI component is an ACTION invoked by an HTML form; hidden fields in the form select a predefined service to access and identify the code to run. Other fields such as text areas and checkboxes are converted to SAS macro variables so they may be used by the SAS code to customize the output returned to the user. Long text strings are broken up into multiple variables.

The Application Server is a SAS SCL program which listens for input from the Broker CGI on a defined TCP/IP socket; runs the requested application and sends the results back to the users browser via the Broker CGI.

This paper explores how to define what program should be run, where it should be run, how parameters are passed to the program from HTML and how the results are returned to the browser.

PROCESSING FLOW

Web applications are constructed so the user's browser provides the user interface functions of querying the user as to what they want to do as well as displaying and formatting the results. In most cases the request for processing starts with the user filling out a form such as that shown in Figure 1.

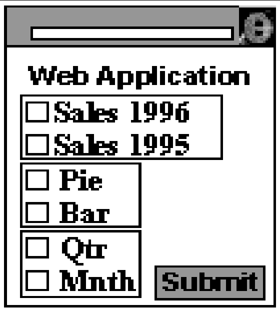


Figure 1: Sample Form

Upon pressing the submit button, the Application Dispatcher begins processing, which is shown graphically in Figure 2.

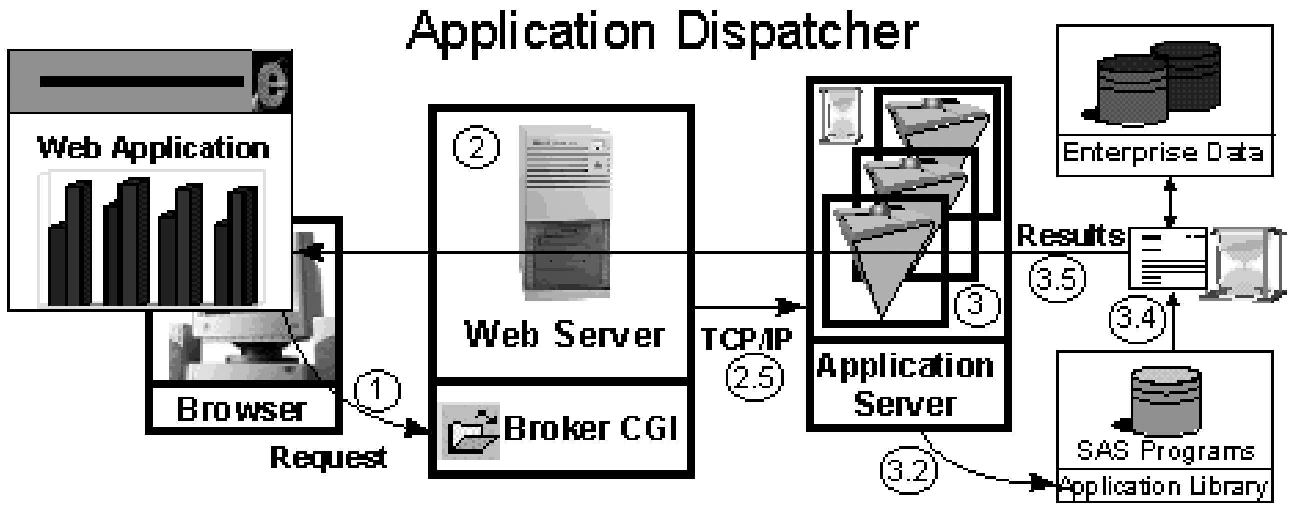


Figure 2: Processing Flow

1 The user clicks on the *submit* button.

2 The Broker CGI program begins execution on the web server and does the following:

2.1 Verifies that all of the parameter names to be passed to the Application Server are valid SAS names.

2.2 Creates, if necessary, multiple values for any parameter as discussed in Appendix A.

2.3 Checks a defined configuration file to verify that the field *_service*, which specifies what Application Server is to perform the processing, is defined.

2.4 If the selected value of *_service* defines a pool of Application Servers, the Broker CGI will select one of them to forward the request to.

2.5 Using the TCP/IP socket (the machine name/IP address and port number), defined in the configuration file for the *_service*, the request is forwarded to an Application Server.

2.6 If the Application Server does not respond (i.e., is not listening on the TCP/IP socket) will either:

2.6.1 Select another Application Server if the *_service* defined a pool of Servers.

2.6.2 If the *_service* is not a pool of Applications Servers or if none of the Application Servers respond, it generates an error page that is returned to the user's browser.

3 The Application Server receives the request by reading the data sent from the Broker CGI on the TCP/IP Socket. It's primary tasks are:

3.1 Define macro variables for parameters passed from the HTML page.

3.2 Examine *_program*, passed from the HTML page. It's value is the program name to run. The Application Server determines if the program is located in an administrator defined Application Library.

3.3 Define the special reserved filenames:

3.3.1 _WEBOUT for text/html output.

3.3.2 _GRPHOUT for graphical output.
NOTE: In the next scheduled release of Application Dispatcher, a separate filename for graphical output may not be needed.

3.4 If the program is found, the Application Server will run the program. If not, the Application Server will return an error page to the user's browser.

3.5 Return the results to the user's browser.

INVOKING DISPATCHER - A SIMPLE EXAMPLE

A SAS/IntrNet Application Dispatcher Application starts with a request from a browser. The HTML references the Broker CGI component and provides the two required fields, *_service* and *_program*. An example in which the browser user selects a data set to display will be used to illustrate this.

Figure 3 shows the HMTL form displayed in a browser that the user sees. The output generated when the user

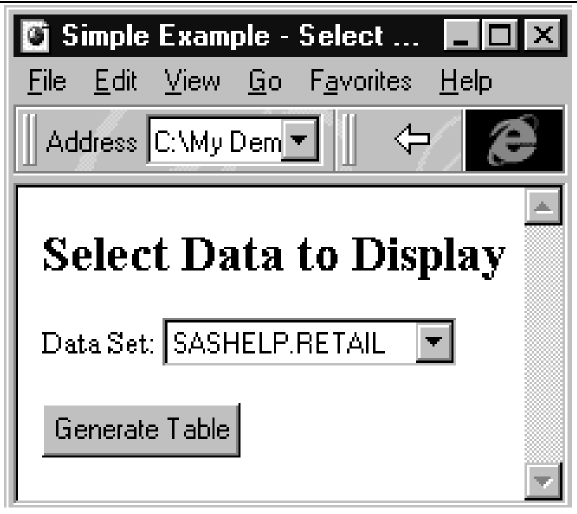
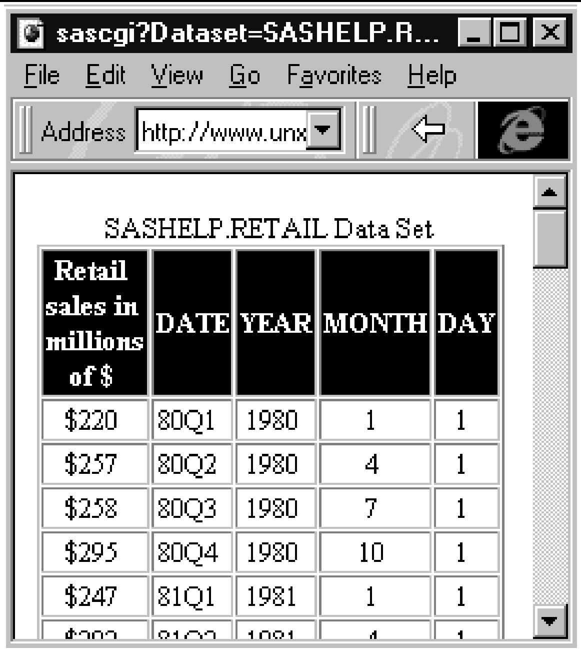


Figure 3: User's Browser Display



SASHELP.RETAIL Data Set

Retail sales in millions of $	DATE	YEAR	MONTH	DAY
$220	80Q1	1980	1	1
$257	80Q2	1980	4	1
$258	80Q3	1980	7	1
$295	80Q4	1980	10	1
$247	81Q1	1981	1	1

Figure 4: The Output

selects the *Generate Table* button is displayed in Figure 4. The HTML for the form is displayed in Figure 5.

The *action* attribute of the *form* tag specifies the location of the Broker CGI component and the architecture of CGI causes that program to begin executing on the Web Server when the “Generate Table” button is selected. The Broker CGI checks in a configuration file for the

```
<html>
<head>
<title>Simple Example - Select Data to Display</title>
</head>
<body bgcolor="#FFFFFF">
<form action="http://www.unx.sas.com/cgi-bin/broker">
<h2>Select Data to Display</h2>
Data Set:
<select name="Dataset">
   <option>SASHELP.RETAIL</option>
   <option>SASHELP.REVHUB</option>
</select>
<p><input type="submit" value="Generate Table">
<input type="hidden" name="_program"
                        value="mktdemo.example1.sas">
<input type="hidden" name="_service"
                        value="mkt-appserver">
</form>
</body>
</html>
```

Figure5: The HTML Source

details of the service *mkt-appserver*, as designated in the HTML hidden field *_service*. Once the Application Server begins its processing, it checks the value provided for the field *_program*. The details for *_service* and *_program* are briefly discussed in the following subsections. Complete documentation can be found for these topics at

http://www.sas.com/rnd/web.

Specifying Services in the Configuration File

In the Dispatcher configuration file an administrator defines "services" which are logical compute servers for user’s applications to use. In the alpha version the only supported service types are Socket services which connect to a SAS Application Server which is already running on some machine and listening on a TCP/IP port. A socket service is defined as follows, with the values in italics filled in with the appropriate values:

SocketService *serviceid* "*A long description goes here*"
  Server *nodename|IPaddress*
  Port *5001*

This can be extended to work with a pool of server machines and ports. In the simplest case, another identical server is running on another machine, and half the requests should be directed to the first machine and half to the other. To do this, the Server line is changed to:

  Server *nodename1|IPaddress1 nodename2|Ipaddress2*

Because the Port line was not changed, Dispatcher assumes the same port on both machines. In many environments it makes sense to start more than one Application Server on each machine; the number will depend on the nature of jobs and the capacity of the machine. To define that three Applications Servers are running on each machine and are listening on ports 5001, 5002, and 5003, change the Port line to read:

```
   Port 5001 5002 5003
```

This means there are three Application Servers running on each of two machines, for a total of six Application Servers. Each server has a 1/6 chance of getting sent a request to process. Because CGI programs, including the Broker CGI, are stateless, the Broker CGI simply picks a random number between 1 and 6 and uses that number to determine the server/port combination to which it will forward the request.

Multiple Server and Port lines can be used to accommodate asymmetrical configurations. For example:

```
   Server A B
   Port 1000 2000
   Port 3000
   Server B
   Port 1500 2500
```

In this example, the server/port combinations are A:1000, A:2000, A:3000,B:1000, B:1500, B:2000, B:2500, and B:3000. Each has a 1 in 8 chance of being selected.

Since some machines are more powerful than others, Dispatcher allows the assignment of weights to machines, for example,

```
   Server machine1 machine2*5
```

Machine1 has a 1 in 6 chance of being assigned a request, while machine2 has a 5 in 6 chance.

Specifying the Program to Run

The field *_program* specifies the name of the program to be invoked. The program can be:

1. An external file containing SAS source code.
2. A SAS source entry containing SAS source code.
3. The name of a compiled SAS macro.
4. A SAS SCL entry containing a compiled SCL program.

The programs must be located in an directory that is defined by an administrator to the Application Server. The first node in the name specifies the nickname for the directory. This nickname can also be thought of as a libname. However, it is more flexible since we can also refer to external files contained in that directory.

Consider the example above where the value of *_program* was *mkt.example1.sas*. The Application Server looked in the directory defined to it as *mkt* for the external file *example1.sas*. The syntax for each of the above types of programs that can be run is:

1. <u>An external file containing SAS source code</u>.
   A three level name. The first node in the name is the directory (i.e., the nickname for the directory). The second node is the actual filename as it is defined to the server operating system. The third node for external files is always **sas**. The Application Server recognizes a three level name whose third node is sas as an external file.
2. <u>A SAS source entry containing SAS source code</u>.
   The four level name of the source entry, including .SOURCE. The first node is the directory or libname, the second node is the catalog name, the third node is the actual entry name and the fourth node is **source**.
3. <u>The name of a compiled SAS macro</u>.
   The four level name of the macro entry, including the .MACRO) for a compiled SAS macro entry. The first node is the directory or libname, the second node is the catalog name, the third node is the macro name and the fourth node is **macro**. Note that the macro source code need not be located in the same catalog.
4. <u>A SAS SCL entry containing a compiled SCL program</u>.
   The four level name of the SCL entry, including .SCL. The first node is the directory or libname, the second node is the catalog name, the third node is the name of the compiled SCL entry and the fourth node is **scl**.

In order for the user's browser to display the output, the program must generate appropriate *Internet Content* such as HMTL, GIF, JPEG output. The SAS program that generated the output in Figure 4 is shown in Figure 6. This program uses the DS2HTM macro which is one of the Web Publishing Tools available from SAS Institute. The Web Publishing Tools are freely downloadable from **http://www.sas.com/rnd/web**. These tools can be used in SAS/IntrNet Application Dispatcher applications to generate *Internet Content*. Thus, it is not necessary for SAS programmers to know HTML.

```
%ds2htm(data=&dataset,
             htmlfref=_webout,
             openmode=replace,
             bgtype=color,
             bg=white,
             clbgcolr=black,
             clcolor=white,
             runmode=S,
             caption=&dataset Data Set)
```

Figure 6: Sample SAS Program.

Note the use of _WEBOUT as the fileref to which the output is to be directed. Also note that the data set selected in the HTML page is referenced in the SAS program as a macro variable.

A GRAPHICS EXAMPLE

Generating graphical output and returning it to the users browser is also straightforward. Figure 7 shows a program that generates a bar chart from the SASHELP.RETAIL data set. Note the use of macro variable references for the chart variable value (&cvar) and the analysis variable (&svar). The values for these are specified in the HTML form.

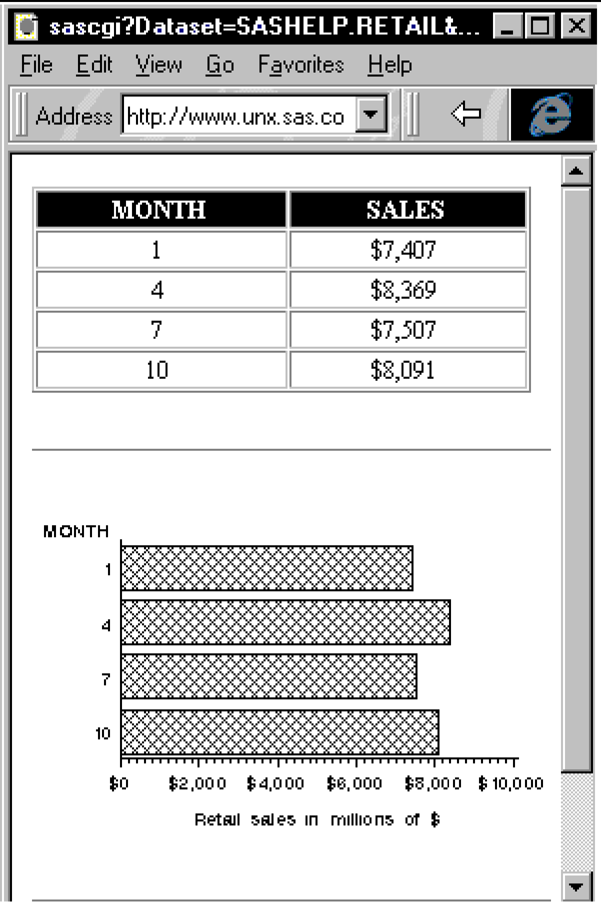
```
goptions reset=all gsfname=_grphout gsfmode=replace
        dev=gif260 ftext=swiss htext=1.5
        gprolog='Content-type: image/gif' '0D0A0D0A'x;
title;

proc gchart data=sashelp.retail;
  hbar  &cvar/sumvar=&svar discrete nolegend nostats;
run;
```

Figure 7: A Graphics Program

The program uses the GIF260 driver to generate a GIF file. The GIF260 driver is another of the Web Publishing Tools now available. The gprolog option is used to generate the header needed by browser to recognize that the output being sent to the browser is gif output.

GENERATING RESULTS CONTAINING HTML TEXT AND GRAPHICS

SAS/IntrNet Application programs can create either HTML or graphics. One program cannot do both at the same time. One program can, however, generate an HTML page containing an image tag which calls the broker a second time thereby attaining a dynamic page with embedded graphics.
Consider an example where from the initial HTML form, the user will select two variables - a grouping variable and an analysis variable. For our example, we will use the SASHELP.RETAIL data set. The desired output is an HTML table that shows the sum of the analysis variable for each value of the grouping variable.


sascgi?Dataset=SASHELP.RETAIL&...
File Edit View Go Favorites Help
Address http://www.unx.sas.co
MONTH
1
4
7
10
$0 $2,000 $4,000 $6,000 $8,000 $10,000
Retail sales in millions of $

MONTH	SALES
1	$7,407
4	$8,369
7	$7,507
10	$8,091

Figure 8. Combined Text/Graphic Output

Below the HTML table we wish to display the same data in a bar chart. The output is shown in Figure 8. This requires that two programs be run. The first will generate the HTML table and will generate an HTML IMG tag that will run another Dispatcher Application to generate the graph and is shown in Figure 9.

Here, the SAS/IntrNet Application Dispatcher is run from an embedded HTML reference instead of from the ACTION parameter of an HTML form. The reference in the IMG tag invokes the program shown in Figure 7. The program shown in Figure 9 also shows the use of the special macro variable *_url* which is made available to Dispatcher Applications and whose value is the name of the Broker CGI component. The macro variable *_url* can be used by a Dispatcher Application that needs to generate HTML which contains other references to Dispatcher Applications. The special macro variable

```
proc sql;
 create view totals as
  select &cvar, sum(&svar) as &svar format=dollar8.
  from sashelp.retail group &cvar;
%ds2htm(data=totals,
                var=&cvar &svar,
                htmlfref=_webout,
                openmode=replace,
                bgtype=color,
                bg=white,
                clbgcolr=black,
                clcolor=white,
                runmode=S)
data _null_;
 file _webout mod;
 put '<IMG SRC="' "&_url?_service=mkt-appserver"
    '&_program=mktdemo.chart.sas&cvar=' "&cvar"
    '&svar=' "&svar" '">';
run;
```

Figure 9: Combined Text/Graphic Program

_service is also used. Recall that *_service* is a required parameter and the value of this macro variable can be used so programs do not have to hard-code the value. These macro variables as well as other special macro variables available to Dispatcher Applications are discussed briefly in Appendix A.

CONCLUSION

SAS Institute is actively developing on the Application Dispatcher and many enhancements are expected. At the time this paper was written in early January 1997 the following items are planned for future releases. However, note that this list may be made obsolete by feedback from the alpha release scheduled for mid-January 1997.

1. Adding a protocol that allows a new SAS session to be launched on the Web Server in response to a browser request.

   Launching a new SAS session is most appropriate when:

   - The startup time for invoking the SAS System is not an issue.
   - The program to be run is a long running job.
   - The program has side effects which could impact other programs.

   Utilizing an already running SAS session is most appropriate when:

   - The startup time to invoke the SAS System is an issue.
   - Quick response for a short program is desirable.
   - The programs are stable and have no side effects.
   - The SAS session must be run on a machine other than the Web Server machine or a machine that the Web Server can initiate a remote process on.

2. Expanding the load balancing capabilities so that it does not just randomly select which Application Server a request should be forwarded to.

3. Other protocols for invoking the SAS System on the server being considered are:
   - Linking to SAS Version 6 Domain Name Services
   - Directly linking to a running SAS/Connect session
4. Enabling Application Dispatcher Applications to remember the state across multiple requests from the client.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the contributions of the following individuals in the development of the SAS/IntrNet Application Dispatcher and this paper.

- Renee Harper
- Aaron Hill
- Chip Kelly
- Liza Lucas
- Michael Stiefhoff
- Jack Wallace
- Bryan Wolfe

SAS and SAS/IntrNet are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

APPENDIX A: HTML TO SAS SOFTWARE COMMUNICATION

The first link in ‘browser to SAS server’ communication is the generation of HTML "name/value" pairs. Macro variables are used to make name/value pairs available to SAS programs and macro variable names are set to the value of the HTML names while the values of macro variables are set to the HTML values.

The Application Dispatcher automatically creates and assigns values to macro variables corresponding to the HTML name/value pairs. As a result, the SAS programmer need not be concerned with obtaining the values specified by the user on the HTML page. However, HTML names must correspond to valid SAS names since non-valid SAS names will be rejected by the Dispatcher.

There are several cases where it is not possible to directly map the values from an HTML page to a single macro variable. This is discussed in the next section

Multiple Values for a Single HTML Name

HTML allows multiple values to be passed using a single HTML name, however, SAS macro variables do not allow multiple values. Macro variables could contain a list of values which a SAS program could parse to recreate the distinct values. However, the Application Dispatcher uses a different method to handle multiple values.

When the Application Dispatcher receives multiple values for a single name, it appends a suffix to the original name creating multiple macro variables. Consider an example where the HTML has four checkbox fields, all called CBOX:

```
<INPUT TYPE=”checkbox” NAME=”cbox” VALUE=’first’>   First
Check Box
<INPUT TYPE=”checkbox” NAME=”cbox” VALUE=’second’>
        Second Check Box
<INPUT TYPE=”checkbox” NAME=”cbox” VALUE=’third’>
        Third Check Box
<INPUT TYPE=”checkbox” NAME=”cbox” VALUE=’fourth’>
        Fourth Check Box
```

If all four values are checked, the Application Dispatcher will define the following macro variables and values (note that you cannot assume that the values will be passed in the indicated order, e.g., the third checkbox could be the last value passed):

CBOX0:	4
CBOX:	first
CBOX1:	first
CBOX2:	second
CBOX3:	third
CBOX4:	fourth

The macro variable, CBOX0, is created to contain the total number of values received. Note that both CBOX and CBOX1 have the same value.

In a case where a checkbox is not selected, the browser may not send a blank value for the name and the SAS program must be prepared for such a scenario.

In a similar situation, if only the third checkbox is selected, the Application Dispatcher only recognizes a single value:

CBOX:	third.

The Application Dispatcher does not know that there could have been multiple values, so it does not rename them. However, since the first value is available as both CBOX and CBOX1, the program only needs to check CBOX0 to determine if multiple values were generated.

For most situations, it is recommended that the HTML program not use a name more than once. However, there are two situations where this is not possible.

First are SELECT tags (i.e., list boxes) that allow multiple selections. The selected values are passed as multiple values for a single name. Thus, if a list box (e.g., *lbox*) allows for multiple selections and only one selection is made, the value will be passed with a name of *lbox*. If three selections are made, the number of selections is passed as the value of *lbox0* and the three values (in no predictable order) are passed as *lbox1*/*lbox*, *lbox2* and *lbox3*.

Second, are TEXTAREA tags (or long text fields). Only one value is passed from the browser. However, such values can be longer than 200 characters and can contain carriage returns. In order to make these values more usable, the Dispatcher will treat these as multiple name/value pairs if the value is longer than a specified width (default width is 80) or if it contains at least one carriage return. In these cases, multiple macro variables will be created from the text value as follows:

- split at carriage return,
- or at the first blank before char #*n*,
- or at char #*n* if there is no blank

The value for *n* is specified using the hidden field *_fldwdth*. As previously discussed, the total number of values passed is stored in a macro variable with a 0 suffix.

The original HTML names must be chosen with care when new macro variables will be created as a result of multiple values. As always, the macro variable names created to contain multiple values must be eight characters or less to be valid.

For example, an HTML listbox with a name of LISTBOX limits the number of multiple values to nine because LISTBOX10 is not a valid SAS name. Keep in mind that HTML does not allow you to specify a number of selections other than 1 or MULTIPLE. In cases where names greater than 8 characters would be required, the Dispatcher will send an error page back to the user informing them of the error.

Another issue to consider when multiple value names are chosen is to avoid names that end in numbers. For example the names LBOX and LBOX1 should not be used in the same application due to the unavoidable confusion if new numbered macro variables were created as a result of multiple values.

Hidden Fields in HTML

HTML allows the creation of 'hidden fields' which are name/value pairs that do not appear on the form. Hidden fields can be used as a mechanism to pass parameters to the SAS program. For example, an HTML page could pass a list of variables to a SAS program for processing. A single SAS program could then be referenced by many HTML files. Another example might be a query application where the user specifies the query details (e.g., what variables and subset) but the data set is specified as a hidden field.

The Application Dispatcher uses another class of hidden fields. The Dispatcher recognizes the following HTML names. A leading underscore (_) is used as a convention to distinguish these hidden fields from user fields:

- _DEBUG

  This field can be used to specify a variety of debugging options. Values are:

  1 Echo all fields
  2 Print elapsed time after each run (default)
  4 Don't run, just dump information on all services
  8 Skip all execution processing
  16 Dump output in hex/ASCII
  128 Send back log file
  256 Trace socket connection attempts

  Note that these values are additive. For example, a value of _DEBUG of 130 specifies that the elapsed time and the SAS log file should both be included in the results sent back to the browser.

- _FLDWDTH

  This field specifies the maximum length for any character value. Its use is described above.

- _SERVICE

  This field defines the service to use for the request. Service specifies both the protocol (method to invoke the SAS system) and the specific implementation of the that protocol.

Other Available Fields

The following fields are created by the Dispatcher and are not specified in the HTML. The Dispatcher passes these macro variables to the Application for 'housekeeping' purposes.

- _ADMAIL

  This field contains the email address of the administrator for the selected service. This field is intended to be used if the submitted program needs to generate a *mailto* tag.

- _ADMIN

  This field contains the name of the administrator for the selected service and should be used with _admail.

- _URL

  This field contains the url address for the Broker component of the Dispatcher.

- _VERSION

  This fields contains an internal version number for the Broker component of the Dispatcher.

Environment variables can also be made available to Dispatcher Applications. Such variables are defined in the Broker CGI configuration file. The syntax for this is:

Export *environment-variable-name SAS-macrovar-name*

Some typical examples might be:

- Web server hostname
  Export SERVER_NAME _SRVNAME
- User's DNS name if known
  Export REMOTE_HOST _RMTHOST
- User's IP address
  Export REMOTE_ADDR _RMTADDR
- User name if authorized
  Export REMOTE_USER _RMTUSER
- Browser name
  Export HTTP_USER_AGENT _HTUA
- Referring page, if known
  Export HTTP_REFERER _HTREFER

Note that the SAS macro variable names listed above are only suggestions. The administrator can define any valid SAS names.

Using Environment Variables to Add Security to Dispatcher Applications

While environment variables can be used in a number of ways, a common use will be to add application level security to a Dispatcher Application. HTPASSWD, for example, which comes with standard HTTPD can be used to control access to the Broker CGI and prompt users for a userid and password. The environment variable REMOTE_USER will be set to the system userid of the person who is successfully running the Broker CGI. REMOTE_USER is passed to the Application Server as &_RMTUSER. The Dispatcher Application can then check the value of &_RMTUSER to see if it matches an observation in the security data set. Optionally, a security access level from that data set can also be used to control what features the user has access to.

Alternatively, REMOTE_ADDR could be used to determine access based on the client's IP number.