SAS/IntrNet® 9.3: htmSQL

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc 2011. *SAS/IntrNet® 9.3: htmSQL*. Cary, NC: SAS Institute Inc.

SAS/IntrNet® 9.3: htmSQL

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, July 2011

Contents

Chapter 1
Overview of htmSQL

What Is htmSQL?

Users of today's information highway demand up-to-the-minute information that is easy to access and read. htmSQL meets the challenge by integrating your SAS data with Web interface technology.

htmSQL is a CGI program that enables you to perform SQL processing from a Web page. You provide an input file containing SQL statements that are embedded in HTML; htmSQL performs updates and queries on your data source and then formats any results. Because htmSQL uses the Web for information delivery, your users can easily access your SAS data from anywhere on your network.

With htmSQL, you design the Web page--it can be as simple or as sophisticated as you want, and you can use whatever HTML elements your Web browser supports. You can display the results of any number of SQL statements on a single page and embed the results anywhere in a page. htmSQL dynamically processes the SQL in response to user requests, ensuring that the most current SAS data is processed.

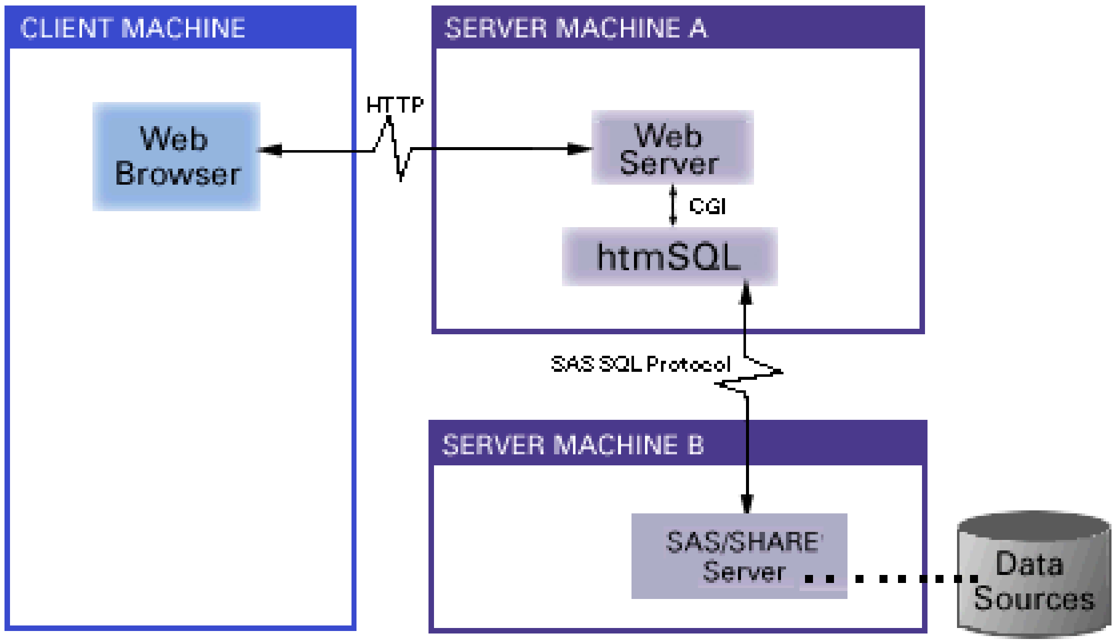
htmSQL is available for the UNIX, Windows, and z/OS platforms.

Note: z/OS is the successor to the OS/390 and MVS operating systems. SAS/IntrNet 9.1 for z/OS is supported on the MVS, OS/390, and z/OS operating systems and, throughout this document, any reference to z/OS also applies to OS/390 and MVS, unless otherwise stated.

How Does htmSQL Work?

htmSQL is a CGI program that is written in the C language and resides on your Web server. It can process special directives that are embedded in an HTML file. These directives describe one or more SQL statements and incorporate formatted results into the Web page that is created by the HTML file.

htmSQL passes your SQL to a SAS/SHARE server, performs the requested updates and queries, and retrieves the results sets. The desired page is created dynamically and returned through the Web server to the Web browser. See the data flow and required components in the following diagram:



The Web server calls htmSQL each time it receives a URL that specifies the htmSQL program name. htmSQL supports both the **`GET`** and **`POST`** CGI methods for sending form data.

htmSQL reads the input file for information that is contained within any of its directives. It processes this information and returns the results to the Web server. It returns all HTML information to the Web server just as it appears in the input file.

The following example illustrates a URL for htmSQL:

`http://yourserver/dir/executable_file/filename.hsql?query_string`

- ***`yourserver`*** is your Web server host name (and port, if required).
- ***`dir`*** is the path of the Web server CGI program directory that contains htmSQL.
- ***`executable_file`*** is the htmSQL program name. For UNIX and z/OS, the program name is htmSQL. For Windows, the program name is htmSQL.exe.
- ***`filename.hsql`*** is your htmSQL input file. Each file can contain multiple SQL statements and other input files by reference.
- ***`query_string`*** specifies values for one or more of the variables that are referenced in the input file. The variable name and value pairs are separated by ampersands (&) and are specified using the following format:

  ```
  var1=value1&var2=value2&...varN=valueN
  ```

Note: Some Web servers can be configured to recognize an input file by its file extension and to automatically call the appropriate CGI program to process the file. If your Web server can be configured this way, you can omit the path to htmSQL when you specify the URL (that is, you can omit the *dir* and *executable_file* values). Consult your Web server documentation for details about whether and how your server can be so configured.

For more information about CGI and CGI scripting, refer to the Common Gateway Interface documentation provided by W3C at **`www.w3.org/CGI`**.

Processing an htmSQL Input File

htmSQL uses a defined set of processing rules to process the information in the input file:

- htmSQL sends all text and characters that are not part of htmSQL (such as HTML tags and newline indicators) to **`stdout`** exactly as they occur in the file. If a variable reference appears in this text, htmSQL resolves the reference before sending the text to **`stdout`**.
- htmSQL collects the information that is contained in the SQL section and sends it to the SAS server as a complete SQL statement to execute. Each variable reference that is in the SQL section is resolved to the current value of the variable. htmSQL ignores newline indicators in the SQL section.
- For SQL queries, htmSQL retrieves a row of the results set and writes it to **`stdout`** according to the information that is included in the eachrow section. Variable references that correspond to column values are resolved. htmSQL repeats this step for each row in the results set.
- For SQL statements that perform update functions, htmSQL processes the success section if the return code is zero and the error section if the return code is not equal to zero.

Requirements for htmSQL

Before you can use htmSQL

- you must know your SAS data, be able to write valid SQL statements, and understand HTML tags
- your Web server must run under UNIX, Windows, or z/OS
- your network must include a version of SAS software with licenses for
  - SAS/SHARE software
  - SAS/IntrNet software.

If your data is in an external DBMS, you must also have the SAS/ACCESS software for that DBMS.

Chapter 2

Programming with htmSQL

htmSQL Input Files

Overview of htmSQL Input Files

To create your Web page, you must first provide htmSQL with an input file. This file contains the HTML and SQL that you want processed.

When someone wants to access your Web page, they pass a URL that contains the name and location of the input file to htmSQL. The URL can also contain information used to resolve variable references that are in the input file.

General Structure of an htmSQL Input File

An input file can contain zero or more of the following elements:

- variable references. The references can be to columns selected in queries, to variables specified in the URL, or to variables that htmSQL automatically defines and supplies values for. The references are replaced by the current value of the variable.
- complete query sections, delimited by the {query} and {/query} directive pair. Each query section contains at least one SQL/eachrow section pair and can contain multiple pairs. For each SQL section, you can include a norows section.
  - The SQL section is delimited by the {sql} and {/sql} directive pair and specifies how to construct a query that htmSQL sends to a SAS/SHARE server.
  - The eachrow section is delimited by the {eachrow} and {/eachrow} directive pair and describes how to display the results set.
  - The norows section is delimited by the {norows} and {/norows} directive pair and contains the steps to take when the previous SQL section does not return any rows.
- complete update sections, delimited by the {update} and {/update} directive pair. Each update section contains one or more SQL sections. For each SQL section, you can include success and error sections.
  - The SQL section is delimited by the {sql} and {/sql} directive pair and specifies how to construct an SQL statement that htmSQL sends to a SAS/SHARE server.
  - The success section is delimited by the {success} and {/success} directive pair and contains the steps to take when the SQL is processed with a return code of zero. The success section can also contain a norows section.
  - The error section is delimited by the {error} and {/error} directive pair and contains the steps to take when the SQL is processed with a nonzero return code.
- {library} directive. The {library} directive can be included in both the query and update sections and defines a high-level qualifier that you use in the names of tables and views in your SQL queries and statements.
- {label} directive. The {label} directive enables you to display the label for a column in a results set.
- included files. Use the {include} directive to specify another file for htmSQL to process before continuing with the current file.

- htmSQL comments. All text contained between {* and a closing brace } is considered an htmSQL comment and is not written to **`stdout`**.

Everything else in the input file is written, as is, to **`stdout`**. This includes text, HTML, and newline characters.

Specifying Values for User IDs and Passwords

You can specify values for the userid= and password= parameters in any of three ways:

1. Hardcode the values.
2. Collect the password or user ID through an HTML form. The value is passed to htmSQL along with the CGI request for your input file. You specify the password or user ID by using a variable reference as the value of this parameter. The following example illustrates the use of variable references as the values of the userid= and password= parameters.

   ```
   {update datasrc="employee1" userid="{&userid}" password="{&password}"}
   ```

3. In the htmSQL input file, perform a query that retrieves the user ID or password from a user ID/password data set. In subsequent query and update sections, you can use a variable reference to refer to the value in either a userid= or password= parameter. (See previous example).

The first of these three methods is not secure; the second method is secure only if you are using secure sockets or secure HTTP; the third method is fairly secure and, if your Web server supports client authentication, can be used to supply a different user ID and password for each user.

Automatic Variables

htmSQL automatically defines a number of variables that contain htmSQL processing information. The following sections list the variables and examples of variable values and usage. htmSQL also provides a sample input file named autovars.hsql that lists values for many of the automatic variables.

Date and Time Variables

The format of the date and time variable information is determined by the LC_TIME and LANG environment variables and according to the NLS installation on your Web server machine.

Variable Name	Description	Example Value
sys.ampm	the time of day before or after noon (AM or PM)	PM
sys.date	the current date in *ddMmmyy* or *ddMmmyyyy* format (depending on the value of the YEARDIGITS run-time configuration option)	04Apr2000

Variable Name	Description	Example Value
sys.datetime	the current date in *ddMmmyy hh:mm:ss TZ* or *ddMmmyyyy hh:mm:ss TZ* format (depending on the value of the YEARDIGITS run-time configuration option)	04Apr2000 16:30:52 EDT *
sys.fulldate	the current date, including the weekday and date	Friday, April 04, 1997
sys.fulldatetime	the current date, including the weekday, date, time, and time zone	Friday, April 04, 1997 04:30:52 PM EDT *
sys.month	the month of the year	September
sys.month3	the month of the year (abbreviated)	Sep
sys.monthday	the day of the month	05
sys.monthnum	the month of the year expressed numerically	09
sys.seconds	the second of the minute	47
sys.time	the time of day using a 12-hour clock	3:36
sys.time24	the time of day using a 24-hour clock	15:36
sys.tz	the time zone	EDT *
sys.weekday	the day of the week	Saturday
sys.weekday3	the day of the week (abbreviated)	Sat
sys.year	the calendar year	1997
sys.year2	the last two digits of the calendar year	97

*For the Windows platforms, the value for the time zone is not abbreviated (for example, Eastern Daylight Time).

SQL-Related Variables

The following list contains variables for information that is related to an SQL statement:

sys.colname

The column names in a results set. Use the following syntax to reference a single specific value, a range of values, or all values:

`&sys.colname[`*`n`*`]`	references the ***n***th column name in the results set where ***n*****`>=1`**.
`&sys.colname[`*`m..n`*`]`	references all column names from the ***m***th value to the ***n***th value where ***m*****`<=1<`*****n*** and the results set contains two or more columns.
`&sys.colname[*]`	references all the column names in the results set.

For example, for the following SQL statement, the selected columns are **`first`**, **`middle`**, and **`last`**:

```
{sql}select * from employee.names{/sql}
```

Following are some example references and their resolved values:

```
{&sys.colname[1]} -----> first
{&sys.colname[2..3]} -----> middle last
{&sys.colname[*]} -----> first middle last
```

sys.qrow

The number of the current row in the results set. The following example shows how to include the sys.qrow variable in an eachrow section:

```
{eachrow}
{&sys.qrow} {&lastname} {&firstname} {&middleinit}
{/eachrow}
```

Each row that is output contains the number of the row and the three values that correspond to the other three variable references. The output might look something like the following:

```
1 Doe John D.
2 Doe Jane R.
3 Doe Sam E.
```

sys.query

The last SQL query that is processed. All the characters in the query are included except for

- newline characters
- leading blank spaces that are before the SELECT keyword.

sys.updrows
sys.updcount

The number of rows that are inserted, deleted, or updated by the last INSERT, DELETE, or UPDATE statement.

sys.updrc

The return code from the last INSERT, DELETE, or UPDATE statement.

Miscellaneous Variables

Variable Name	Description	Example Value
sys.dirurl	the URL directory path (with no filename) for the top-level input file	/myfiles/
sys.filetime	the date and time of the last modification of the current input file	Monday, May 05, 1997 02:05:45 PM EDT *
sys.fileurl	the URL pathname for the top-level input file	/myfiles/myinput.hsql
sys.url	the URL for invoking htmSQL (not including the pathname of the input file)	/cgi-bin/htmSQL
sys.version	the version number for htmSQL	2.0

*For the Windows platforms, the value for the time zone is not abbreviated (for example, Eastern Daylight Time).

Formats for Variable Values and Labels

Formats in htmSQL

htmSQL supports formats in two different ways:

- by providing the format= parameter, which enables you to explicitly specify a format with variable references and directives.
- by using the format that is associated with the data set column.

Specified Formats

The following values can be used for the format= parameter of htmSQL variable references and directives:

comma

formats numeric values using commas to separate every three digits. When you specify a format of *w.d* with the comma format, you must specify either a 0 or 2 for the value of *d* (the number of decimal places to the right of the decimal character). If you specify any other value for *d*, then *d* defaults to a value of 2. When *d* is equal to 2, htmSQL outputs a decimal point followed by two fraction digits.

The following is an example of how to use this format:

```
{&abc format=(comma, 10.2)}
```

`commax`

formats numeric values using periods to separate every three digits. When you specify a format of *w.d* with the commax format, you must specify either a 0 or 2 for the value of *d* (the number of decimal places to the right of the decimal character). If you specify any other value for *d*, then *d* defaults to a value of 2. When *d* is equal to 2, the htmSQL outputs a comma followed by two fraction digits.

The following is an example of how to use this format:

```
{&abcx format=(commax, 10.2)}
```

`dollar`

formats numeric values using a leading dollar sign ($) or currency symbol and using commas to separate every three digits. If you specify a format of *w.d* with the dollar format, and you specify a nonzero value for *d* (the number of decimal places to the right of the decimal character), then htmSQL outputs a decimal point followed by two fraction digits.

The following are examples of how to use this format:

```
{&abc format=(dollar, 15.2)}
{&abc format=dollar}
```

`dollarx`

formats numeric values using a leading dollar sign ($) or currency symbol and using periods to separate every three digits. If you specify a format of *w.d* with the dollarx format, and you specify a nonzero value for *d* (the number of decimal places to the right of the decimal character), then htmSQL outputs a comma followed by two fraction digits.

The following are examples of how to use this format:

```
{&abcx format=(dollarx, 15.2)}
{&abc format=dollarx}
```

`exp`

formats numeric values in scientific notation. For example, if the numeric variable abc has a value of -13454, then htmSQL formats the value as -1.345e+04 if you use the following variable reference:

```
{&abc format=(exp, 10.3)}
```

.

`hex`

formats numeric values in hexadecimal. If you use the *w.d* format with the hex format, htmSQL ignores the *d* value.

`htmlescape`

replaces each of the following special characters with the corresponding character entity reference whenever htmSQL encounters the special character in a variable value:

Special Character	Character Entity Reference
left angle bracket (<)	**<**
right angle bracket (>)	**>**
ampersand (&)	**&**

Special Character	Character Entity Reference
double quotation mark (")	"

Use this option if the variable's value includes special characters that should be rendered as is when the output Web page appears.

`left`
prints the value of a numeric variable with no leading blanks.

`notrim`
retains the trailing blank spaces in the variable value. Retaining trailing blanks enables you to line up values on the Web page more easily. If you do not specify **`format=notrim`**, htmSQL discards trailing blanks in the variable value when it resolves the reference.

Note: Because most Web browsers collapse consecutive blank spaces, the notrim format is most effective when it is used with the **`<PRE>`** HTML element.

`right`
right aligns character variable values and pads enough blank spaces on the left to fill up the field width. For example, if the variable linename contains the string "line01", then**`:{&linename format=10}:`** appears as

```
:line01 :
```

and **`:{&linename format=(right, 10)}:`** appears as

```
: line01:
```

Note: Because most Web browsers collapse consecutive blank spaces, the right format is most effective when it is used with the **`<PRE>`** HTML element.

`urlencode`
URL-encodes the value of the variable. When the values are URL-encoded, the spaces are replaced with plus signs (+). All other nonalphanumeric characters are replaced with escape sequences (%*xx*), where *xx* is the hexadecimal representation of the ASCII code point.

Use this option when you include variable references in the values for the **`ACTION`** or **`HREF`** attributes of HTML elements.

wd.
w specifies the width of the print field. The allowed values for *w* are integers from 1 to 32767. *d* is the precision specifier (specifies the number of decimal places to the right of the decimal character). The maximum value for *d* depends on the exponent of the largest numeric value that an operating system can store in a double. If you do not specify a value for *d*, the default value is 0.

You can specify a value either for *w*, for *d*, or for both. If you specify *d* by itself, you must precede it with a period (.). The value for *d* is useful only for numeric values and is ignored for variables containing character and integer values. Note that the following format values are all equivalent: **`format=8.`**, **`format=8`**, and **`format=8.0`**.

Note: The htmSQL implementation of field widths (*w*) for numeric values differs from the SAS implementation. For SAS software, *w* is generally an absolute specification for the field width, and SAS software changes the formatting of the number to accommodate the width (by doing such things as reducing precision and changing formats). For numeric values in htmSQL, the *w* width specification is a minimum and is adjusted upward, if necessary, to accommodate the numeric value and the precision specifier (*d*).

The htmSQL implementation of field widths (*w*) for character data is the same as the SAS implementation. Both implementations indicate the exact number of characters to format, either truncating or blank-padding as necessary.

`zero`
: pads enough zeros on the left of numeric values to fill up the field width. Without the zero format, numeric values are left-padded with blanks. The zero format is ignored when either the left or exp formats are used.

Associated Formats

- When you refer to a column in a query results set, if you do not specify the format= parameter but the column has one of the following formats associated with it, then htmSQL uses the associated format to resolve the reference:
  - *w.d*
  - COMMA*w.d*
  - COMMAX*w.d*
  - DOLLAR*w.d*
  - DOLLARX*w.d*
  - E*w*.
  - Z*w.d*
- When you refer to a column in a query results set, if the column has a date, time, or datetime format associated with it, the following formats are used:

  For date values:
  Regardless of what date format your column has, htmSQL always outputs date values as either *ddMmmyy* or *ddMmmyyyy*, where
  - *dd* is the day of the month
  - *Mmm* is the first three letters of the month of the year
  - *yy* and *yyyy* are the last two and four digits of the year, respectively (depending on the value of the YEARDIGITS run-time configuration option).

  For time values:
  Regardless of what time format your column has, htmSQL always outputs time values as *hh:mm:ss*, where
  - *hh* is the hour of the day using a 24-hour clock
  - *mm* is the minute of the hour
  - *ss* is the second of the minute. Note that htmSQL does not handle fractions of seconds.

  For datetime values:
  Regardless of what datetime format your column has, htmSQL always outputs datetime values as *ddMmmyy hh:mm:ss* or *ddMmmyyyy hh:mm:ss*, where
  - *dd* is the day of the month
  - *Mmm* is the first three letters of the month of the year

- *yy* and *yyyy* are the last two and four digits of the year, respectively (depending on the value of the YEARDIGITS run-time configuration option)
- there are two spaces separating the date value and the time value
- *hh* is the hour of the day using a 24-hour clock
- *mm* is the minute of the hour
- *ss* is the second of the minute. Note that htmSQL does not handle fractions of seconds.

Note: If you want to use other date, time, or datetime formats, you can use the PUT() function in your SQL query to change the format.

Invoking htmSQL

Creating an HTML Form

From your Web browser, you can use an HTML form to invoke htmSQL. The HTML form must use the URL for your Web page as the value of the **ACTION** attribute of the HTML **FORM** element. You can also use the optional **METHOD** attribute to specify the CGI method to use for sending form data:

```
<form action="http://yourserver/dir/executable_file/input-file" [method=get|post]>
```

- ***yourserver*** is your Web server host name (and port, if required).
- ***dir*** is the path of the Web server CGI program directory that contains htmSQL.
- ***executable_file*** is the htmSQL program name. For UNIX and z/OS, the program name is htmSQL. For Windows, the program name is htmSQL.exe.
- ***input-file*** is your input file as a relative pathname under the Web server's root directory (which can include a Web server alias).
- **GET** and **POST** are the two CGI methods for sending form data.

  method=get
  : Your Web browser sends the form data to the Web server as part of the URL. The Web server passes the form data to htmSQL through the environment variable QUERY_STRING.

  method=post
  : Your Web browser sends the form data to the Web server as part of the body of the HTTP request. htmSQL reads the form data from **stdin**.

On the form, use HTML **INPUT** elements to collect variable values. For the **NAME** attribute of the **INPUT** element, use the same variable names that you use for your variable references. When the form is submitted, the Web browser automatically generates the following query string from the form input and appends it to the URL that is specified by the **ACTION** attribute (for the **GET** method) or sends it in the body of the HTTP request (for the **POST** method):

```
var1=value1&var2=value2&...varN=valueN
```

Note: Some Web servers can be configured to recognize an input file by its file extension and to automatically call the appropriate CGI program to process the file. If your Web server can be configured this way, you can omit the path to htmSQL when you specify the URL (that is, you can omit the *dir* and *executable_file* values).

Consult your Web server documentation for details about whether and how your server can be so configured.

Specifying a URL for Your Web Page

From your Web browser, you can specify a URL to invoke htmSQL. Specify the URL either on an existing Web page or on your Web browser command line. A URL for htmSQL must include the pathname for an input file and must be of the form:

```
http://yourserver/dir/executable_file/input-file[?query-string]
```

- ***`yourserver`*** is your Web server host name (and port, if required).
- ***`dir`*** is the path of the Web server CGI program directory that contains htmSQL.
- ***`executable_file`*** is the htmSQL program name. For UNIX and z/OS, the program name is htmSQL. For Windows, the program name is htmSQL.exe.
- ***`input-file`*** is your input file as a relative pathname under the Web server's root directory (which can include a Web server alias).
- ***`query-string`*** specifies values for one or more of the variables that are referenced in the input file. This parameter is optional. The variable name and value pairs are separated by ampersands (&) and are specified using the following format:

  ```
  var1=value1&var2=value2&...varN=valueN
  ```

  Be sure to properly encode any nonalphabetic characters that are in the query string: spaces become plus signs (+) and other characters are replaced by a percent sign (%) and the two-digit ASCII representation.

The following example shows a URL that invokes htmSQL: **`http://support.sas.com/cgi_bin/htmSQL/empdata.hsql?first=fname&last=lname&middle=mi`**

Note: Some Web servers can be configured to recognize an input file by its file extension and to automatically call the appropriate CGI program to process the file. If your Web server can be configured this way, you can omit the path to htmSQL when you specify the URL (that is, you can omit the *dir* and *executable_file* values). Consult your Web server documentation for details about whether and how your server can be so configured.

From the Command-Line Prompt

The output that htmSQL generates is sent to **`stdout`** (usually your terminal display). You can capture the generated output by redirecting **`stdout`** to a file. Use this method of invocation to test an htmSQL input file or to produce a static page that contains SAS data.

The following is the syntax for the htmSQL command (parameters that are within square brackets (**`[]`**) are optional):

```
htmSQL input-file ["query-string"] [-rc config-file] [-dsf datasrc-file]
```

- *input-file* specifies the pathname for your input file.
- *query_string* specifies values for one or more of the variables that are referenced in the input file. The variable name and value pairs are separated by ampersands (&) and are specified using the following format:

  ```
  var1=value1&var2=value2&...varN=valueN
  ```

You do not need to encode nonalphabetic characters that are in the query string.

- **`-rc`** ***`config-file`*** specifies the pathname of the configuration file to use. You can name the file whatever you want and locate the file in whichever path you choose. htmSQL does not run if the specified file cannot be opened.

  This parameter is optional. For more information about running htmSQL without explicitly specifying a configuration pathname, see "Specifying and Naming the Configuration File" on page 47.

- **`-dsf`** ***`datasrc-file`*** specifies the pathname of the data source definition file. If you specify a simple filename, htmSQL looks in the current directory for the file. This option overrides any data source definition file that is specified in the configuration file. The file does not have to be named htmSQL.datasrc, htmSQL.dsf, or HTMSQL DATASRC.

- the **`-rc`** and **`-dsf`** parameters can be placed anywhere after the htmSQL command name.

Configuring Your Web Server to Recognize htmSQL Input Files

Overview of Configuring Your Web Server for htmSQL

When you configure your Web server to recognize htmSQL input files, you no longer have to specify the pathname of the htmSQL executable in the URL that you use for invoking htmSQL. For example, instead of the URL **`http://support.sas.com/cgi-bin/htmSQL/myinput/myfile.hsql`**, you can use **`http://support.sas.com/myinput/myfile.hsql`** and the Web server automatically invokes htmSQL to process the input file.

The following sections provide some instructions on how to configure various Web servers to recognize htmSQL input files.

Note: This section discusses only servers that we have tested. It is not an exhaustive discussion of the topic. If you encounter problems when configuring your Web server or for more information about configuring your Web server, consult your Web server documentation.

Apache

To configure the Apache Web server (version 1.1 or later), add the following lines to the srm.conf file. The lines that begin with the pound sign (#) are comment lines that are already in the srm.conf file that is included in the Apache download package.

```
# AddHandler allows you to map certain file extensions to "handlers",
# actions unrelated to filetype. These can be either built into the server
# or added with the Action command (see below)
# Format: AddHandler action-name ext1

AddHandler htmSQL .hsql

# Action lets you define media types that execute a script whenever
# a matching file is called. This eliminates the need for repeated URL
# pathnames for oft-used CGI file processors.
# Format: Action media/type /cgi-script/location
# Format: Action handler-name /cgi-script/location

Action htmSQL /cgi-bin/htmSQL
```

Peer Web Services and the Internet Information Server

Internet Information Server (IIS) 7.0

To set up htmSQL on IIS 7.0, complete the following steps:

1. Open the applicationHost.config file located in **`C:\Windows \System32\inetsrv\config\`**.
2. Locate and then edit the following line so that the overrideModeDefault variable is set to Allow:

   ```
   <section name="handlers" overrideModeDefault="Allow" />
   ```

3. Save the file.
4. Open the web.config file. The default location for the file is in the same folder where the CGI executables are located. However, if there is no web.config file in this folder, then you need to create one.
5. Make the following edit to the file:

   ```
   <configuration>
      <system.webServer>
         <handlers>
            <add name="ScriptMap1" path="broker.exe" verb="*" modules="CgiModule"
               scriptProcessor="c:\inetpub\scripts\broker.exe" resourceType="File"
               allowPathInfo="true" />
            <add name="ScriptMap2" path="htmSQL.exe" verb="*" modules="CgiModule"
               scriptProcessor="c:\inetpub\scripts\htmSQL.exe" resourceType="File"
               allowPathInfo="true" />
         </handlers>
      </system.webServer>
   </configuration>
   ```

6. Save the file.
7. Restart the IIS Web server.

See **`http://support.sas.com/techsup/technote/ts794.pdf`** for more information about configuring IIS 7.0 for SAS/IntrNet software.

Internet Information Server (IIS) 6.0

To set up htmSQL on IIS 6.0, complete the following steps within the IIS Administrator:

1. In IIS, right-click the individual Web site or the **`Web Sites`** folder, and then click **Properties**.
2. On the **Home Directory** tab, click **Configuration**.
3. Under **Application Configuration**, click **Add**, and then click the **Mappings** tab.
4. With the Add/Edit Application Extension Mapping dialog box open, click **Browse** to select the htmsql.exe file from the local path on the Web server.

Note: You must type the path to a valid file in the Executable text box or the **OK** button remains unavailable. The easiest way to ensure that you enter a valid path is to select the file by using the **Browse** button.

5. After the path appears in the Executable text box, click in the Executable text box to initialize the path.
6. Click in the Extension space, and then enter **`.hsql`** as the filename extension.

   Note: You must enter the period (.) in front of the extension in the Extension text box, or the **OK** button remains unavailable.
7. Click **OK**.
8. Configure IIS to allow the hsql MIME type. This can be done by completing the following steps:
   a. In the left window of the IIS Administrator, right-click your machine name and select **Properties**.
   b. Add a MIME type of **`.hsql, application/octet-stream`**.
9. Right-click the directory where the htmsql.exe file is located (for example, cgi-bin) and allow scripts and executables. If your IIS6 Web Server does not already have a **`scripts`** directory or a **`cgi-bin`** directory, you need to create one. After you create this directory, on the **Virtual Directory** tab, specify **`Scripts and Executables`** for the value of Execution Permissions.

   Some versions of IIS might have a check box called **Execute (such as ISAPI applications or CGI)**. If you have this selection, then the box should be checked.
10. Right-click the directory where the Web site is located (for example, MyWeb) and allow scripts and executables.
11. Enable Web Service Extensions for the htmsql.exe file. This can be done by completing the following steps:
    a. In the left window of the IIS Administrator, select your machine name. Then select the **`Web Service Extensions`** folder. The **`Web Service Extensions`** folder is under the folder that specifies the name of your machine. For example, if your machine name is SUSAN2, then there should be three folders under SUSAN2 (called **`Application Pools`**, **`Web Sites`**, and **`Web Service Extensions`**).
    b. Browse to the EXE for htmSQL and click **Allow** (to permit the htmsql.exe file to execute). Otherwise, the status is DENY.
12. Start your SAS/SHARE server.

For more information about running CGI tools on the Microsoft IIS 6.0 Web server, see SAS Note #23931 at **`support.sas.com/kb`**.

Internet Information Server (IIS) 5.0 and Earlier and Peer Web Services

To configure Microsoft Peer Web Services for Windows NT workstations or the Microsoft IIS for Windows NT servers, you must add two string values for the following key in the Windows registry on your Web server machine:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\ScriptMap]
```

Use the Registry Editor (**`regedit`**) to add the following string values:

```
".hsql"="c:\directory_for_htmSQL_executable\htmSQL.exe"
".hsq"="c:\directory_for_htmSQL_executable\htmSQL.exe"
```

Ensure that you change ***`directory_for_htmSQL_executable`*** to the directory in which you installed htmSQL.

Note: You must also turn on the execute bit for the directory that contains the .hsq or .hsql files.

Creating an htmSQL Web Page

The following steps guide you through the process of creating and displaying a Web page with htmSQL:

1. Construct your SQL statement. Decide whether you want to vary any part of it (that is, use variable references). For example

   ```
   select name, address, city, zip
      from def.mailresp
      where incr="{&incr}" and ager="{&ager}"
      order by city, zip
   ```

   where incr and ager are variables that you define on the URL when you invoke htmSQL.

2. Decide how you want to organize the results--perhaps in a table or in a preformatted section.

3. Decide how you want your Web page to look and where you want to place the various elements of your page (be sure to include titles, headings, images, and any query results).

4. If necessary, ask your Webmaster to add data sources to your data source definition file. To define a data source , the Webmaster uses the dsdef program that is provided with htmSQL.

5. If you want to update data, use an update section in your input file. In the update section, include

   - an SQL section that contains the SQL that you wrote in step 1. The SQL keywords that are allowed in an update section are ALTER, CREATE, DELETE, DROP, INSERT, and UPDATE.
   - a success section that contains the steps to take if the SQL is processed with a return code of zero.
   - an error section that contains the steps to take if the SQL is processed with a nonzero return code.

6. To perform a query and display data from the results set, use a query section at the location where you want your query results to appear. In the query section, include

   - an SQL section that contains the SQL that you wrote in step 1. Use variable references for the parts that you want to vary, such as column values in a WHERE clause and column names in an ORDER BY clause.
   - an eachrow section that contains the text and HTML that is to accompany each row of the results set (depending on what you decided in step 2). Use variable references in the appropriate places for the columns that you want to display.
     - If you want the results shown as preformatted text, the eachrow section should be contained within an HTML **`PRE`** element.

- If you want the results shown in an HTML table, the eachrow section should be contained within an HTML **`TABLE`** element. Between **`{eachrow}`** and **`{/eachrow}`**, the row details should be contained within an HTML **`TR`** element.

Continuing with the example:

```
{query datasrc="demos"}

{sql}
   select name, address, city, zip
      from def.mailresp
      where incr="{&incr}" and ager="{&ager}"
      order by city, zip
{/sql}

{eachrow}
{&name}<br>
{&address}<br>
{&city}, TX {&zip}<p>
{/eachrow}

{/query}
```

7. You can test your file by invoking htmSQL from a command-line prompt and passing the file and any required variables on the command line:

   ```
   htmSQL mailres3.hsql "incr=20K to 39K&ager=30 to 39" > file1.out.html
   ```

   where mailres3.hsql is the name of an input file.

   The string that is enclosed within the quotation marks (") specifies values for the variables that are used by the input file. See "Invoking htmSQL" on page 14 for more information about htmSQL command-line options.

   You can then display file1.out.html in a Web browser to ensure that what htmSQL produced is what you want.

8. After you test your input file, you can link to your new Web page

   - from an HTML form, or
   - by specifying the URL for your Web page either on an existing Web page or on your Web browser command line.

You can see the complete example input file that is described on this page by visiting the following URL: **`www2.sas.com/htmSQL/mailres3.txt`**.

Tips and Techniques for Using htmSQL

This section includes tips and hints that other users have found useful.

Checking the Version Number

To find out which version of htmSQL you are running, invoke htmSQL without specifying an input file. htmSQL displays the version number at the end of the usage page that it displays.

Comparing Floating-Point Values

Numeric values in htmSQL are always represented as floating-point values. When comparing numeric values in htmSQL, you must ensure that the values that you compare can be represented exactly in floating-point notation. That is, the value that you specify must be able to remain the same after going through conversions from binary floating-point to string and vice-versa (these conversions are necessary for SQL functions). Some floating-point values, namely fractions, cannot be exactly reproduced following conversions.

Be aware that when you use the SAS TIME() and DATETIME() functions to generate your time and date values, your generated values are likely to contain fractions because SAS dates and times are stored as floating-point values.

If you want to use a WHERE clause that compares fractional values or htmSQL variable references with numeric columns that contain fractional values (for example, **`WHERE x=1.22`** or **`WHERE x={&myfloat}`**), you can apply any of the following strategies:

- Truncate date and time values to integer values before you store them. Note that htmSQL does not output the fractional part of dates and times.
- Use the PUT() function to change the floating-point value into a different format before you store it.

Unique Namespaces

For any given scope, there is one namespace for variable names (that is, there is only one list in which variable names are stored). Each input file has a separate scope. Include files have separate scopes so they also have separate namespaces.

If, in a query, you select a variable with the same name as a variable that was passed in on the query string or that was selected in a previous query, then the original value of that variable is replaced by your newly selected value.

It is also possible to overwrite the value of an automatic variable if you pass a value for an automatic variable in the query string of a URL.

Using Existing Connections

htmSQL recognizes when a **`{query}`** directive uses the same parameters as the previous **`{query}`** directive. Instead of making a new connection, htmSQL uses the existing connection, which saves execution time.

Using SAS Formats and the PUT() Function

If you want to format data using a SAS format that is not supported by htmSQL, you can use the PUT() function in your SQL statement to specify that format. For example, if you want to use the SAS *mmddyy*8. format to format a date, write an SQL statement like the following SELECT statement:

```
select put(datevar,mmddyy8.) as datevar ...
```

The value that is returned in the datevar variable is a character value in the *mmddyy*8. format. You can then provide more formatting for the variable by using the format option. For example, the following applies a format that htmSQL supports (left) onto the date variable:

```
{&datevar format=left}
```

Note that if you use the following select statement, htmSQL does not format the datevar variable using the *mmddyy*8. format:

```
select datevar format=mmddyy8...
```

Instead a numeric value is returned and formatted using the default date format.

Using User-defined Formats

If you want to use a format that you defined, then you must provide a libref definition in the SAS program that starts your SAS server. For example:

```
libname myfmts '/u/joeuser/formats';
options fmtsearch=myfmts;
.
.
.
proc serverid=shr1...;
```

After you define your formats in this way, you can use the PUT() function to specify the format in your SQL code.

Getting Started Exercises

About the Exercises

The following exercises illustrate the steps that your organization must follow to install htmSQL and run htmSQL applications. The first exercise requires no data source definition file and does not require that you modify the TCP/IP SERVICES file on your Web server machine.

Note:

- Our exercises assign each step to either the Webmaster or the programmer. However, depending on the way your organization is set up, you might have other people performing the tasks.
- For more htmSQL samples, see your htmSQL administrator for the URL of the samples that are installed with the htmSQL software. The default URL is **`http://`*`yourserver`*`/sasweb/IntrNet9/htmSQL/samples.html`** where ***`yourserver`*** is your Web server host name.

Exercise 1

Basic Example

In the samples directory, we provide an example htmSQL input file (retail1.hsql) that you can modify and use. This exercise shows how to customize and run the example file.

Note: You do not need to save the example htmSQL input file that is shown here. All of the example files in these exercises are supplied in the samples directory.

1. **Webmaster:** Install htmSQL by following the instructions in the README file that is in the download package.

2. **Webmaster:** Define your SAS/SHARE server in the TCP/IP SERVICES file on the SAS/SHARE server machine.

3. **Webmaster:** Ensure that a SAS/SHARE server is running. The following SAS commands can be used to start a SAS/SHARE server. Our exercise uses an example value of **`shr10`** for the server ID; replace **`shr10`** with the service name that you defined in step 2.

```
options comamid=tcp;
proc server id=shr10;
run;
```

4. **Programmer:** Edit the **`{query}`** directive in the example input file that is in the samples directory:

   - Change **`sampnode.pc.sas.com`** to the IP name of the SAS/SHARE server machine.

   - Change **`5000`** to the port number that is assigned to the service name that you defined in step 2.

   In our exercise, the SAS/SHARE server, **`shr10`**, is running on **`sampnode.pc.sas.com`**. Service **`shr10`** is assigned to port 5000.

5. **Programmer:** Invoke htmSQL to process the example input file. To run our example from the command line, change to the samples directory and issue the following command:

```
htmSQL retail1.hsql param=1992
```

retail1.hsql Sample Input File

```
{*-----------------------------------------------------------------*}
{* Note: You do not need to save this file from your Web browser. *}
{* This file is available in the htmSQL "samples" subdirectory. *}
{* *}
{* This input file produces a Web page that lists sales information *}
{* that is stored in the 'retail' SAS data set of the SASHELP library.*}
{*-----------------------------------------------------------------*}

<HEAD><TITLE>htmSQL: Retail Data</TITLE></HEAD>
<BODY BGCOLOR="FFFFFF">

{*-----------------------------------------------------------------*}
{* The following section queries the SAS server that is identified by *}
{* the server= parameter and dynamically generates the sales *}
{* information. *}
{*-----------------------------------------------------------------*}

{query server="sampnode.pc.sas.com:5000"}
{sql}
select sales,date,year,month,day
from sashelp.retail where
year = {&PARAM}
{/sql}

{*-----------------------------------------------------------------*}
{* We use a table to show the sales information. *}
{*-----------------------------------------------------------------*}
```

```
<TABLE BORDER=1 ALIGN=LEFT>
<TR>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Sales</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Date</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Year</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Month</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Day</TH>
</TR>

{*-------------------------------------------------------------------*}
{* Each row of our results set corresponds to a row in the table *}
{*-------------------------------------------------------------------*}

{eachrow}
<TR>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&sales}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&date}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&year}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&month}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&day}</TD>
</TR>
{/eachrow}
</TABLE>

{*-------------------------------------------------------------------*}
{* End the query section. *}
{*-------------------------------------------------------------------*}

{/query}
</BODY>
```

Exercise 2

Advanced Example

In the samples directory, we also provide two files that demonstrate how to use a data source definition with htmSQL. This example htmSQL input file (retail2.hsql) and example data source definition file (retail.datasrc for UNIX and z/OS and retail.dsf for Windows) perform the same task as the input file in exercise 1 but give you the ability to centralize the definition of your data.

Note: You do not need to save the example htmSQL input file and data source definition file that are shown here. All of the example files in these exercises are supplied in the samples directory.

1. **Webmaster:** Install htmSQL by following the instructions in the README file that is in the download package.
2. **Webmaster:** Define your SAS/SHARE server in the TCP/IP SERVICES file on both the SAS/SHARE server machine and on the Web server machine.
3. **Webmaster:** Ensure that a SAS/SHARE server is running. The following SAS commands can be used to start a SAS/SHARE server. Our exercise uses an example value of **`shr10`** for the server ID; replace **`shr10`** with the service name that you defined in step 2.

```
options comamid=tcp;
proc server id=shr10;
run;
```

4. **Webmaster:** Modify the example data source definition file that is in the samples directory (you can use a text editor to make these changes):
   - Change **`sampnode.pc.sas.com`** to the IP name of the SAS/SHARE server machine.
   - Change all occurrences of **`sampnode.shr10`** to *node.service* where
     - *node* is the node name of the server machine
     - *service* is the service name that is specified in the TCP/IP SERVICES file in step 2 (which is also the value for the ID= option of the PROC SERVER statement in step 3).

   In our exercise, the SAS/SHARE server, **`shr10`**, is running on sampnode.pc.sas.com.

5. **Programmer:** Invoke htmSQL to process the example input file. To run our example from the command line, change to the samples directory and issue one of the following commands:

   For UNIX and z/OS:

   ```
   htmSQL retail2.hsql param=1992 -dsf retail.datasrc
   ```

   For Windows:

   ```
   htmSQL retail2.hsql param=1992 -dsf retail.dsf
   ```

Sample Data Source File

```
{*-------------------------------------------------------------------*}
{* Note: You do not need to save this file from your Web browser. *}
{* This file is available in the htmSQL "samples" subdirectory. *}
{* *}
{* This input file produces a Web page that lists sales information *}
{* that is stored in the 'retail' SAS data set of the SASHELP library.*}
{*-------------------------------------------------------------------*}

<HEAD><TITLE>htmSQL: Retail Data</TITLE></HEAD>
<BODY BGCOLOR="FFFFFF">

{*-------------------------------------------------------------------*}
{* The following section queries the SAS server that is identified by *}
{* the data source and dynamically generates the sales information. *}
{*-------------------------------------------------------------------*}

{query datasrc="retail"}
{sql}
select sales,date,year,month,day
from sashelp.retail where
year = {&PARAM}
{/sql}

{*-------------------------------------------------------------------*}
{* We use a table to show the sales information. *}
{*-------------------------------------------------------------------*}
```

```
<TABLE BORDER=1 ALIGN=LEFT>
<TR>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Sales</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Date</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Year</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Month</TH>
<TH ALIGN=CENTER VALIGN=MIDDLE BGCOLOR="FF0000" NOWRAP>Day</TH>
</TR>

{*-------------------------------------------------------------------*}
{* Each row of our results set corresponds to a row in the table *}
{*-------------------------------------------------------------------*}

{eachrow}
<TR>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&sales}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&date}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&year}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&month}</TD>
<TD ALIGN=LEFT VALIGN=MIDDLE NOWRAP>{&day}</TD>
</TR>
{/eachrow}
</TABLE>

{*-------------------------------------------------------------------*}
{* End the query section. *}
{*-------------------------------------------------------------------*}

{/query}
</BODY>
```

retail2.hsql Sample Input File

```
*---------------------------------------------------------------
*
* Notes about this sample htmSQL data source file.
*
* o You do not need to save this file from your Web browser.
* This file is available in the htmSQL "samples" subdirectory.
*
* o This data source definition file specifies one data source
* named 'retail'.
*
* o The retail data source specifies the SAS/SHARE server
* 'sampnode.shr10'.
*
* o The server name is derived as follows:
*
* The SAS/SHARE server is running on sampnode.pc.sas.com
* (machine name). The ID (which is also the TCP/IP service name)
* is shr10. The following job is running on
* sampnode.pc.sas.com:
*
* options comamid=tcp;
* proc server id=shr10;
* run;
```

```
*
* o A data source definition file can include one or more
* libraries. However, because the library we are using,
* SASHELP, is already predefined to the server, you do
* not need to define it in the data source file.
*
* o Note that the 'retail' data source is referenced in
* the associated 'retail.hsql' file. Similarly, the 'sashelp'
* library is also referenced in 'retail.hsql'.
*
*-------------------------------------------------------------

* -------------------------------------------------------------
* Identify Data Sources
*
* In this example, the data source is named 'retail'.
*
* -------------------------------------------------------------

Data Source Name:retail=sas

* -------------------------------------------------------------
* Define Method
* -------------------------------------------------------------

Method tcp:COMMEXE=wqetcp
Methods:tcp=

* -------------------------------------------------------------
* Define Server
*
* In this example, the server is named 'sampnode.shr10'.
*
* -------------------------------------------------------------

Server sampnode.shr10:AM=tcp
Server sampnode.shr10:SASSHARE=yes
Server sampnode.shr10:Server Version=6
Server sampnode.shr10:ServerAddress=sampnode.pc.sas.com
Server sampnode.shr10:TraceFlags=0
Servers:sampnode.shr10=

* -------------------------------------------------------------
* Define Data Sources
*
* In this example, the data source is named 'retail'.
* Note that this data source is identified under the
* 'Identify Data Sources' section of this file.
*
* -------------------------------------------------------------

retail:Description=Retail Data Source
retail:Server=sampnode.shr10
retail:UndoRequired=y
```

Chapter 3

Syntax for htmSQL Directives

Overview of htmSQL Directives

htmSQL directives are commands that process SQL statements and results sets for your Web page. For more information on the structure of htmSQL input files, see "htmSQL Input Files" on page 6.

The following rules apply to all of the directives:

- directives are delimited by braces {}
- all htmSQL keywords (directives, parameters, parameter values) are not case-sensitive
- unless otherwise noted, parameter values can be delimited by either double or single quotation marks
- white space within a directive is ignored
- no line breaks are allowed in the middle of any string that is delimited by quotation marks (").

Dictionary

{query} Directive

Delimits the query section

Syntax

{**query** datasrc="*htmSQL-ds*" server="*host*:*port*" userid="*id*" password="*pw*" <sapw="*sapw*">}
...{/query}

Required Arguments

datasrc="*htmSQL-ds*"
htmSQL-ds is a name or a variable reference that identifies an htmSQL data source.

Examples:

```
{query datasrc="sales data"}

{query datasrc="{&dsrc}"}
```

A data source specifies a SAS/SHARE server and the libraries that are available through the server. Data sources are defined in data source definition files. To define a data source, use the dsdef program that is provided with htmSQL.

server="*host*:*port*"
host:*port* specifies the SAS/SHARE server to connect to. You can use the server= parameter instead of the datasrc= parameter to specify the SAS/SHARE server. When used alone, the server= parameter must specify both the host name and the port for the SAS/SHARE server.

You can also use server= together with datasrc= to override the host and port that are specified in a data source definition. When used together with the datasrc= parameter, the server= parameter can specify the host name, the port, or both for the SAS/SHARE server. If you specify only one of these items, you must include a colon (:) to indicate which one you are specifying.

If any libraries are defined in the data source definition that is specified by the datasrc= parameter, then htmSQL attempts to define those same libraries to the server that is specified by the server= parameter.

The host name can be specified as a fully qualified domain name or it can be specified in any shortened form that is sufficient to enable network services to identify it.

The port can be specified as a number or as a service name that is defined in the TCP/IP SERVICES file.

Users who are familiar with the SAS syntax for specifying a server name can use a period (.) instead of a colon (:) to separate the host name and port. All of the other syntax rules for the server= option still apply. The following are examples of valid syntax:

```
{query server="klondike.acme.com:5228"}

{query server="penn.sylvania:6500"}

{query server="yukon.sasshr1"}

{query datasrc="finance" server="testsrv:"}

{query datasrc="sales & Marketing" server=":5010"}

{query datasrc="alaska" server="yukon:sasshr1"}
```

If you are just getting started with htmSQL and do not want to define a data source definition file, you can use the server= parameter instead of defining a data source. And, if you specify a port number instead of a service name for this parameter, you also do not need to add an entry to your TCP/IP SERVICES file.

userid="*id*" (conditionally optional)
id is a user ID for the system that the SAS/SHARE server runs on. If your server is running in secure mode, you must specify a user ID. This can be done by specifying a value for this parameter or by specifying a user ID in your data source definition. You do not have to specify user IDs in both places. If the data source definition contains a user ID, then the user ID that you specify for this parameter overrides the user ID that is stored in the data source definition.

password="*pw*" (conditionally optional)
pw is the password for the user ID that is specified in the userid= parameter. If your server is running in secure mode, you must specify a password. This can be done by specifying a value for this parameter or by specifying a password in your data source definition. You do not have to specify passwords in both places. If the data source definition contains a password, then the password that you specify for this parameter overrides the password that is stored in the data source definition.

Optional Argument

sapw="*sapw*"
sapw is the SAS/SHARE server access password for users. This must be the same password that is specified in

- the UAPW= option of the SERVER procedure that was used to define the SAS/SHARE server. You must specify a password if user access to the server is password-protected and if this password is not already specified in your data source definition.
- the SAPW= option of the LIBNAME statement and the SQL procedure's CONNECT TO statement.

The password that you specify for this parameter overrides the password that is stored in the data source definition.

Details

The {query} and {/query} directive pair delimits the query section. An input file can contain multiple query sections. Query sections can be nested within the eachrow and norows sections of a query section and within the success , error , and norows sections of an update section. Each query section must contain at least one SQL/eachrow section pair and can contain multiple pairs.

- The SQL section is delimited by the {sql} and {/sql} directive pair and contains the query to send to the SAS/SHARE server.

- The eachrow section is delimited by the {eachrow} and {/eachrow} directive pair. htmSQL applies the details in the eachrow section to the results set that is generated by the SQL section that immediately precedes that eachrow section. The eachrow section is processed once for each row in the results set.

A query section can also contain a norows section, a {library} directive, and other text, including HTML and variable references. The text can appear before sections, between sections, and after sections.

Example

The following example illustrates a typical query section:

```
{query datasrc="data_source_name"}

{sql}
   [SQL query here]
{/sql}

<p>This text is always output.</p>

{norows}
   [Things to do if no rows are returned]
{/norows}

<p>This text is output only when some rows are returned.</p>

{eachrow}
   [HTML formatting here]
{/eachrow}

<p>More text that is output only when some rows are returned.</p>

{/query}
```

{sql} Directive

Delimits the SQL section

Syntax

{**sql** <empty="success"|"error"> <error="noprint">}…{/sql}

Optional Arguments

empty="success"|"error"

The value for empty= specifies whether processing transfers to the success section (**`empty="success"`**) or error section (**`empty="error"`**) if the SQL section resolves to an empty section. The default value is **`error`**.

Note that this parameter is used only for SQL sections that are within update sections.

error="noprint"

Specify **`error="noprint"`** if you want to suppress error messages that are produced by htmSQL during SQL processing.

Note that if you do not specify this option and your SQL statement contains a SAS data set password, then you risk exposing the password because htmSQL includes the SQL statement along with the SQL error message.

Details

Overview of the SQL Section

The {sql} and {/sql} directive pair delimits the SQL section. The SQL section is a part of both the query and update sections and contains the SQL statements to be sent to the SAS/SHARE server. You can use any SQL statement that is supported by the SAS SQL processor. You can have only one SQL statement per SQL section, but you can have multiple SQL sections within both the query and update sections.

For more information about using SQL statements with SAS data, see the *Base SAS Procedures Guide*.

SQL for a Query Section

In a query section, the information between the beginning and ending {sql} directives must begin with the SELECT keyword and must contain one valid SQL query.

An SQL query can be either static or dynamic.

- If you want each of your users to use the same query every time they access your Web page, write a static query. Static queries consist of expressions and clauses that contain constant values and no variable references.
- With a dynamic query, the users of your Web page can customize the query by specifying their own values for search parameters. The query is written using variable references that are given values when users access the Web page.

  For example, if your data contains a DATE column, and you want users to specify their own dates to search on, you can place a variable reference in the query for DATE. The following example illustrates this query:

  ```
  {sql}
     select NAME, TITLE, DEPT from EMPDB.EMPLOYEE
        where START='{&DATE}'
  {/sql}
  ```

  The values that users provide can be specified on the htmSQL URL or collected from an HTML form that you link them to. If you nest a query in the eachrow section of another query section, your nested query can refer to variables in the results set of the encompassing query.

SQL for an Update Section

In an update section, the information between the beginning and ending {sql} directives must begin with the ALTER, CREATE, DELETE, DROP, INSERT, or UPDATE keyword and must contain one valid SQL statement.

Note: The Webmaster can disable ALTER, CREATE, DELETE, DROP, INSERT, and UPDATE statements by specifying the READONLY option in the htmSQL configuration file.

Example

The following example SQL sections are placed consecutively in an update section. The first section creates a data set named def.play, and the second section inserts values into it:

```
{sql}
   create table def.play
      (a numeric, b numeric, c numeric, d char, e char);
{/sql}

{sql}
   insert into def.play
      set a=1, b=2, c=3, d='xx', e='yy';
{/sql}
```

{eachrow} Directive

Delimits the eachrow section

Syntax

{**eachrow** <n="*n1*"> <first="*n2*"> <last="*n3*"> <closequery="yes|no">}…{/eachrow}

Optional Arguments

n="*n1*"
n1 is the total number of rows that you want htmSQL to retrieve. A value of **`max`** tells htmSQL to retrieve all the rows in the results set. The default value is **`max`**.

The n= and last= parameters are mutually exclusive. When you specify both of them, whichever parameter is specified last is the one that is used. For example, if you specify the following, then the last row that is retrieved is row 10:

```
{eachrow last="15" n="10"}
```

first="*n2*"
n2 is the number of the first row that you want htmSQL to retrieve. The default value is 1.

last="*n3*"
n3 is the number of the last row that you want htmSQL to retrieve. A value of **`max`** specifies the last row in the results set. For example, if you specify the following, then you retrieve all but the first 19 rows in the results set:

```
{eachrow first="20" last="max"}
```

The n= and last= parameters are mutually exclusive. When you specify both of them, whichever parameter is specified last is the one that is used. For example, if you specify the following, then the last row that is retrieved is row 15:

```
{eachrow n="10" last="15"}
```

closequery="yes|no"
`closequery="yes"` tells htmSQL to send the SAS/SHARE server a message to terminate query processing when htmSQL finishes processing the eachrow section. When the SAS/SHARE server terminates query processing, it closes the input tables and frees the memory associated with this query. If you do not specify this parameter

or if you specify **`closequery="no"`**, then query processing is not terminated until htmSQL finishes processing the main input file.

Specify this parameter

- if your htmSQL input file contains more than 64 queries to the same SAS/SHARE server
- if you want to perform a DROP or ALTER TABLE on a table that the query refers to.

Note: If you specify this parameter, then no references to the columns of the results set are allowed after the eachrow section.

Details

The {eachrow} and {/eachrow} directive pair delimits the eachrow section. The eachrow section is a part of the query section and contains instructions on how to format the results that are generated by the SQL section that immediately precedes that eachrow section. The formatting information is applied to each row of the results set and can include any valid HTML tag and variable reference. The variable references contained in the eachrow section are resolved for each row of output as that row is formatted.

Note: Because htmSQL sends all HTML information to **`stdout`** exactly as it is encountered, eachrow sections that are enclosed within HTML **`PRE`** elements might not format the way that you expect them to. If your {eachrow} directive is followed by a line break, htmSQL sends that line break to **`stdout`** and causes the output to appear double-spaced. The following lines

```
<pre>
{eachrow}
X: {&x} Y: {&y}
{/eachrow}
</pre>
```

are sent to **`stdout`** (the Web browser) as

```
<pre>

X: 1     Y: A

X: 2     Y: B

X: 3     Y: C

</pre>
```

and the Web browser displays the following double-spaced output on the Web page:

```
X: 1     Y: A

X: 2     Y: B

X: 3     Y: C
```

You can avoid double-spacing by putting the {eachrow} directive on the same row as the variables:

```
<pre>
{eachrow}X: {&x}     Y: {&y}
{/eachrow}
</pre>
```

If you want to submit more SQL statements from within your eachrow section, you can do one of the following:

- nest one or more SQL sections (with accompanying eachrow and norows sections) in the eachrow section. All queries are sent to the data source that is specified by the encompassing {query} directive.

  Note: You can also nest a {library} directive. However, it is recommended that you nest them only if your {library} directive contains variable references that change as the eachrow section is processed. Otherwise, if the parameters in the {library} directive have static values, then place the {library} directive before the eachrow section to avoid unnecessary processing.

- nest a complete query section in the eachrow section, and use the {query} directive to specify a different data source. The SQL, eachrow, and norows sections that are within the nested section will work with data from this new data source.
- nest a complete update section in the eachrow section to update data from the same data source or from a different data source.

Note: htmSQL does not limit the number of times that you can nest query sections or SQL/eachrow section pairs within eachrow sections. However, beyond a certain point, you might experience poor performance or your system might run out of memory.

{norows} Directive

Delimits the norows section

Syntax

{**norows**}…{/norows}

Details

The {norows} and {/norows} directive pair delimits the norows section. The norows section is a part of the query and success sections. In this section, you include the HTML elements and htmSQL directives that you want htmSQL to process when the previous SQL section does not return or update any rows. After processing a norows section in a query section, htmSQL skips to either the next SQL section or to the end of the query section, whichever comes first. After processing a norows section in a success section, htmSQL skips to the end of the success section.

When an SQL section returns or updates at least one row of data, htmSQL ignores the norows section and continues processing the input file starting with the first line after the norows section.

If you want to submit more SQL statements from within your norows section, you can do one of the following:

- nest one or more SQL sections (with accompanying {library} directive and eachrow, success, error, and norows sections when necessary) in the norows section. All SQL statements are sent to the data source that is specified by the encompassing {query} or {update} directives.
- nest a complete query or update section in the norows section. The SQL, eachrow, success, error, and norows sections that are within the nested section will work with data from the data source that is specified on the {query} or {update} directive.

Note: htmSQL does not limit the number of times that you can nest sections within norows sections. However, beyond a certain point, you might experience poor performance or your system might run out of memory.

{update} Directive

Delimits the update section

Syntax

{**update** datasrc="*htmSQL-ds*" server="*host*:*port*" userid="*id*" password="*pw*" <sapw="*sapw*">}
…{/update}

Required Arguments

datasrc="*htmSQL-ds*"

htmSQL-ds is a name or a variable reference that identifies an htmSQL data source.

Examples:

```
{update datasrc="employee_data"}
{update datasrc="{&dsrc}"}
```

A data source specifies a SAS/SHARE server and the libraries that are available through the server. Data sources are defined in data source definition files. To define a data source, use the dsdef program that is provided with htmSQL.

server="*host*:*port*"

host:*port* specifies the SAS/SHARE server to connect to. You can use the server= parameter instead of the datasrc= parameter to specify the SAS/SHARE server. When used alone, the server= parameter must specify both the host name and the port for the SAS/SHARE server.

You can also use server= together with datasrc= to override the host and port that are specified in a data source definition. When used together with the datasrc= parameter, the server= parameter can specify the host name, the port, or both for the SAS/SHARE server. If you specify only one of these items, you must include a colon (:) to indicate which one you are specifying.

If any libraries are defined in the data source definition that is specified by the datasrc= parameter, then htmSQL attempts to define those same libraries to the server that is specified by the server= parameter.

The host name can be specified as a fully qualified domain name or it can be specified in any shortened form that is sufficient to enable network services to identify it.

The port can be specified as a number or as a service name that is defined in the TCP/IP SERVICES file.

Users who are familiar with the SAS syntax for specifying a server name can use a period (.) instead of a colon (:) to separate the host name and port. All of the other syntax rules for the server= option still apply.

The following are examples of valid syntax:

```
{update server="klondike.acme.com:5228"}
```

```
{update server="penn.sylvania:6500"}

{update server="yukon.sasshr1"}

{update datasrc="finance" server="testsrv:"}

{update datasrc="sales & Marketing" server=":5010"}

{update datasrc="alaska" server="yukon:sasshr1"}
```

If you are just getting started with htmSQL and do not want to define a data source definition file, you can use the server= parameter instead of defining a data source. And, if you specify a port number instead of a service name for this parameter, you also do not need to configure a TCP/IP SERVICES file for htmSQL.

userid="*id*" (conditionally optional)
id is a user ID for the system that the SAS/SHARE server runs on. If your server is running in secure mode, you must specify a user ID. This can be done by specifying a value for this parameter or by specifying a user ID in your data source definition. You do not have to specify user IDs in both places. If the data source definition contains a user ID, then the user ID that you specify for this parameter overrides the user ID that is stored in the data source definition.

password="*pw*" (conditionally optional)
pw is the password for the user ID that is specified in the userid= parameter. If your server is running in secure mode, you must specify a password. This can be done by specifying a value for this parameter or by specifying a password in your data source definition. You do not have to specify passwords in both places. If the data source definition contains a password, then the password that you specify for this parameter overrides the password that is stored in the data source definition.

Optional Argument

sapw="*sapw*"
sapw is the SAS/SHARE server access password for users. This must be the same password that is specified in

- the UAPW= option of the SERVER procedure that was used to define the SAS/SHARE server. You must specify a password if user access to the server is password-protected and if this password is not already specified in your data source definition.
- the SAPW= option of the LIBNAME statement and the SQL procedure's CONNECT TO statement.

The password that you specify for this parameter overrides the password that is stored in the data source definition.

Details

The {update} and {/update} directive pair delimits the update section. An input file can contain multiple update sections. Update sections can be nested within the success, norows, and error sections of an update section and within the eachrow and norows sections of a query section. Each update section must contain at least one SQL section and can contain multiple sections. The update section can also contain a success section and an error section.

- The SQL section is delimited by the {sql} and {/sql} directive pair and contains the SQL statement to be sent to the SAS/SHARE server. The allowed SQL statements are ALTER, CREATE, DELETE, DROP, INSERT, and UPDATE.

Note: The Webmaster can disable these SQL statements by specifying the READONLY option in the htmSQL configuration file.

- The success section is delimited by the {success} and {/success} directive pair and contains instructions on what to do if the SQL statement returns with a return code of zero. The success section can contain a norows section for instances where no rows are updated.
- The error section is delimited by the {error} and {/error} directive pair and contains instructions on what to do if the SQL statement returns with a return code that is not equal to zero.

An update section can also contain a {library} directive and other text, including HTML and variable references. The text can appear before sections, between sections, and after sections.

Example

The following example illustrates a typical update section:

```
{update datasrc="data_source_name"}

{sql}
   [SQL statement here]
{/sql}

{success}
   [Things to do if the return code is 0]
   {norows}
      [Things to do if no rows are returned]
   {/norows}
{/success}

{error}
   [Things to do if the return code is not 0]
{/error}

{/update}
```

{success} Directive

Delimits the success section

Syntax

{**success**}…{/success}

Details

The {success} and {/success} directive pair delimits the success section. The success section is a part of the update section. In this section, you include the HTML elements and htmSQL directives that you want htmSQL to process when the previous SQL section completes with a return code of zero.

When an SQL section returns with a return code that is not equal to zero, htmSQL ignores the success section.

The success section can include a norows section for instances where no rows are updated.

If you want to submit more SQL statements from within your success section, you can do one of the following:

- nest one or more SQL sections (with accompanying {library} directive and success, error, and norows sections when necessary) in the success section. All SQL statements are sent to the data source that is specified by the encompassing {update} directive.
- nest a complete update section in the success section, and use the {update} directive to specify a different data source. The SQL, success, error, and norows sections that are within the nested section will work with data from this new data source.
- nest a complete query section in the success section to query data from the same data source or from a different data source.

Note: htmSQL does not limit the number of times that you can nest sections within success sections. However, beyond a certain point, you might experience poor performance or your system might run out of memory.

{error} Directive

Delimits the error section

Syntax

{**error**}…{/error}

Details

The {error} and {/error} directive pair delimits the error section. The error section is a part of the update section. In this section, you include the HTML elements and htmSQL directives that you want htmSQL to process when the previous SQL section completes with a nonzero return code.

When an SQL section returns with a return code of zero, htmSQL ignores the error section.

If you want to submit more SQL statements from within your error section, you can do one of the following:

- nest one or more SQL sections (with accompanying {library} directive and success, error, and norows sections when necessary) in the error section. All SQL statements are sent to the data source that is specified by the encompassing {update} directive.
- nest a complete update section in the error section, and use the {update} directive to specify a different data source. The SQL, success, error, and norows sections that are within the nested section will work with data from this new data source.
- nest a complete query section in the error section to query data from the same data source or from a different data source.

Note: htmSQL does not limit the number of times that you can nest sections within error sections. However, beyond a certain point, you might experience poor performance or your system might run out of memory.

Variable Reference

Syntax

{&*varname* <format=*formats*> before="*string1* between="*string2*" after="*string3*"}

Required Arguments

&varname

varname is the name of the variable. If the variable that you want to reference was specified in the URL with more than one value, you can use the following syntax to reference a single specific value, a range of values, or all values:

&*varname*[*n*]	references the *n*th value that the variable contains where *n*>=1.
&*varname*[*m..n*]	references all values from the *m*th value to the *n*th value, where *m*<=1<*n* and the variable contains two or more values.
&*varname*[*]	references all values that the variable contains.

In the first two instances, *m* and *n* can be references to numeric variables.

before="*string1*"
between="*string2*"
after="*string3*"

These parameters enable you to output strings of characters before, between, and after variable values. You can use an unlimited number of characters in your strings (note that the maximum number of characters that you can use depends on how much memory your system has). If you want to include double quotation marks within your string, then delimit the string with single quotation marks or use two double quotation marks within your string. For example, both of the following values return the string **`"How are you?"`**:

```
between='"How are you?"'
between="""How are you?"""
```

To include single quotation marks within your string, delimit the string with double quotation marks or use two single quotation marks within your string. For example, both of the following values return the string **`What's up?`**:

```
before="What's up?"
before='What''s up?'
```

The default value for before= is a null string (""). The default value for between= is a blank space (" "). The default value for after= is a null string ("").

Optional Argument

format=*formats*
: *formats* is a single formatting option (**`format=value`**) or a comma-delimited list of options enclosed in parentheses (**`format=(value,...,valueN)`**). Do not use quotation marks to delimit the values. This is an optional parameter. See "Formats for Variable Values and Labels" on page 10 for a list of formatting options.

Details

A variable reference is a string that htmSQL replaces with the value of a variable. Variables are symbols that are defined on the URL, columns that are selected by a query, or symbols that htmSQL automatically defines and supplies values for.

When htmSQL encounters a variable reference, it replaces the reference with the current value of the variable. If the reference is to a column in a results set and

- the variable reference occurs before the SQL section, then the variable is undefined and cannot be resolved. The variable reference is written to **`stdout`** unresolved.
- the variable reference occurs between the SQL section and the eachrow section, then htmSQL replaces the variable reference with the variable's value from the first row of the results set.
- the variable reference occurs within an eachrow section, then htmSQL replaces the variable reference with the current row's value for the variable.
- the variable reference occurs after the eachrow section, then htmSQL replaces the variable reference with the variable's value from the last row of the results set.

htmSQL also supports the resolution of nested variable references. That is, the value of a variable can itself be a variable that htmSQL can resolve. For example, if you have a variable named taxi, and the value of taxi is the variable name driver,

```
{&taxi} -----> driver
```

and the value of driver is **`Bob`**,

```
{&driver} -----> Bob
```

then when you specify **`{&{&taxi}}`**, htmSQL resolves the nested references to a value of **`Bob`**.

```
{&{&taxi}} -----> Bob
```

htmSQL can resolve an infinite number of these nested variable references.

Example

The following are examples of variable references:

```
{&myname}
{&weekdays[1..7]}
{&theworld[*]}
{&array1[{&counter}]}
{&array2[{&begin}..{&end}]}
{&months;[1..12] before="(" between="," after=")"}
{&{&varname}}
{&{&sys.colname[*]}}
```

{library} Directive

Defines a high-level qualifier that you use in the names of tables and views in your SQL queries and statements

Syntax

{**library** sqlname="*table-qualifier*" path="*library-path*"}

Required Arguments

sqlname="*table-qualifier*"
table-qualifier is the high-level qualifier that you use in your SQL for the names of tables and views that reside in this SAS library. This qualifier corresponds to the libref in a SAS program. You can use the keyword **`libref`** as a synonym for sqlname.

path="*library-path*"
library-path is the pathname of the SAS library.

Details

The {library} directive can be included in both the query and update sections and defines a high-level qualifier that you use in the names of tables and views in your SQL queries and statements. Use this directive when the SAS library that contains the tables and views that you want to access is

- not predefined to the SAS/SHARE server through which the library is accessed and
- not defined as part of a data source that specifies that server.

Note:

- The keyword **`libname`** is a synonym for library.
- When you specify a value for sqlname (or libref), you can use that high-level qualifier for any update or query section that specifies the same values for the datasrc= and server= parameters.
- You can also specify the same high-level qualifier for two different libraries if they are on different servers. If you specify the same high-level qualifier for two libraries that are on the same server, then the second value overrides the first one.

{label} Directive

Enables you to display the label for a variable that is in a results set

Syntax

{**label** var="*varname1 varname2 ... varnameN*" <format=*formats*> before="*string1*"
between="*string2*" after="*string3*"}

Required Arguments

var="*varname1 varname2 ... varnameN*"

varname1 varname2 ... varnameN are the names of variables whose labels you want to display. You can specify one or more variable names; separate the variable names with single spaces.

You can also use a variable reference for a value. For example:

```
var="{&mylabel}"
var="{&sys.colname[*]}"
```

before="*string1*"
between="*string2*"
after="*string3*"

These parameters enable you to output strings of characters before, between, and after labels. You can use an unlimited number of characters in your strings (note that the maximum number of characters that you can use depends on how much memory your system has). The only character that is not allowed is the double quotation mark ("). If you want to include double quotation marks within your string, then delimit the string with single quotation marks or use two double quotation marks within your string. For example, both of the following values return the string **`"How are you?"`**:

```
between='"How are you?"'
between="""How are you?"""
```

To include single quotation marks within your string, delimit the string with double quotation marks or use two single quotation marks within your string. For example, both of the following values return the string **`What's up?`**:

```
before="What's up?"
before='What''s up?'
```

The default value for before= is a null string (""). The default value for between= is a blank space (" "). The default value for after= is a null string ("").

Optional Argument

format=*formats*

formats is a single formatting option (**`format=`*`value`***) or a comma-delimited list of options enclosed in parentheses (**`format=(`*`value`*`,...,`*`valueN`*`)`**). Do not use quotation marks to delimit the values. This is an optional parameter. See “Formats for Variable Values and Labels” on page 10 for a list of formatting options.

Details

The {label} directive enables you to display the label for a variable that is in a results set. The label is either returned from the data set or set in the SQL statement. If you want to display the label using a particular format, you can specify the format= parameter.

{include} Directive

Enables you to include other HTML files into the current file

Syntax

{**include** file="*web-server-host-pathname*" vars="*var1=value1&var2=value2&…*"}

Required Arguments

file="*web-server-host-pathname*"

web-server-host-pathname is the pathname (either absolute or relative) for a file that is processed as an htmSQL input file. If the pathname is relative, then it is relative to either the current working directory for htmSQL or to the path of the calling input file. See your setting for the RELATIVE run-time configuration option. You can use a variable reference to specify the filename. For example:

```
{include file="/dept/web/{&proj}.hsql"}
```

The value for the file= parameter must be a physical pathname on the Web server machine. It is not a URL.

vars="*var1=value1&var2=value2&…*"

var1=value1&var2=value2&... is one or more variable name and value pairs that the included input file requires. The variables that you specify exist in the scope of the included file. This scoping is done so that a variable that is set by both the input file and the included file can retain separate values for each file. When htmSQL finishes processing the included file and returns to the calling input file, the value of the variable is restored to the value that it had before the included file was called. If a variable is defined only for the included file, then you cannot access it after htmSQL returns to the calling file.

The following example uses an input file named emps.hsql that requires values for the variables name and status:

```
{include file="emps.hsql" vars="name={&emp}&status=EXEMPT"}
```

Note: In this example, you can use one or more variable references (such as {&emp}) in the value for the vars parameter.

Details

The {include} directive enables you to include other HTML files into the current file. The included file can be a simple HTML file or another htmSQL input file. If it is an htmSQL input file, then it must be complete; it cannot contain a partial query or update section.

The {include} directive cannot appear inside any other htmSQL directive section.

Comment

Specifies an htmSQL comment that is not written to **`stdout`**

Syntax

{**your comments here*}

Details

All text contained between {* and the closing brace } is considered an htmSQL comment and is not written to **`stdout`**. You can comment out single directives or entire sections with one pair of comment braces.

Note: HTML comments are considered text and are written to **`stdout`** along with the other text in the input file.

Example

The following example shows an entire query section that is commented out:

```
{*
{query datasrc="employee"}
   {sql}
      select * from empdb.employee
   {/sql}
   {eachrow}
      lastname: {&lname} firstname: {&fname}
   {/eachrow}
{/query}
}
```

Chapter 4
Administering htmSQL

The htmSQL Configuration File

Specifying and Naming the Configuration File

The htmSQL configuration file contains the values for the htmSQL run-time configuration options. htmSQL can run both with and without a configuration file. If you want it to run with a configuration file, htmSQL can automatically locate your file if you follow these rules for naming and locating the file. Or if you need to name or locate the file in a manner different from what is required, you can explicitly specify the file's pathname in either the -rc parameter or the HTMSQL_CFG Web server environment variable.

If you want htmSQL to automatically locate your file, you must do both of the following:

- Name the configuration file

  ```
  executable_name.cfg
  ```

  where *executable_name* is the name of the htmSQL executable file. For example, if the name of your htmSQL executable file is **`htmSQL`**, then name the configuration file **`htmSQL.cfg`**. If the name of your htmSQL executable file is **`htmSQL.exe`**, then you still name the configuration file **`htmSQL.cfg`**. If you rename the executable file, then you must also rename the configuration file to match.

Note: If you are a UNIX user and you are upgrading from a previous release of htmSQL, if your configuration file is named .htmSQLrc, then you can keep that name (that is, you do not need to follow the rule for naming the file).

- Put the configuration file in a location that htmSQL knows about. htmSQL looks for the configuration file
  1. first in the current directory
  2. then in the directory where the htmSQL executable file is located
  3. and if it still cannot find the file, then in
     - **`/usr/local/lib/IntrNet/htmSQL`** (for UNIX and z/OS)
     - **`C:\Program Files\SAS Software`** (for Windows)

Note: If htmSQL does not find a configuration file, then it runs without a configuration file. If htmSQL finds a configuration file but cannot open it, then htmSQL does not run.

If you want to explicitly specify the name and location of your configuration file, then provide either a relative or absolute pathname for one of the following:

- the -rc parameter when you run htmSQL from the command line
- the HTMSQL_CFG Web server environment variable.

You can name your configuration file whatever you want and locate the file in whichever path you choose.

Both of the values are optional, but if you specify both values, then the value of the -rc parameter takes precedence over the value of the HTMSQL_CFG environment variable.

Note: If htmSQL cannot open the file, then htmSQL does not run.

Customizing the Configuration File

A default configuration file is downloaded with the htmSQL package. Modify the preset options to match your needs.

The syntax rules for the file are as follows:

- all options must be specified as **`option = value`** on a single line unless a continuation character is specified
- option names are not case-sensitive
- white space characters around the option names and values are not significant
- blank lines are ignored
- any line that begins with !, #, or * is ignored.

The following configuration options are available:

CONTENT-TYPE
: The CONTENT-TYPE option specifies the string to include in the Content-type HTTP header that is output by htmSQL. If this option is not specified, the header defaults to **`text/html`**. If the option is specified with no value, then no Content-type header is generated. For example:

```
CONTENT-TYPE =
```

CONTINUATION

The CONTINUATION option specifies the list of continuation characters that can be used in your configuration file. If the last non-blank character of the line is a continuation character, then at run time, the continuation character (and all blank spaces that immediately precede and follow the continuation character) is replaced with the contents of the next line (minus leading white space).

- If you continue an option value to the next line and the first character of the continued text is an !, #, or *, do not place the character in column 1 because htmSQL interprets it as a comment character and ignores the rest of that line.
- The CONTINUATION option itself cannot be continued from one line to the next (that is, the CONTINUATION option must be specified on a single line).

If the CONTINUATION option is not specified, then the continuation character defaults to the backward slash (\). For example:

```
CONTINUATION = +\,
DATASRCFILE = /local/disk1/htmSQL/htmSQL.datasrc: \
              /local/disk1/htmSQL/alt.datasrc: +
#             /local/disk2/hr/apps/per.dsf: \
              /local/disk2/hr/apps/per2.dsf: ,
              /local/disk2/fac/apps/fac.datasrc
```

at run time is

```
DATASRCFILE = /local/disk1/htmSQL/htmSQL.datasrc:/local/disk1/htmSQL/alt.datasrc:
   /local/disk2/hr/apps/per2.dsf:/local/disk2/fac/apps/fac.datasrc
```

DATASRCFILE

The DATASRCFILE option specifies the pathnames of one or more data source definition files to use. If this option is not specified, then the pathname defaults to

- **/usr/local/lib/IntrNet/htmSQL/htmSQL.datasrc** (for UNIX and z/OS)
- **C:\Program Files\SAS Software\htmSQL.dsf** (for Windows).

If you specify more than one data source definition file, then specify a path separator character between the pathnames. (Path separator characters are listed in the PATHSEPARATOR option.) htmSQL loads the definitions from the files in the order in which they are specified. If the same data source or SAS/SHARE server is specified in two files, then the later definition overrides the earlier one.

Some examples:

For UNIX and z/OS:

```
DATASRCFILE = /usr/local/SAS/htmSQL/data_sources
DATASRCFILE = /local/htmSQL/htmSQL.datasrc:/local/htmSQL/alt.datasrc
```

For Windows:

```
PATHSEPARATOR = ;
DATASRCFILE = c:\htmSQL\mydata.src
DATASRCFILE = c:\htmSQL\mydata.src;c:\htmSQL\yourdata.src
```

EXPORT

The EXPORT option enables the Webmaster to make Web server environment variables available as htmSQL variables. The default htmSQL configuration file that comes with the htmSQL package exports the following environment variables. The Webmaster can add to or delete from this list:

```
EXPORT = HTTP_USER_AGENT, REMOTE_ADDR, REMOTE_HOST, REMOTE_USER
```

Although htmSQL variable names are not case-sensitive, environment variable names are case-sensitive. The Webmaster must specify the proper case when referring to a variable in the EXPORT option, but anyone creating an htmSQL input file can use uppercase, lowercase, or a combination of the two cases.

Note: Because htmSQL does not distinguish between upper- and lowercase, the Webmaster can export only one environment variable whose case-normalized name is a given sequence of characters (for example, you cannot export both the HOME and the home environment variables).

FULLHEADER

The FULLHEADER option specifies whether htmSQL generates a complete set of HTTP headers. This option is intended for Web servers that require CGI programs to generate a full set of HTTP headers (typically, the Web server generates these header lines).

The values for this option are **`YES`**, **`Y`**, **`TRUE`**, **`T`**, **`NO`**, **`N`**, **`FALSE`**, and **`F`** (case is not significant).

When the value of FULLHEADER is **`YES`**, **`Y`**, **`TRUE`**, or **`T`** or if no value is specified, then htmSQL generates a complete set of HTTP headers, which consists of the following lines:

```
HTTP/1.0 200 OK
MIME-Version: 1.0
```

plus the Content-type HTTP header, as indicated by the CONTENT-TYPE option. If the CONTENT-TYPE option is specified with no value, then the FULLHEADER option is ignored.

When the value of FULLHEADER is *NO*, **`N`**, **`FALSE`**, or **`F`** or if the option is not specified, then the complete set of headers is not generated.

INCLUDE

The INCLUDE option specifies a list of filename patterns. The patterns are specified with path separator characters in between them. Any input filename must match at least one of the patterns in the list. If this option is not specified, input filenames are not required to match any pattern. If both this option and the NOINCLUDE option are specified, any input file must match at least one pattern in the INCLUDE list and must not match any pattern in the NOINCLUDE list. The following example allows only files whose names end in .hsql or .hsq:

```
INCLUDE = *.hsql:*.hsq
```

LAST-MODIFIED

The LAST-MODIFIED option specifies whether htmSQL generates a Last-Modified HTTP header, which shows the date and time at which htmSQL executes.

The values for this option are **`YES`**, **`Y`**, **`TRUE`**, **`T`**, **`NO`**, **`N`**, **`FALSE`**, and **`F`** (case is not significant).

When the value of LAST-MODIFIED is **`YES`**, **`Y`**, **`TRUE`**, or **`T`** or if the option is not specified or is specified without a value, then a Last-Modified header is generated.

If the value is **`NO`**, **`N`**, **`FALSE`**, or **`F`**, the header is not generated.

For example:

```
LAST-MODIFIED = TRUE
```

NOINCLUDE

The NOINCLUDE option specifies a list of filename patterns. The patterns are specified with path separator characters in between them. Any input filename must not match any of the patterns in the list. If this option is not specified, then the value defaults to an empty list. If both this option and the INCLUDE option are specified, any input file must match at least one pattern in the INCLUDE list and must not match any pattern in the NOINCLUDE list. The following example disallows any input file whose name begins with **`local`** or **`etc`** or includes a subdirectory named **`private`**:

```
PATHSEPARATOR = ;
NOINCLUDE = c:\local\*;c:\etc\*;c:\*\private\*
```

PATHSEPARATOR

The PATHSEPARATOR option specifies the list of characters that can be used to separate pathnames or path patterns in the values of the INCLUDE, NOINCLUDE, and DATASRCFILE options.

Specifying one separator character between pathnames and patterns is sufficient, although you are allowed to specify more than one character. For example:

```
PATHSEPARATOR = ;
INCLUDE = *.hsq;*.hsql;;*.html
```

If this option is not specified, then the path separator character defaults to the colon (:).

PRAGMA

The PRAGMA option specifies the string to include in the Pragma HTTP header that is output by htmSQL. If the PRAGMA option is not specified or is specified without a value, then the Pragma header is not generated.

For example:

```
PRAGMA = no-cache
```

READONLY

The READONLY option specifies whether the SQL UPDATE, INSERT, DELETE, CREATE, DROP, and ALTER statements are allowed in htmSQL input files.

The values for this option are **`YES`**, **`Y`**, **`TRUE`**, **`T`**, **`NO`**, **`N`**, **`FALSE`**, and **`F`** (case is not significant). Specify **`YES`**, **`Y`**, **`TRUE`**, or **`T`** to prevent users from using these SQL statements. If the READONLY option is not specified, then the statements are allowed.

For example:

```
READONLY = YES
```

REFRESH

The REFRESH option specifies the string to include in the Refresh HTTP header that is output by htmSQL. When the Web browser receives the Refresh header, it automatically reloads the document after a delay of a specified number of seconds. When you specify the REFRESH option, you must specify the number of seconds that the Web browser delays. You can specify a URL that designates an alternate Web page to load at the refresh time.

If the REFRESH option is not specified or is specified without a value, then the Refresh header is not generated.

Some examples:

```
REFRESH = 3
REFRESH = 3,URL=http://support.sas.com
```

Note: You can achieve the same refresh effect (on Web browsers that support them) on a per-page basis by including an HTML META element in the htmSQL input file. For example:

```
<META HTTP-EQUIV="Refresh" CONTENT="3;URL=http://support.sas.com">
```

RELATIVE

The RELATIVE option specifies whether the relative pathnames of included input files are specified with respect to the location of the calling input file or with respect to the working directory. By default, htmSQL treats relative pathnames of included input files as being relative to that of the calling input file (a value of **`CALLING`**). If you want to change the default so that htmSQL regards the pathname as being relative to the working directory, then specify a value of **`WORKING`**. For example:

```
RELATIVE = WORKING
```

SET

The SET option enables you to specify default values for variables. You can specify one or more variable name and value pairs. The pairs are separated by ampersands (&) and are specified using the following format:

```
SET var1=value1&var2=value2&...varN=valueN
```

The values can contain URL-encoded data.

You can specify multiple instances of the SET option in a configuration file. For example:

```
SET var1=value1&var2=value2
SET var3=value3
```

YEARDIGITS

The YEARDIGITS option specifies the number of digits that htmSQL uses in the year portion of its date and datetime formats. You can specify a value of either **`2`** or **`4`**. For example:

```
YEARDIGITS = 4
```

If this option is not specified, then the value of YEARDIGITS defaults to **`2`**. This option also affects certain automatic variables.

Defining a Data Source

Overview of Defining a Data Source

A data source identifies a SAS/SHARE server that htmSQL can get data from. A data source definition can also include SAS libraries or an external database management system (DBMS) that htmSQL accesses through that server.

After the Webmaster defines a data source, an htmSQL programmer can access it by specifying its name in the query or update section of an htmSQL input file.

Creating a Data Source Definition File

A program called dsdef is supplied with htmSQL. Use dsdef to define data sources for htmSQL. dsdef prompts the user for data source information and then creates or updates a data source definition file. For invocation and syntax information about dsdef, see "Instructions for Invoking dsdef" on page 57.

Note: The data source definition file should be modified only by the Webmaster. The file could be corrupted if it is simultaneously modified by multiple users.

Using dsdef

dsdef prompts for information about data sources, SAS libraries, and SAS/SHARE servers. If the data source, library, or server that you specify was defined previously, dsdef puts the existing attribute values in square brackets ([]) next to the prompts for new values. You can accept the existing value by not entering a new value and pressing **Enter**.

The following dialog box is generated by dsdef. To get more information about each step, select the number that precedes the prompt.

```
1 SystemPrompt> dsdef<return>

Configure data sources for htmSQL
=================================

Use this program to create or modify the definition of one or more data
sources for htmSQL.

A data source specifies exactly one SAS/SHARE server and may also specify
one or more SAS data libraries or an external DBMS to be accessed through
the server.

Data source names can be any length and can contain any character except a
double quote ("). They are case-sensitive and must be entered in an htmSQL
input file exactly as they are defined.

In the dialog that follows, default or previously specified values are shown
in square brackets ([]); to accept such a value, press return or enter.
The only required values in a data source definition are the data source
name and server name. You can omit all other values by press return or
enter when you are prompted for them.

When you have finished defining data sources, you can save your changes
by pressing return or enter at the 'Enter a Data Source Name' prompt.
You can cancel your changes by entering a 'c' instead.

2 Enter a Data Source Name to configure: datasrc1<return>
Enter information for: datasrc1
 3 Description: sample data source<return>
 4 SAS/SHARE server name (host.service): node1.server1<return>
5 Require SAS SQL processor to undo partial updates? (usually NO): <return>
6 DBMS to pass SQL to (omit for SAS data): <return>
7 Options to pass when connecting to DBMS: <return>
 Enter information for: Server node1.server1
8 SAS/SHARE server host IP name (fully qualified) or address [node1]: <return>
9 User ID for SAS/SHARE server host: <return>
10 Password for specified user ID: <return>
11  SAS/SHARE server user access password: pword<return>
12 Enter a library in data source "datasrc1" to configure: userlib1<return>
Enter information for: datasrc1 USERLIB1
13 Description: a sample library<return>
14 Library path name: sasuser/<return>
15  SAS engine the SAS/SHARE server should use: <return>
16  Options (only ACCESS=READONLY and SLIBREF=server-libref supported): <return>
17 Enter a library in data source "datasrc1" to configure: <return>
18 Enter a Data Source Name to configure: datasrc2<return>
Enter information for: datasrc2
3 Description: sample data source 2<return>
4 SAS/SHARE server name (host.service): node1.server1<return>
5 Require SAS SQL processor to undo partial updates? (usually NO): <return>
6 DBMS to pass SQL to (omit for SAS data): <return>
7 Options to pass when connecting to DBMS: <return>
19 Do you want to update configuration for server node1.server1? <return>
17 Enter a library in data source "datasrc1" to configure: <return>
18 Enter a Data Source Name to configure: <return>
```

The following steps explain the information that you must provide to the dsdef program.

1 At the system command-line prompt, enter **dsdef**. If you want to save your data source definition file in a directory other than the default directory, you must specify the -config option and the pathname for the file. The following example illustrates this:

```
dsdef -config c:\htmSQL\mydata.dsf
```

If -config is not specified, the definition is written to a default pathname. If the definition file already exists, it is updated. Otherwise, the file is created.

Note: To end the program, enter `c` to cancel without saving or press **Enter** to save your data source information and then end the program. Depending on where you are in the program, you might need to press **Enter** more than once to completely exit the program.

2 At the **Enter a Data Source Name to configure:** prompt, enter the name of your data source. This is the value that you specify for the datasrc= parameter of the {query} or {update} directive that you specify in your htmSQL input (.hsql) file.

A data source name can be of any length and can contain any character (including blank spaces) except for the following characters: []{}()"?*=!@,:;. Use a name that you can remember and type accurately. Case is significant in data source names.

3 At the **Description ():** prompt, enter a description of the data source. This value is optional. The description can be up to 1,024 characters long.

4 At the **SAS/SHARE server name (host.service):** prompt, enter the name of the SAS/SHARE server for this data source. Specify a two-part name (*host.service* or *host.port*), where

- *host* is the nodename of the machine where the server runs
- *service* is the service name that is specified when the SAS/SHARE server is defined as a service in the TCP/IP SERVICES file. The service name is specified for the ID= option of the PROC SERVER statement that is used to define the SAS/SHARE server.

5 *port* is the port number of the SAS/SHARE server. This two-part name is the same name that you specify in a LIBNAME or PROC SQL CONNECT TO statement in a SAS program.

Note: If you use a port number to identify a SAS/SHARE server, then you do not need to modify the SERVICES file on the Web server machine.

6 At the **Require SAS SQL processor to undo partial updates? (usually NO):** prompt, specify the setting for the UNDO_POLICY option of the SAS SQL processor. The following values are valid:

n, N, no, or NO (default value)	resets UNDO_POLICY to NONE. NONE specifies that if the UPDATE or INSERT of a row fails, then any rows that were updated or inserted by that SQL statement (before the failure) remain inserted or updated.
y, Y, yes, or YES	retains the default value (REQUIRED) of UNDO_POLICY. REQUIRED specifies that if the UPDATE or INSERT of a row fails, then any rows that were updated or inserted by that SQL statement (before the failure) are undone.

7 At the **DBMS to pass SQL to (omit for SAS data):** prompt, if your data is in an external DBMS, specify the SAS/ACCESS engine for the DBMS. Example values are **`DB2`**, **`ORACLE`**, and **`SQLDS`**. If your data is in a SAS library, do not specify a value.

8 At the **Options to pass when connecting to DBMS:** prompt, enter any options that are required for connecting to the external DBMS. The exact options that are available and the exact option names depend on the DBMS that you specify for step 6 and for the SAS/ACCESS view engine for that DBMS. The connection options correspond to the DBMS arguments that are documented in the SQL Procedure Pass-Through facility's documentation for that SAS/ACCESS view engine. Example

values are `USERID=`*`userid`* and `PASSWORD=`*`password`*, where *userid* and *password* are the user ID and password for the DBMS.

9 At the **SAS/SHARE server host IP name (fully qualified) or address [node1]:** prompt, enter the server's nodename. If you do not enter a nodename, this value defaults to the nodename that you specified in step 4 (in this example, `node1` is the default value). In a complex environment, you might need to specify a fully qualified domain address for the server, such as `server1.unx.sas.com`.

10 At the **User ID for SAS/SHARE server host:** prompt, enter a user ID for the system that the server runs on. This is an optional value that you specify if the server is running in secure mode. Otherwise, the value is ignored.

If you omit the user ID from the data source definition, the htmSQL programmer must specify the user ID in the htmSQL input file by using the userid= parameter of the {query} or {update} directive.

11 At the **Password for specified user ID:** prompt, enter the password for the user ID that you specified in step 9. This is an optional value that you specify if the server is running in secure mode. Otherwise, the value is ignored.

If the server is running in secure mode and you omit the password from the data source definition, the htmSQL programmer must specify the password in the htmSQL input file by using the password= parameter of the {query} or {update} directive.

12 At the **SAS/SHARE server user access password:** prompt, enter the server access password for users. This is an optional value. This must be the same password that is specified in

- the UAPW= option of the SERVER procedure that was used to define the SAS/SHARE server. You must specify a password if user access to the server is password-protected.
- the SAPW= option of the LIBNAME statement and the SQL procedure's CONNECT TO statement.

13 If the SAS library that contains your data is not predefined to the SAS/SHARE server, then at the **Enter a library in data source "datasrc1" to configure:** prompt, enter a libref for the library. htmSQL programmers use this libref as the high-level qualifier for the table names in the SQL queries and statements that their applications send to the SAS/SHARE server. Steps 13 through 16 request additional information about this library.

The library name can be up to 8 characters long. The first character must be a letter or an underscore. Subsequent characters can be letters, numeric digits, or underscores. Blanks and special characters are not allowed.

14 At the **Description ():** prompt, enter a description of the library. This value is optional. The description can be up to 1,024 characters long.

15 At the **Library path name:** prompt, enter the physical name of the library. This must include a valid pathname for the operating system in which your server library is stored.

16 At the **SAS engine the SAS/SHARE server should use:** prompt, specify the SAS engine that is required for writing to and reading from this server library. This option is required only if you do not want the SAS/SHARE server to use the engine that the server selects by default. For information about other engines, see the description of the LIBNAME statement in the SAS companion for the operating system in which your server library is stored.

17 At the **Options (only ACCESS=READONLY and SLIBREF=server-libref supported):** prompt, specify one or both of the following values (these values are optional):

SLIBREF=*server-libref*	specifies the server's library reference name for the library.
ACCESS=READONLY	gives users read-only access to the SAS data sets in the library.

18 At the **Enter a library in data source "datasrc1" to configure:** prompt, you can either enter the name of another server library or you can press **Enter** if you do not want to add any more libraries to this data source.

Note: If you do specify another library, dsdef takes you through steps 13 through 16 for that library. If you do not specify another library, dsdef proceeds to step 18.

19 At the **Enter a Data Source Name to configure:** prompt, you can either enter the name of another data source or you can press **Enter** if you do not want to add any more data sources.

Note: If you do specify another data source, dsdef takes you through steps 3 through 7 for that data source. If you do not specify another data source, the dsdef program ends.

20 If in step 4 you specify a SAS/SHARE server that is already defined for the data source, dsdef prompts to see whether you want to update the server configuration information. You can either enter **`yes`** or press **Enter** for **`no`**.

Note: If you do specify **`yes`**, dsdef takes you through steps 8 through 11 so you can update the information for that server. Otherwise, dsdef proceeds to step 12.

Instructions for Invoking dsdef

dsdef is a line-mode configuration program that defines data sources for use with htmSQL. It enables you to configure data sources and the SAS/SHARE servers and libraries that you include in the data sources.

Syntax

Use the following syntax to invoke dsdef:

```
dsdef -config pathname
```

where -config *pathname* is optional, and *pathname* specifies the pathname (including the filename) for the data source definition file. You can use -c as an alias for -config.

The definition information is written to the definition file specified by the -config option. If -config is not specified, the definition is written to a default pathname of

- **`/usr/local/lib/IntrNet/htmSQL/htmSQL.datasrc`** (for UNIX and z/OS)
- **`C:\Program Files\SAS Software\htmSQL.dsf`** (for Windows).

If you name your definition file something other than the default pathname, you must specify the following line in the htmSQL configuration file:

```
datasrcfile = pathname
```

where *pathname* is the full pathname of your data source definition file.

If the definition file already exists, it is updated. Otherwise, the file is created.

Note: The directory for the data source definition file must exist before you invoke dsdef. Otherwise, no file is created.

Examples

For UNIX and z/OS

```
dsdef -config /usr/local/data_source/employee.data_source
dsdef -c /myfiles/financial.datasrc
```

For Windows:

```
dsdef -config c:\mydata\datasources\personal.dsf
dsdef -c m:\network\central\datasources\mis.datasource
```

For more information on the dsdef dialog box, see "Using dsdef" on page 53.

Configuring TCP/IP

htmSQL uses TCP/IP to communicate with a SAS/SHARE server. To enable communication between your Web server and a SAS/SHARE server through TCP/IP, you must perform the following configuration steps:

- Specify one of the following options on the SAS command line or in an OPTIONS statement when you start the SAS/SHARE SERVER procedure (PROC SERVER):
  - **`COMAMID=TCP`**
  - **`COMAUX1=TCP`**
- Define the SAS/SHARE server in the TCP/IP SERVICES file that is on the SAS/SHARE server machine and on the Web server machine.

  Note: If you use a port number to identify a SAS/SHARE server, then you do not need to modify the SERVICES file on the Web server machine.

  - For UNIX and z/OS, the SERVICES file is

    ```
    /etc/services
    ```

  - For Windows NT and Windows 2000, the SERVICES file is

    ```
    %SYSTEMROOT%/system32/drivers/etc/SERVICES
    ```

    where %SYSTEMROOT% is the directory where Windows NT is installed.

Each entry in the SERVICES file associates a service name with the port number and communications protocol that are used by that service. For htmSQL, use the name of the SAS/SHARE server as the service name. An entry for a SAS/SHARE server has the form

```
<server-name> <port number>/tcp # <comments>
```

The server name must be 1-8 characters long and is generally case-sensitive. The first character must be a letter or underscore; the remaining 7 characters can include letters, digits, underscores, the dollar sign ($), or the at sign (@). You specify this same server name when you define the server for your data source (either in the data source definition file or in the server= parameter of the {query} or {update} directive).

Index