SAS SQL Library for C, Version 1.3

The SAS SQL Library for C provides an application programming interface (API) that enables your applications to send SQL queries and statements through a SAS/SHARE server to data on remote hosts.

This library of C functions enables you to build small, fast applications that do not require the full power of SAS software. You can create applications that query SAS data and database management system (DBMS) data. These applications can also use UPDATE, INSERT, and other SQL statements to create and update SAS data.

If you have questions or comments after reading this information, please let us know.

What's  in the SAS SQL Library for C?

Version 1.3

August, 1999

New Feature

The `sasSQLconnect()` function now contains a parameter that enables you to specify the number of rows to request from the server for a single fetch request.

Moving to SAS/SHARE Software

The SAS SQL Library for C is now available as part of SAS/SHARE software, Version 8.

You can find the SQL library software in the `sasmisc` directory of your SAS/SHARE installation.

Version 1.2

June 25, 1998

New Platforms

- The SAS SQL Library for C is now available for the Windows NT and Windows 95 platforms.

New Features

- We replaced three of the parameters for `sasSQLconnect()` with a single parameter block that includes the original three parameters as well as additional information.
- We added a description element to the `SCA_ATTR_NV` structure. The GET_NV function can display this description when it prompts for values.
- You no longer need to define a data source if your application can identify the SAS/SHARE server explicitly by specifying the host name or IP address and the service name or port number.
- If your application uses a port number to identify a SAS/SHARE server, then you do not need to modify the SERVICES file on client machines when you configure TCP/IP.
- We added two new return codes: `SCA_ERROR_DESCERR` and `SCA_SETUP_GENERR`.
- We added a new `-server` parameter for the qsas sample application.

Enhancements

- We enhanced the documentation about data source definitions and user-defined functions.

Requirements

The SAS SQL Library for C runs under UNIX, Windows 95, and Windows NT, each of which provides the TCP/IP software that is required for your client application. **No additional software (other than the TCP/IP software) is required for the client machine.** The software requirements for the server machine depend on the type of data that you want to access:

Type of Data	Server Machine Software Requirements
SAS data	• Base SAS software • SAS/SHARE software • TCP/IP software
DBMS data	• Base SAS software • SAS/SHARE software • SAS/ACCESS interface for each DBMS • TCP/IP software

---

Package Contents

Select one of the following links to view the contents of the SQL library's download package:

- UNIX package
- Windows package

Package Contents for UNIX

The SAS SQL Library for C download package includes the following files and subdirectories:

sasSQL.a
the archive version of the SQL library.

shared library file
one of the following shared versions of the SQL library:

Environment	Filename
AIX/6000	sasSQL.o
Compaq Tru64 UNIX	sasSQL.so
HP-UX	sasSQL.sl
Intel ABI+	sasSQL.so
IRIX	sasSQL.so
Linux	Not available
Solaris 2 for Sparc	sasSQL.so
SunOS 4	sasSQL.so

sasSQL.h
the main header file for the SQL library. This file includes interface structures, typedefs, function

prototypes, and preprocessor symbols

sasSQLrc.h
: a header file that defines preprocessor symbols for the return codes and reason codes that the SQL library functions return.

sampnv.c

sampnv.h
: a working sample implementation of two user-defined functions that are used by the SQL library functions and supplied by the calling application.

readme.txt
: a text file that describes the download package.

doc
: the subdirectory that contains the documentation for the SQL library (in HTML). The top-level page for this document is index.html.

samples
: the subdirectory that contains sample programs. A README file for the sample programs is also located in this subdirectory.

Package Contents for Windows

The SAS SQL Library for C download package includes the following files and subdirectories:

sasSQL.lib
: an import library that the linker uses to provide the information that is needed to resolve references to the SQL library functions.

sasSQL.dll
: a dynamic link library that contains the SQL library functions.

sasSQL.h
: the main header file for the SQL library. This file includes interface structures, typedefs, function prototypes, and preprocessor symbols

sasSQLrc.h
: a header file that defines preprocessor symbols for the return codes and reason codes that the SQL library functions return.

sampnv.c

sampnv.h
: a working sample implementation of two user-defined functions that are used by the SQL library functions and supplied by the calling application.

readme.txt
: a text file that describes the download package.

doc
: the subdirectory that contains the documentation for the SQL library (in HTML). The top-level page for this document is index.html.

samples
: the subdirectory that contains sample programs. A README file for the sample programs is also located in this subdirectory.

Uses for the SAS SQL Library for C

The SAS SQL Library for C consists of functions for

- configuring data sources
- initializing your client environment
- connecting to a data source
- preparing and sending SQL statements to a SAS server
- disconnecting from a SAS server
- terminating your client environment.

The SQL library also provides flexibility by allowing you to use your own user-defined functions for acquiring and storing data source information.

You can use the SAS SQL Library for C to

- create stand-alone, general purpose query and update interfaces for some or all of the SAS data at your site
- create compact interfaces that are tailored specifically for the data that is associated with a particular SAS application
- connect existing 3GL or 4GL applications to your SAS data
- implement third-party APIs for which SAS Institute does not provide a SAS/SHARE client interface
- write custom versions of programs that conform to open interfaces such as CGI.

What Kinds of Data Can I Access?

You can use the SAS SQL Library for C to access the following types of data:

- SAS data sets (including view descriptors)
- data in DBMSs that have SAS/ACCESS software support for the SQL Procedure Pass-Through facility.

What SAS Concepts Do I Need to Understand?

To use the SAS SQL Library for C, you need to understand three components of SAS software:

- SAS data sets. For more information about SAS data sets, see the *SAS Language: Reference*.
- server libraries. For more information about server libraries, see the *SAS/SHARE Software: Usage and Reference*.
- SAS/SHARE servers.

Using the SQL Library

- Setting up for TCP/IP
- Defining data sources (optional--you do not need to define a data source if your application can identify the SAS/SHARE server explicitly by using the host name or IP address and the service name or port number)
- Steps for creating a client application
- Compiling and linking your programs

TCP/IP Configuration

The SAS SQL Library for C uses TCP/IP to communicate with a SAS/SHARE server. To enable communication between your client application 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 or in an OPTIONS statement when you start the SAS/SHARE SERVER procedure (PROC SERVER):
  - `COMAMID=TCP`
  - `COMAUX1=TCP`
  - `COMAUX2=TCP`.
- Define your servers in the TCP/IP SERVICES file on the server machine and on each client machine.

Note: If your application uses a port number to identify a SAS/SHARE server, then you do not need to modify the SERVICES file on the client machines. See the description of the parameter block to `sasSQLconnect()` for more information.

- For UNIX, the SERVICES file is /etc/services.
- For Windows NT, the SERVICES file is %SYSTEMROOT%/system32/drivers/etc/SERVICES, where %SYSTEMROOT% is the directory where Windows NT is installed.
- For Windows 95, the SERVICES file is SERVICES and is in the directory where Windows 95 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 the SAS SQL Library for C, use the name of a 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 seven characters can include letters, digits, underscores, the dollar sign ($), or the at sign (@). You specify this same server name when you define server information for your data source.

Defining a Data Source

A data source specifies the SAS/SHARE server and server libraries or DBMS that your client application can access. You must define your data sources to your application before your application can access data. A data source definition consists of three entities: data sources, servers, and server libraries. To create a data source definition, you must

- define the data source
- define a SAS/SHARE server
- define server libraries. If your application works only with data on DBMSs, you do not need to define server libraries.

You can use the `sasSQLconfigure()` function to create a data source definition file that your client application can use. The SAS SQL Library for C provides the qsasconfig sample data source definition program, which uses `sasSQLconfigure()` to create a data source definition file. *The data source definition file that is created by qsasconfig can be used to define data sources for any client application that uses the `usergnv()` (GET_NV) and `usersnv()` (SET_NV) sample functions for retrieving and storing data source information.*

You can also create your own data source definition program that uses the `usergnv()`, and `usersnv()` functions or create both your own data source definition program and your own GET_NV (for retrieving information) and SET_NV (for storing data source information) functions.

qsasconfig Dialog

The following dialog is an example qsasconfig dialog that requests data source definition information. It is followed by an explanation of the information that you need to provide for the data source definition. This information is the same regardless of whether you use qsasconfig, `usergnv()`, and `usersnv()` or your own data source definition program and functions.

All but five of the qsasconfig prompts are supplied by the SQL library. The five that are provided by `usergnv()` are

- `Configure data sources for qsas:`
  `(enter c to cancel w/o saving, e to end w/saving)`
- `Enter a Data Source Name to configure:`
- `Enter information for:`
- `Enter a library in data source "my_data_src" to configure:`
- `Do you want to update configuration for server myhost.server1?`

`usergnv()` also outputs the colons (`:`) that are at the ends of the prompts.

qsasconfig prompts the user for data source information and then creates a data source definition file with a default name of `qsas.config`. The definition file can be used by any client application that uses the `usergnv()` and `usersnv()` sample functions for retrieving and storing data source information.The source code for qsasconfig is in the `qsasconfig.c` file.

Note: User input is indicated by **bold** print.

```
1 SystemPrompt> qsasconfig<return>

  Configure data sources for qsas:
   (enter c to cancel w/o saving, e to end w/saving)

2 Enter a Data Source Name to configure: my_data_src<return>

  Enter information for: my_data_src

3 Description: An example data source for me to use<return>
4 SAS/SHARE server name (host.service): myhost.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 myhost.server1
```

```
 8 SAS/SHARE server host IP name (fully qualified) or address [myhost]: server1.unx.
 9 Userid for SAS/SHARE server host: myuser<return>
10 Password for specified userid: mypw<return>
11 SAS/SHARE server user access password: <return>

12 Enter a library in data source "my_data_src" to configure: lib1<return>

   Enter information for: my_data_src LIB1

13 Description: An example library for me to use<return>
14 Library path name: /usr/lib/mystuff<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 "my_data_src" to configure: <return>

18 Enter a Data Source Name to configure: datasrc2<return>

   Enter information for: datasrc2

 3 Description: Another example data source<return>
 4 SAS/SHARE server name (host.service): myhost.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 myhost.server1? <return>

17 Enter a library in data source "datasrc2" to configure: <return>

18 Enter a Data Source Name to configure: <return>
```

A Step-by-Step Explanation of qsasconfig and Data Source Information

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

1. At the system command line prompt, enter `qsasconfig`. 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:

   ```
   qsasconfig -config c:\sasSQL\datasrcs.dsf
   ```

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

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

you may need to press the **Enter** key 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. 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. Note that 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 1024 characters long.

4. At the `SAS/SHARE server name (host.service):` prompt, enter the name of the SAS/SHARE server for this data source. If you specify a one-part name such as `server1`, the SQL library assumes that that is your server name and that your local node (or machine) is your server nodename. If you specify a two-part name such as `myhost.server1`, the SQL library assumes that the first part of the name is the server node and that the second part of the name is the server name.

   To use the TCP/IP access method, you must specify a server name that matches

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

5. 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 the 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.

6. 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.

7. 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 userid and password for the DBMS.

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

9. At the `Userid for SAS/SHARE server host:` prompt, enter a userid for the system that the server runs on. This value is required if the server is running in secured mode; otherwise, it is ignored. The user name is encrypted before it is stored in the data source definition.

10. At the `Password for specified userid:` prompt, enter the password for the userid that you specified in step 9. This is an optional value that you specify if the server is running in secured mode; otherwise, the value is ignored. If the server is running in secured mode and you specify a userid without a password, then you will be prompted for a password when your application connects to the data source. No default value is provided. The password is encrypted before it is stored in the data source definition.

11. 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.

12. 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 "<`*`datasrcname`*`>" to configure:` prompt, enter a libref for the library. <*`datasrcname`*> is the name of the data source that contains the libraries. The libref must reference an existing SAS library that your application wants to access. This value corresponds to the libref value in the SAS LIBNAME statement. Use this libref as the high-level qualifier for the table names in the SQL queries and statements that your application sends to the SAS server. Steps 13 through 16 request additional information about this library.

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

13. At the `Description ():` prompt, enter a description of the library. This value is optional. The description can be up to 1024 characters long.

14. 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.

15. 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.

16. 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.

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

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

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

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

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

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

Steps for Creating a Client Application

1. Define functions for getting and storing data source information.
2. Initialize the client environment.
3. Connect your client to the data source.
4. Process SQL statements.
5. Process reason codes.
6. Process retrieved data.

7. Disconnect from the server.
8. Terminate the client environment.

User-defined Functions

In order for the SQL library functions to access your data, you must provide two additional functions. These two functions are used by the SQL library to store and retrieve data source attributes One function (GET_NV) must be able to get the attribute information, and the other function (SET_NV) must be able to store that information in a form that is retrievable by the first function.

You can design your functions to create a persistent definition of your data source (in a file) or a temporary definition, which must be redefined with each use of the application. Which method you choose depends upon the nature of your application. Although both methods are acceptable, a persistent definition is more appropriate for most applications.

At client initialization time, the addresses of these two functions are passed to the `sasSQLinitialize()` function so that the other SQL library functions can use your functions when they need access to attribute information.

Sample Implementations

The SQL library provides the `sampnv.c` sample file, which contains our sample implementations of the GET_NV and SET_NV functions: `usergnv()` and `usersnv()`. The `usergnv()` function gets attribute information (either from the user or from a data source definition file), and the `usersnv()` function stores attribute information into the definition file. You can either use these two functions in your client application or create functions that are more appropriate for your application.

We also provide qsasconfig, a sample application that uses the `usergnv()` and `usersnv()` functions to create or update a data source definition file. qsasconfig calls `sasSQLconfigure()` which in turn calls `usergnv()` and `usersnv()`.

If you use `usergnv()` and `usersnv()` in your client application, you will need a header file called `dsfile.h`. This file is `#included` by `sampnv.h` and must use the `CONFIG_FILE` preprocessor symbol to define the pathname of the default data source definition file. You must provide this header file in order to use the sample functions. Your application can provide a mechanism for the end user to override the default name.

Tips for Creating Your Own Functions

- Typedefs `GET_NV` and `SET_NV` are provided in the `sasSQL.h` sample file and include prototypes for

your functions.

- Your GET_NV function is called by `sasSQLconfigure()` and `sasSQLconnect()`.

- Your SET_NV function is called by `sasSQLconfigure()`.

- To obtain the name of something that is to be defined, the `sasSQLconfigure()` function calls your GET_NV function.

- In your GET_NV function, you can either read data source information from a file each time you need the information, or you can read the information once and then store it in memory for subsequent access. (The sample implementation `usergnv()` uses the latter approach.) How you implement your GET_NV function depends upon the nature of your client application.

- Just before it returns to its caller, `sasSQLconfigure()` calls the SET_NV function with a NULL second parameter (normally the entity name). If your SET_NV function stores data source information in memory, this call gives you an opportunity to store the data source information into a file (for a persistent data source definition).

  Your GET_NV function can use the SCA_SETUP_CANCEL return code to indicate to `sasSQLconfigure()` that data source information will not be saved and that `sasSQLconfigure()` should not make its pre-termination call to your SET_NV function.

- You can implement the server configuration portion of your data source configuration application in any of three ways:

  1. allow the user to configure their server once and then do not let them do it again
  2. make the user configure their server each time they run the configuration program
  3. allow the user to either keep existing server information or update it. Do this by programming your GET_NV function to first determine whether information already exists for an attribute.

- Your client application can pass information to your GET_NV and SET_NV functions through a handle that is passed to `sasSQLinitialize()`. This handle is then passed to your GET_NV and SET_NV functions as a parameter when they are called by the SQL library functions. This handle can be used by your routines to "remember" information from one call to the next. Your implementation of the GET_NV and SET_NV functions determines the type of information that this handle points to, which could include :

  - the name of the data source definition file

  - a pointer to data source information that is read into memory during your client’s initialization or during a previous call to your GET_NV or SET_NV function

  - flags that modify the behavior of your GET_NV and SET_NV functions.

- Any return codes that your GET_NV and SET_NV functions return are passed (percolated) through `sasSQLconfigure()` or `sasSQLconnect()` and then back to your client application.

- The values that are passed to SET_NV are not necessarily null-terminated. They can contain embedded NULLs.

Processing SQL Statements

After your client application connects to a data source, the application can begin sending SQL statements to the data source.

SQL statements must reference a SAS data file or data view as either *libref.data-file* or *libref.data-view*, where *libref* is defined either by a server administrator for administrator-defined server libraries or in the data source definition for user-defined server libraries.

Before your application can receive the information that an SQL statement returns, the application must establish an environment for receiving the information that the SQL statement retrieves.

- Use `sasSQLprepare()` to submit an SQL SELECT statement and to determine the number of columns that your results set contains. You must know the number of columns in order to allocate a large enough array of structures for the results set.
- Use `sasSQLdescribe()` to allocate the array of structures for the columns in the results set and to obtain information about the results set.
- Use `sasSQLfetch()` to sequentially fetch the rows of data in the results set.
- Use `sasSQLdestroy()` to destroy the resources that are used to prepare, describe, and fetch your results set.
- Use `sasSQLexecute()` to process LIBNAME statements and nonquery SQL statements.

Processing Reason Codes

When your client application receives a nonzero return code from an SQL library function, further information is sometimes available. You can pass the environment handle, connection handle, or statement handle used by the failing function to the `sasSQLreasonCode()` function and retrieve a reason code and associated message, if they are available.

Processing Data

- Values in numeric columns are returned to your client application in double-precision floating-point format (C type `double`). If a SAS format or informat is permanently associated with a column, the name of the format or informat, the length of its name, and the width and number of decimal places (if any) of the format or informat are returned in the `SCAFORMAT` substructure of the `SCACOL` structure for the column. You can use standard UNIX library functions like `printf()` and `sprintf()` to format the floating point values.

  When a SAS date format or informat is associated with a column, `sasSQLdescribe()` indicates that the column probably contains SAS date values by setting the `SCA_DATE_VALUE` flag in the `SCACOL` structure for the column. For a SAS time format or informat, `sasSQLdescribe()` sets the `SCA_TIME_VALUE` flag. For a datetime value, `sasSQLdescribe()` sets both flags.

  The SQL library includes functions to translate a SAS date, time, or datetime value into a form that can be understood by users of your client application.

  - Use the `sasSQLformatDate()` function to translate a SAS date value or the date part of a SAS datetime value.

  - Use the `sasSQLformatTime()` function to translate a SAS time value or the time part of a SAS datetime value.

- To indicate that the value of a numeric column is missing in a particular row, SAS software uses a special notation in its programs, its input data, and its printed output. These missing values are returned to your client application using a special floating point value. To determine if a numeric value is the special SAS missing value, use the `sasSQLmissingValue()` to test the value.

Compiling and Linking Your Programs

Compiling

- Ensure that you `#include` the necessary `.h` files in your `.c` files. Note that the `sasSQLrc.h` file is included at the end of the `sasSQL.h` file. When you write your C program, you do not need to include the `sasSQLrc.h` file in your `.c` file if you already include `sasSQL.h`.

- If you are not using an ANSI C compiler, then define the preprocessor symbol `NO_PROTOTYPES` (the value does not matter). You can define this symbol:

  - in your `.c` file, just before the `#include` of `sasSQL.h`
  - on the compile command if your compiler supports defining preprocessor symbols on the command line.

Linking

UNIX
: For UNIX systems, include either `sasSQL.a` or the shared library file (`sasSQL.sl`, `sasSQL.o`, or `sasSQL.so`) in your link command when you build your executable file. You must also include the `-lM` option to make the standard UNIX math library functions available to the SQL library functions.

  When you run an executable file that is linked with the shared library file, ensure that the shared library file is available for the executable file to use.

Windows
: For Windows 95 and Windows NT, include the import library file (`sasSQL.lib`) in your link command when you build your executable file. When you run the executable file, ensure that the dynamic link library file (`sasSQL.dll`) is available for the executable file to use.

List of Functions

The following table describes the functions (including two sample user-defined functions) that are provided by the SAS SQL Library for C. Interface structures and prototypes for these functions are defined in `sasSQL.h`. For more detail about a function, select the function name.

Function Name	Description
`sasSQLconfigure()`	defines data sources
`sasSQLconnect()`	connects the client application to the SAS/SHARE server and defines any required libraries or connects to a DBMS
`sasSQLdescribe()`	describes the column information of the requested data
`sasSQLdestroy()`	destroys the query resources
`sasSQLdisconnect()`	disconnects the client application from the SAS/SHARE server
`sasSQLexecute()`	processes LIBNAME statements and non-SELECT (non-query) SQL statements
`sasSQLfetch()`	retrieves the data in the results set
`sasSQLformatDate()`	translates a SAS date or datetime value
`sasSQLformatTime()`	translates a SAS time or datetime value
`sasSQLinitialize()`	initializes the client environment
`sasSQLmissingValue()`	indicates if an input value is a missing value
`sasSQLprepare()`	prepares SQL statements
`sasSQLreasonCode()`	returns reason codes
`sasSQLterminate()`	terminates the client environment
`usergnv()`	is a sample user-defined function for retrieving entity attributes
`usersnv()`	is a sample user-defined function for storing entity attributes

---

sasSQLconfigure()

Purpose

Use this function to acquire data source information from a user and put the information into a format that your client application can access.

Syntax

```
int sasSQLconfigure(SCAenv env_handle);
```

Parameters

env_handle *(input)*
the handle for the client application environment

Return Values

- SCA_OK - successful completion
- SCA_ERROR_PARM - the environment is not valid
- SCA_SETUP_CANCEL - the data source definition process was canceled by the user

Important Tips

- This function uses the two user-defined functions for getting and storing data source information. In order for `sasSQLconfigure()` to be able to access these two functions, you must first use the `sasSQLinitialize()` function to point to these functions.

  For more information about how `sasSQLconfigure()` interacts with your user-defined GET_NV function, see What does sasSQLconfigure() expect from the GET_NV function?.

- Pass the environment block pointer that is allocated by `sasSQLinitialize()` as the argument for `sasSQLconfigure()`.

Related Functions

- `sasSQLinitialize()`
- user-defined functions

Example

```
#include "sasSQL.h"

   int              rc;
   SCAenv           env_handle;
   struct my_handle NV_handle;
   rc = sasSQLinitialize(&env_handle, (void *)&NV_handle, usergnv, usersnv);
   rc = sasSQLconfigure(env_handle);
```

What does sasSQLconfigure() expect from the GET_NV function?

To help you better understand what the `sasSQLconfigure()` function expects from your GET_NV function, we provide the following sequence of exchanges that occur between the `sasSQLconfigure()` function and the GET_NV function. `sasSQLconfigure()` calls the GET_NV function several times.

Get Data Source Name

The first time `sasSQLconfigure()` calls GET_NV, it wants GET_NV to get the name of the data source. `sasSQLconfigure()` passes

1. the string "`Data Source Name`" as the second parameter

2. a single SCAattrNV structure that contains a NULL name element as the third parameter
3. a value of 1 as the fourth parameter
4. a value of 0 as the fifth parameter (promptable/public).

sasSQLconfigure() does not provide a user prompt--GET_NV must create its own prompt to get information from the user. Our sample GET_NV function, usergnv() provides the prompt "Enter *<variable>* to configure" and replaces *<variable>* with the string that is passed in in the second parameter (in this case, "Data Source Name"). The completed prompt that the user sees is "Enter a Data Source Name to configure". (See the sampnv.c sample file for an example implementation of the appropriate logic.)

After the data source name is passed back to sasSQLconfigure(), sasSQLconfigure() checks to ensure that all the characters in the data source name are allowed. If an unacceptable character is found, sasSQLconfigure() ends with a return code of SCA_ERROR_CHARS (9).

Get Data Source Attribute Values

Next, sasSQLconfigure() calls GET_NV and uses the data source name that it received from the first call to get the values for the data source's attributes. sasSQLconfigure() calls GET_NV twice, once for private (not promptable) attributes and once for public (promptable) attributes. The calls get the values for all the data source attributes. GET_NV can use either the names or descriptions of the public attributes as prompts for those values. sasSQLconfigure() passes

1. the name of the data source as the second parameter
2. a SCAattrNV structure that contains the names and descriptions of the attributes as the third parameter
3. the number of attributes as the fourth parameter
4. a value of 0 or 1 as the fifth parameter (0=public, 1=private).

For the private attributes, GET_NV should check to see if these values are already in existence in an existing data source definition. GET_NV should not prompt the user for the values of private attributes.

Check for Already Defined Server

At this point, the GET_NV function checks to see if the server needs to be defined. If it is already defined, then GET_NV finds the attribute sasSQLconfigure() asks for.

GET_NV can make a special check for the sasSQLconfigure() call that asks for the attribute. When GET_NV detects the call, it can ask the user whether server information needs to be updated.

sasSQLconfigure() passes

1. the string "Servers" as the second parameter.
2. for the 3rd parameter, a SCAattrNV structure that contains the name of the server (which GET_NV received when it got the values for the data source attributes) as both the name and description of the first structure.
3. a value of 1 as the fourth parameter.
4. a value of 1 as the fifth parameter (not promptable/private).

Check for Already Defined Server Address

If the server is not defined or if the user wants to update the information, `sasSQLconfigure()` first checks to see if a server address is already defined for the server that is specified. If the server address is not defined, then `sasSQLconfigure()` saves the first part of the server name (the part before the period, if there is a period) and uses it as the default value for the server's address. If the server address is already defined, then the previously specified server address is used as the default. `sasSQLconfigure()` passes

1. the string "`Server`" plus the name of the server as the second parameter (for example, "`Server host1.unx.sas.com`")
2. a `SCAattrNV` structure that contains the name and description of the server address attribute as the third parameter
3. a value of 1 as the fourth parameter
4. a value of 1 as the fifth parameter (not promptable/private).

Get Server Attribute Values

Next, `sasSQLconfigure()` gets the values of all server attributes. GET_NV can use either the names or descriptions of the attributes as prompts for those values. `sasSQLconfigure()` passes

1. the string "`Server`" plus the name of the server as the second parameter (for example, "`Server host1.unx.sas.com`")
2. a `SCAattrNV` structure that contains the names and descriptions of the server attributes as the third parameter
3. the number of attributes as the fourth parameter
4. a value of 0 as the fifth parameter (promptable/public).

Get Library Name

Next, `sasSQLconfigure()` needs to get a library to configure. It calls GET_NV to get a library name. `sasSQLconfigure()` passes

1. for the second parameter, the string "`library in data source <datasrcname>`", where *`<datasrcname>`* is the name of the data source
2. a single `SCAattrNV` structure that contains a NULL name element as the third parameter
3. a value of 1 as the fourth parameter
4. a value of 0 as the fifth parameter (promptable/public).

`sasSQLconfigure()` does not provide a user prompt--GET_NV must create its own prompt to get information from the user. Our sample GET_NV function, `usergnv()`, provides the prompt "`Enter a <variable> to configure`" and replaces *`<variable>`* with the string that is passed in the second parameter (in this case, "`library in data source <datasrcname>`". See the `sampnv.c` sample file for an example implementation of the appropriate logic.).

Get Library Attribute Values

Next, `sasSQLconfigure()` calls GET_NV to get the values of all library attributes. GET_NV can use either the names or descriptions of the public attributes as prompts for the values. `sasSQLconfigure()` passes

1. the string "*<datasrcname> <libraryname>*" as the second parameter (for example, "`mydatsrc MYLIB`"). Note that the library name is converted to all uppercase.
2. for the third parameter, a `SCAattrNV` structure that contains the names and descriptions of the library attributes.
3. the number of attributes as the fourth parameter.
4. a value of 0 as the fifth parameter (promptable/public).

---

sasSQLconnect()

Purpose

Use this function to connect your application to a data source. Connecting to a data source includes using the information that is specified in the data source to

- connect your application to a SAS/SHARE server
- define server libraries to the SAS/SHARE server or connect to a DBMS through the SAS/SHARE server.

Syntax

```
int sasSQLconnect(SCAenv    env_handle,
                  SCAcon   *conn_handle,
                  char     *app_name,
                  pSCAconP  parm_block);
```

Parameters

env_handle *(input)*
the handle for the client application environment.

conn_handle *(output)*
the handle for the connection between the client application and the SAS/SHARE server.

app_name *(input)*
a `char *` to the name of the client application. The name of the application can have a maximum of eight characters and is a null-terminated string. The application name is used in messages that are written to the SAS/SHARE server's log.

parm_block *(input)*
a pointer to a parameter block that holds the following values:

- a `char *` to the data source name (`ds`). You must set this pointer to NULL if you do not use a data source name. The data source name must be a null-terminated string. If you specify a

data source name, you do not need to specify the server's host IP name or address and the service name or port number (and depending on the application you are creating, you also may not need to specify a userid, password, or server access password).

- a `char *` to the SAS/SHARE server's host IP name or address (`serv_host`). You must set this pointer to NULL if you do not specify a host IP name or address. The host IP name or address must be a null-terminated string.

  Note: This value is required if you do not specify a data source name. If you do specify a data source name, the value that `serv_host` points to overrides the IP name or address in the data source definition.

- a `char *` to the SAS/SHARE server's service name or port number (`serv_port`). You must set this pointer to NULL if you do not specify a service name or port number. The service name or port number must be a null-terminated string.

  Note: This value is required if you do not specify a data source name. If you do specify a data source name, the value that `serv_port` points to overrides the service name or port number in the data source definition.

- a `char *` to the SAS/SHARE server's host userid (`userid`). You must set this pointer to NULL if you do not specify a host userid. This value is required if your server is running in secured mode and you do not specify a data source name. If you do specify a data source name, the value that `userid` points to overrides the userid in the data source definition. The host userid must be a null-terminated string.

- a `char *` to the SAS/SHARE server's host password (`pw`). You must set this pointer to NULL if you do not specify a host password. This value is required if your server is running in secured mode and you do not specify a data source name. If you do specify a data source name, the value that `pw` points to overrides the password in the data source definition. The host password must be a null-terminated string.

- a `char *` to the SAS/SHARE server's access password (`sapw`). You must set this pointer to NULL if you do not specify an access password. The access password must be a null-terminated string, and it 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.

- the number of destroys to defer (`defer_destroys`). This parameter can improve performance by minimizing the number of requests that are made to the server. The `sasSQLdestroy()` function can collect up to 64 destroy requests before actually sending the server a request to release server resources (client-side resources are released each time `sasSQLdestroy()` is called). Any deferred requests are sent to the server when one of the other functions sends a request to the server or when the number of deferred requests reaches

the value that is specified for this parameter.

Use one of the following values to specify the number of requests to defer:

- `-1` specifies the maximum of 64
- `0` specifies no deferrals
- `n` where `n` specifies some number from 1 to 64. If a value higher than 64 is specified, the SQL library resets the value to 64.

Note: If you are writing an interactive application, you may not want to defer destroy requests. This is so tables will not be left open while waiting for a user to decide upon the next action. With non-interactive applications, requests are sent in rapid sequence and ensure that the server resources are cleaned up quickly.

- the number of rows to request from the server for a single fetch request (`bufrecs`). Values can range from 1 to 32,000. If you specify a value outside of that range, then the following rules apply:

  - If you specify 0 or a negative number, then `bufrecs` defaults to 100.

  - If you specify a value larger than 32,000, then `bufrecs` defaults to 32,000.

  If the number of rows that are requested is too large to fit into one of the server's transmission buffers, then the number is automatically reduced by the server. The `bufrecs` parameter is ignored by Version 6 SAS/SHARE servers, which always return 10 rows in response to a fetch request.

  Note that the buffering of multiple rows for transmission is handled by the server and the SQL library; you must still make a separate call to `sasSQLfetch()` for each row in the results set.

Return Values

- SCA_OK - successful completion
- SCA_ERROR_MEM - unable to allocate memory
- SCA_ERROR_PARM - the value that was passed to the `env_handle` parameter is not valid
- SCA_ERROR_SERVER - the server is not defined
- SCA_ERROR_COMM - access method error
- SCA_WARN_NOEXEC - one of the libraries included in the data source could not be defined to the server.

Important Tips

- Pass the environment block pointer that is allocated by `sasSQLinitialize()` as the first argument for `sasSQLconnect()`.

- The `sasSQLconnect()` function uses the user-defined get function to obtain data source information.

- Use the `sasSQLdisconnect()` function to disconnect the client when the client no longer needs to be connected to the server.

Related Functions

- `sasSQLdestroy()`
- `sasSQLdisconnect()`
- user-defined get function

Example

```
#include "sasSQL.h"

   int     rc;
   SCAenv  env_handle;
   SCAcon  conn_handle;
   SCAconP  parm_block;


   parm_block.ds             = "sales data";
   parm_block.serv_host      = "test1.acme.com";   /* override the host name in the
   parm_block.serv_port      = NULL;
   parm_block.userid         = NULL;
   parm_block.pw             = NULL;
   parm_block.sapw           = NULL;
   parm_block.defer_destroys = -1;
   parm_block.bufrecs        = 0;

   rc = sasSQLconnect(env_handle,     /* environment handle from sasSQLinitialize()
                      &conn_handle,   /* get back a connection handle */
                      "Qsales",       /* name of this application */
                      &parm_block);
```

---

sasSQLdescribe()

Purpose

Use this function to allocate an array of structures for the columns in the results set and to fill the array with information about the results set.

Syntax

```
int sasSQLdescribe(SCAstmt  stmt_handle,
                   SCAcol  *columns);
```

Parameters

stmt_handle *(input)*

the handle for the prepared SQL statement

column *(update)*
a pointer to an array of column descriptors that are updated with information about the individual columns

Return Values

- SCA_OK - successful completion
- SCA_ERROR_MEM - unable to allocate memory
- SCA_ERROR_PARM - the value that was passed to the `stmt_handle` parameter is not valid
- SCA_ERROR_SEQ - the SAS SQL Library call is out of order
- SCA_ERROR_COMM - access method error
- SCA_ERROR_DESCERR - `sasSQLdescribe()` error occurred before current function call

Related Functions

- `sasSQLprepare()`

Example

```
#include "sasSQL.h"

   int     rc;
   SCAstmt stmt_handle;
   SCAcol  columns;

   rc = sasSQLdescribe(stmt_handle, &columns);
```

---

sasSQLdestroy()

Purpose

Use this function to release resources that `sasSQLprepare()`, `sasSQLdescribe()`, and `sasSQLfetch()` allocated.

Syntax

```
int sasSQLdestroy(SCAstmt stmt_handle);
```

Parameters

stmt_handle *(input)*
the handle for the prepared SQL statement

Return Values

- SCA_OK - successful completion
- SCA_ERROR_MEM - unable to allocate memory
- SCA_ERROR_PARM - the value that was passed to the `stmt_handle` parameter is not valid
- SCA_ERROR_SEQ - the SAS SQL Library call is out of order
- SCA_ERROR_COMM - access method error

Important Tips

- Release your resources only after you finish fetching the rows of data that you need.

- The SQL library enables you to defer sending destroy requests to the server. Up to a specified number (set by `sasSQLconnect()`) of requests are deferred and sent either when that number is reached or when the next non-destroy function call sends a request to the server.

  Note: If you are writing an interactive application, you may not want to defer destroy requests. This is so tables will not be left open while waiting for a user to decide upon the next action. With non-interactive applications, requests are sent in rapid sequence and ensure that the server resources are cleaned up quickly.

- If you do not call `sasSQLdestroy()` before you call `sasSQLdisconnect()`, then `sasSQLdisconnect()` automatically calls `sasSQLdestroy()`.

Related Functions

- `sasSQLconnect()`
- `sasSQLdescribe()`
- `sasSQLfetch()`
- `sasSQLprepare()`

Example

```
#include "sasSQL.h"

   int     rc;
   SCAstmt stmt_handle;
   rc = sasSQLdestroy(stmt_handle);
```

---

sasSQLdisconnect()

Purpose

Use this function to disconnect a client application from its SAS/SHARE server.

Syntax

```
int sasSQLdisconnect(SCAcon conn_handle);
```

Parameters

conn_handle *(output)*
the handle for the connection between the client application and the SAS/SHARE server

Return Values

- SCA_OK - successful completion
- SCA_ERROR_PARM - the value passed to the `conn_handle` parameter is not valid
- SCA_ERROR_COMM - access method error

Important Tips

Use this function to clean up any resources that support the connection created by `sasSQLconnect()`.

Related Functions

- `sasSQLconnect()`

Example

```
#include "sasSQL.h"

   int     rc;
   SCAcon  conn_handle;
   rc = sasSQLdisconnect(conn_handle);
```

---

sasSQLexecute()

Purpose

Use this function to send LIBNAME statements and non-SELECT (non-query) SQL statements to the SAS/SHARE server.

Syntax

```
int sasSQLexecute(SCAcon conn_handle,
                  char  *statement,
                  int    statement_len,
                  int   *rows);
```

Parameters

conn_handle *(input)*
the handle for the connection between the client application and the SAS/SHARE server.
statement *(input)*
a `char *` to the statement that is to be processed. This is a required value.
statement_len *(input)*
the length of the statement.
rows *(output)*
an `int *` to the number of rows that are updated, inserted, or deleted.

Return Values

- SCA_OK - successful completion
- SCA_ERROR_API - an SQL (native) view engine API error was received
- SCA_ERROR_MEM - unable to allocate memory
- SCA_ERROR_PARM - the value passed to the `conn_handle` parameter is not valid
- SCA_ERROR_COMM - access method error
- SCA_WARN_NOEXEC - a non-successful return code (warning or error) was returned by the server

Related Functions

- `sasSQLconnect()`

Example

```
#include "sasSQL.h"

   int     statement_len,
           rc,
           rows;
   ptr     statement;
   SCAcon  conn_handle;

   rc = sasSQLexecute(conn_handle,
                      statement,
                      statement_len,
                      &rows);
```

---

sasSQLfetch()

Purpose

Use this function to retrieve the next row in the results set. Rows are retrieved sequentially, one at a time.

Syntax

```
int sasSQLfetch(SCAstmt stmt_handle);
```

Parameters

stmt_handle *(input)*
the handle for the prepared SQL statement

Return Values

- SCA_OK - successful completion
- SCA_CHAR_TRUNC - the value of one or more character columns was truncated
- SCA_WARN_EOF - end of file reached
- SCA_ERROR_PARM - the value that was passed to the `stmt_handle` parameter is not valid
- SCA_ERROR_SEQ - the SAS SQL Library call is out of order
- SCA_ERROR_COMM - access method error
- SCA_ERROR_DESCERR - `sasSQLdescribe()` error occurred before current function call

Important Tips

- After a successful fetch, the *data* member of each column's `SCACOL` structure contains a pointer to that column's data value.

- If the value of one or more character columns is truncated when the server retrieves it from an external DBMS, then `sasSQLfetch()` returns the SCA_CHAR_TRUNC return code. The `trunc` flag is turned on in the `SCACOL` structure for each column whose value was truncated.

Related Functions

- `sasSQLprepare()`

Example

```
#include "sasSQL.h"

   int     rc;
   SCAstmt stmt_handle;

   rc = sasSQLfetch(stmt_handle);
```

---

sasSQLformatDate()

Purpose

Use this function to translate a SAS date or datetime value. The month, day, year, quarter, Julian day,

and weekday are returned and stored in a SCAFDATE structure.

Syntax

```
int sasSQLformatDate(double            SASdate,
                     struct SCAFORMAT *fmt,
                     SCAfdate          date);
```

Parameters

SASdate *(input)*
: a SAS date or datetime value

fmt *(input)*
: a pointer to the SCAFORMAT in the SCACOL structure for the date column

date *(output)*
: a pointer to the SCAFDATE structure that contains the translated date information

Return Values

- SCA_OK - successful completion
- SCA_ERROR_PARM - the value that was passed to the SASdate parameter is not a valid SAS date

Related Functions

- sasSQLdescribe()
- sasSQLformatTime()

Example

```
#include "sasSQL.h"

   double          *SASdate;
   int              rc;
   SCAcol           cur_col;
   struct SCAFDATE  date;

   if (cur_col->dtval & SCA_DATE_VALUE)
   {
     SASdate = (double *) cur_col->data;
     rc = sasSQLformatDate(*SASdate, &cur_col->format, &date);
   }
```

---

sasSQLformatTime()

Purpose

Use this function to translate a SAS time or datetime value. Hours, minutes, and seconds are returned and stored in a SCAFTIME structure.

Syntax

```
void sasSQLformatTime(double            SAStime,
                      struct SCAFORMAT *fmt,
                      SCAftime          time);
```

Parameters

SAStime *(input)*
: a SAS time or datetime value

fmt *(input)*
: a pointer to the SCAFORMAT in the SCACOL structure for the time column

time *(output)*
: a pointer to the SCAFTIME structure that contains the translated time information

Return Values

- SCA_OK - successful completion
- SCA_ERROR_PARM - the value that was passed to the SAStime parameter is not a valid SAS time

Related Functions

- sasSQLdescribe()
- sasSQLformatDate()

Example

```
#include "sasSQL.h"

   double           *SAStime;
   int               rc;
   SCAcol            cur_col;
   struct  SCAFTIME  time;

   if (cur_col->dtval & SCA_TIME_VALUE)
   {
     SAStime = (double *) cur_col->data;
     rc = sasSQLformatTime(*SAStime, &cur_col->format, &time);
   }
```

---

sasSQLinitialize()

Purpose

Use this function to initialize a client environment. The `sasSQLinitialize()` function allocates and initializes an environment block and then stores its address in a pointer of type `SCAenv`.

Syntax

```
int sasSQLinitialize(SCAenv *env_handle,
                     void   *NV_handle,
                     GET_NV  usergnv,
                     SET_NV  usersnv);
```

Parameters

env_handle *(output)*
: the handle for the client application environment.

NV_handle *(input)*
: the handle that points to the data source information that the user-defined functions processes. This handle is passed on to the user-defined functions.

usergnv *(input)*
: the pointer to the user-defined function that gets the value associated with a name.

usersnv *(input)*
: the pointer to the user-defined function function that sets a value associated with a name.

Return Values

- SCA_OK - successful completion
- SCA_ERROR_MEM - unable to allocate memory
- SCA_ERROR_PARM - a parameter that is not valid was received

Important Tips

- Your application must call `sasSQLinitialize()` before the application can use the rest of the SQL library.
- Call this function only once per client application.
- Pass the environment block pointer that is allocated by `sasSQLinitialize()` as the argument to `sasSQLconfigure()` and `sasSQLterminate()` and as the first argument to `sasSQLconnect()`.
- When a client terminates, call the `sasSQLterminate()` function to remove the client environment.

Related Functions

- `sasSQLterminate()`
- user-defined functions()

Example

```
#include "sasSQL.h"

   int    rc;
   SCAenv env_handle;
   void  *NV_handle;

   rc = sasSQLinitialize(&env_handle, NV_handle, usergnv, usersnv);
```

---

sasSQLmissingValue()

Purpose

Use this function to determine whether the value of a numeric column in the current row is a missing value.

Syntax

```
int sasSQLmissingValue(double double_val);
```

Parameters

double_val *(input)*
the double-precision floating-point value that you want to test

Return Values

- Nonzero - the value is a missing value
- Zero (0)- the value is not a missing value

Example

```
#include "sasSQL.h"

   int    is_missing;
   SCAcol cur_col;

   is_missing = sasSQLmissingValue(*(double *)cur_col->data);
```

---

sasSQLprepare()

Purpose

Use this function to submit an SQL SELECT statement to the SAS/SHARE server and to determine the number of columns that your results set contains.

Syntax

```
int sasSQLprepare(SCAcon   conn_handle,
                  char    *prepare,
                  int      prepare_len,
                  SCAstmt *stmt_handle,
                  int     *columns);
```

Parameters

conn_handle *(input)*
: the handle for the connection between the client application and the SAS/SHARE server.

prepare *(input)*
: a `char *` to the SQL statement that is to be prepared. This is a required value.

prepare_len *(input)*
: the length of the statement.

stmt_handle *(output)*
: the handle for the prepared SQL statement.

columns *(output)*
: an `int *` to the number of columns that are returned.

Return Values

- SCA_OK - successful completion
- SCA_ERROR_MEM - unable to allocate memory
- SCA_ERROR_PARM - the value passed to the `conn_handle` parameter is not valid
- SCA_ERROR_SEQ - the SAS SQL Library call is out of order
- SCA_ERROR_COMM - access method error

Related Functions

- `sasSQLconnect()`
- `sasSQLdescribe()`

Example

```
#include "sasSQL.h"

   int     prepare_len,
           columns,
           rc;
   ptr     prepare;
   SCAcon  conn_handle;
   SCAstmt stmt_handle;

   rc = sasSQLprepare(conn_handle,
                      prepare,
                      prepare_len,
```

```
                    &stmt_handle,
                    &columns);
```

---

sasSQLreasonCode()

Purpose

Use this function to obtain the reason code and error text that are associated with the last error return code, if they are available.

Syntax

```
int sasSQLreasonCode(void  *handle,
                     int   *reason_code,
                     char **error_text);
```

Parameters

handle *(input)*
: the handle that was passed to the SQL library function that returned an error

reason_code *(output)*
: an `int *` to the reason code that is associated with the error

error_text *(output)*
: a `char *` to the error text that corresponds to the reason code

Example

```
#include "sasSQL.h"

   char   *error_text;
   int     rc,
           reason_code;
   SCAcon  conn_handle;

   rc = sasSQLreasonCode(conn_handle, &reason_code, &error_text);
```

---

sasSQLterminate()

Purpose

Use this function to terminate the client environment.

Syntax

```
int sasSQLterminate(SCAenv env_handle);
```

Parameters

env_handle *(input)*
the handle for the client application environment

Return Values

- SCA_OK - successful completion
- SCA_WARN_DISC - `sasSQLterminate()` disconnected active connections

Important Tips

- Pass the environment block pointer that is allocated by `sasSQLinitialize()` as the argument for `sasSQLterminate()`.
- If connections are active when `sasSQLterminate()` is called, `sasSQLterminate()` disconnects those connections and returns an SCA_WARN_DISC return code.

Related Functions

- `sasSQLinitialize()`

Example

```
#include "sasSQL.h"

   int    rc;
   SCAenv env_handle;

   rc = sasSQLterminate(env_handle);
```

---

usergnv()

Purpose

This sample implementation of the GET_NV user-defined function retrieves values for a specified list of entity attributes.

Syntax

```
int usergnv(void      *NV_handle,
```

```
char      *name,
SCAattrNV *attr_nv,
int       *num_attrs,
int        privy);
```

Parameters

NV_handle *(input)*
: the handle that points to the data source information. This is the same handle that is passed to `sasSQLinitialize()`. The handle structure that `usergnv()` uses is the `SAMP_HNDL` structure that is defined in sample file `sampnv.h`.

name *(input)*
: a `char *` to the name of an entity. This is a null-terminated string.

attr_nv *(input)*
: an `SCAattrNV *` to an array of lengths and pointers that point to the attribute name, description, and value sets.

num_attrs *(input)*
: an `int *` to the number of elements in the attr_nv array.

privy *(input)*
: a true (1) or false (0) value that indicates whether the attribute name and value pairs are private or public. In a GET_NV function, a true (1) value indicates that the function should not prompt for the value of a requested attribute if the value is not already known. In `usergnv()`, `privy` is used in conjunction with the NO_PROMPT, PROMPT_ALWAYS, and PROMPT_NOT_KNOWN flags in the handle structure to determine whether to prompt the user. The flags in the handle are set by the client application.

Return Values

- SCA_OK - successful completion
- SCA_SETUP_END - end data source definition setup and save the known values
- SCA_SETUP_CANCEL - indicate to `sasSQLconfigure()` that it should not make its pre-termination call to `usersnv()`.

Important Tips

This function is passed to the `sasSQLinitialize()` function in the qsas sample program.

Related Functions

- `sasSQLconfigure()`
- `sasSQLinitialize()`
- `usersnv()`

---

usersnv()

Purpose

This sample implementation of the SET_NV user-defined function stores the value for a specified entity attribute.

Syntax

```
int usersnv(void *NV_handle,
            char *name,
            char *attr,
            char *value,
            int   val_len);
```

Parameters

NV_handle *(input)*
: the handle that points to the data source information. This is the same handle that is passed to `sasSQLinitialize()`.

name *(input)*
: a `char *` to the name of an entity. The name is a null-terminated string.

attr *(input)*
: a `char *` to the name of an attribute that corresponds to the entity. The name of the attribute is a null-terminated string.

value *(input)*
: a `char *` to the value of the attribute.

val_len *(input)*
: the length of the name value.

Return Values

- SCA_OK - successful completion
- !SCA_OK - unsuccessful completion

Important Tips

This function is passed to the `sasSQLinitialize()` function in the qsas sample program.

Related Functions

- `sasSQLconfigure()`
- `sasSQLinitialize()`
- `usergnv()`

Sample Code

The SAS SQL Library for C provides source files for two complete sample applications. The `samples` subdirectory contains a subdirectory for each sample program.

- The qsas program takes an SQL query from the command line, sends the query to a SAS/SHARE server, and displays the results set.
- The qsasconfig program defines the data sources for the qsas program and creates a data source definition file.

For the UNIX operating system, a makefile and a man page are included in the subdirectory for each program. To build one of the sample programs, change to the appropriate subdirectory and enter `make` at the command line prompt.

---

qsas

The qsas program consists of the following files. To use qsas, see Instructions for Using qsas. To see the source code, select the file that you want to see.

File Name	Description
`qsas.c`	the main qsas program.
`qpsas.c`	the functions that process the SQL statements for qsas.
`sampnv.c`	sample source code that gets and sets data source information. This file contains the `usergnv()` and `usersnv()` functions.
`dsfile.h`	an include file used by `sampnv.h`. It contains the name of the default data source definition file for qsas.
`qpsas.h`	an include file used by `qsas.c` and `qpsas.c`.
`qsas.h`	an include file used by `qsas.c`.
`sampnv.h`	an include file used by `sampnv.c`.
`sasSQL.h`	an include file used by `sampnv.h`. It contains the SAS SQL Library for C function prototypes.
`sasSQLrc.h`	an include file used by `sasSQL.h`. It contains the SAS SQL Library for C return codes.

qsasconfig

The qsasconfig program consists of the following files. To use qsasconfig, see Instructions for Using qsasconfig. To view the dialog that is generated by the qsasconfig program, see qsasconfig Dialog. To see the source code, select the file that you want to see.

File Name	Description
`qsasconfig.c`	the main qsasconfig program. The qsasconfig program is a stand-alone program and can be run independently from the rest of the qsas program.
`sampnv.c`	sample source code that gets and sets data source information. This file contains the `usergnv()` and `usersnv()` functions.
`dsfile.h`	an include file used by `sampnv.h`. It contains the name of the default data source definition file for qsasconfig.
`sampnv.h`	an include file used by `sampnv.c`.
`sasSQL.h`	an include file used by `sampnv.h`. It contains the SAS SQL Library for C function prototypes.
`sasSQLrc.h`	an include file used by `sasSQL.h`. It contains the SAS SQL Library for C return codes.

Instructions for Using qsas

The qsas program is a line-mode program that sends an SQL query to a SAS server and displays the results set.

Note: If you use a Windows operating system, you must run this program from an MS-DOS command prompt window.

Select one of the following topics for information about using the qsas program:

- Syntax
- Parameters
- SQL query syntax
- Output
- Metadata

Syntax

```
qsas data-source | -server host:port
     ["SQL-query"] [-config path-name] [-nohead]
     [-noid] [-nowhitesp] [-raw] [-htmlesc] [-silent]
     [-userid userid] [-password password]
```

Parameters

data-source

the name of a pre-defined qsas data source. You must specify a data source if you do not specify a value for the `-server` parameter; do not specify both parameters. A data source definition specifies a SAS server and the SAS libraries that are available through it. Libraries defined to the server by an administrator are automatically available and are not included in the data source definition.

`-server` ***`host:port`***
the SAS/SHARE server to connect to. You must specify a server if you do not specify a value for the *`data-source`* parameter; do not specify both parameters.

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.

The following are examples of valid values:

```
klondike.acme.com:5228

penn.sylvania:6500

yukon.sasshr1

yukon:sasshr1
```

`SQL-query` (*optional*)
an SQL query that is to be processed by the SAS server. If the query is omitted from the command line, then qsas reads it from `stdin`. Specify SAS data set names using the libref defined in the data source definition. For administrator-defined server libraries not included in the data source definition, use the server's libref.

`-config` ***`path-name`*** (*optional*)
an alternate qsas data source definition pathname. Override of the definition file can be disallowed by the qsas installer or administrator. `-c` is an alias for `-config`.

`-nohead` (*optional*)
an option that tells qsas to not output the header line that contains the names of the columns.

`-noid` (*optional*)
an option that tells qsas to not output the "OBS" column that contains the observation numbers.

`-nowhitesp` (*optional*)
an option that tells qsas to not output the blank lines that precede and follow the query results.

`-raw` (*optional*)
a synonym for `-nohead -noid -nowhitesp .`

`-htmlesc` (*optional*)
escape characters with special meaning in HTML.

`-silent` (*optional*)
an option that tells qsas to not output error messages (give a return code only).

`-userid` ***`userid`*** (*optional*)
a userid for the machine on which the SAS server is running. This option overrides any server userid that is stored in the qsas data source definition file. This option is ignored by the server if the server is not running in a secured mode. `-u` is an alias for `-userid`.

`-password` ***`password`*** (*optional*)
the password for the userid specified by `-userid`. This option overrides any user password that is stored in the qsas data source definition file. This option is ignored by the server if the server is not running a secured mode. `-p` is an alias for `-password`.

SQL Query Syntax

The qsas program expects a complete SQL query. The syntax for such a query is shown below. Note that case is significant only for character values in the query; keywords are shown in caps below for illustrative purposes.

```
SELECT column [, column]...
    FROM table | view [, table | view]...
    [WHERE expression]
    [GROUP BY column [, column]... ]
    [HAVING expression]
    [ORDER BY column [, column]...]
```

See the *SAS Procedures Guide, Version 7, First Edition* for further details on specifying SQL queries for SAS data.

Output

The qsas program writes to `stdout` the values of all the selected variables for each returned record. Each record is printed on a single line.

- Character variables are printed as they are stored, including trailing blanks.
- Numeric variables are minimally formatted with a maximum of eight significant digits in an eight-byte field. This means that integer values are printed with no decimal, and non-integer values are printed with the minimum number of decimal places required to accurately represent the value (for example, 1, 1.25, 1.375, 1.5).

- Numeric variables with an associated date, time, or datetime format are presumed to contain SAS date, time, or datetime values, respectively, and are formatted as such.

    - Date values are formatted using the DATE7. format (for example, 21SEP95).
    - Time values are formatted using the TIME. format (for example, 16:34:17).
    - Datetime values are formatted with the date expressed in DATE7. format and the time in TIME. format (for example, 16OCT95 09:10:35).

- To override the default formatting for any of the numeric variables, use the SAS PUT function in the SQL query as follows:

```
select put(cost,dollar8.2) as cost, put(date,mmddyy10.)
  as date from ...
```

- To get just the date or time part of a datetime value, use the SAS DATEPART or TIMEPART function in conjunction with the SAS PUT function to format the value appropriately. For example

```
select put(datepart(datetime),mmddyy10.) as date from ...
```

- SQL reserves certain column names for special values. In particular, the column name USER asks for the current userid. If you have a SAS variable that is also named USER, and you want to select it, then you must use the RENAME= data set option to rename it:

```
select ruser as user from foo.bar (rename=(user=ruser)) ...
```

  Otherwise, it will return the current userid instead of the values for your variable.

Metadata

The following sample queries return metadata information that you can use in constructing queries for qsas.

- To list all librefs available through a data source named *sampdata*, including those for administrator-defined server libraries:

```
qsas sampdata "select unique libname from
  dictionary.tables"
```

- To list all data sets available through the *sampdata* data source:

```
qsas sampdata "select libname,memname,memtype from
  dictionary.tables"
```

- To list the variables in a data set available through the *sampdata* data source:

```
qsas sampdata "select name,type,length,format from
  dictionary.columns where libname='DEF' and
  memname='BUGDATA'"
```

```
/*-----------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ----------------------------------------------------------------------
*
*  NAME:       qsas.c
*  DATE:       03 Aug 1995
*  SUPPORT:    *** sample client implementation ***
*  PURPOSE:    Pass a specified query for a specified data source
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*----------------------------------------------------------------------*/

/*----------------------------------------------------------------------*/
/* include stuff for the SAS SQL Library                                */
/*----------------------------------------------------------------------*/
#include "sasSQL.h"


#include "qsas.h"
#include "qpsas.h"
#include <sys/stat.h>


/*----------------------------------------------------------------------*/
/* include stuff for the sample NV routines                             */
/*----------------------------------------------------------------------*/
#include "sampnv.h"


#ifndef FALSE
#define FALSE 0
#endif


int
main(argc, argv)
   int                argc;      /* Count of arguments received      */
   char              *argv[];    /* Array of arguments received      */
{
   char              *error_text;
   char              *query = NULL,
                     *qp = NULL,
                     *read_it = NULL,
                     *c = NULL;
   int                len,
                      remains,
                      rc = SCA_OK,
                      reason,
                      statrc,
                      i;
   struct QP          qpargs;    /* query & print parms              */
   struct stat        dummy;
```

```
SCAconP             conp;
SCAcon              con = NULL;
SCAenv              env;
struct SAMP_HNDL    handle;    /* Our sample's handle             */
static char         name[] = "middle";


/*--------------------------------------------------------------------+
 |  Initialize our head node                                          |
 +--------------------------------------------------------------------*/
handle.head.left     =
handle.head.right    =
handle.head.parent   = NULL;
handle.head.value    = NULL;
handle.head.name     = name;
handle.head.name_len = sizeof(name) - 1;
handle.config        = NO_PROMPT;
handle.config_file   = NULL;
handle.file_loaded   = FALSE;
handle.flags         = 0;


/*--------------------------------------------------------------------+
 |  Initialize the sasSQLconnect() parm block.                        |
 +--------------------------------------------------------------------*/
conp.ds        =
conp.serv_host =
conp.serv_port =
conp.userid    =
conp.pw        =
conp.sapw      = NULL;
conp.defer_destroys = 1;


/*--------------------------------------------------------------------+
 |  Initialize the query & print parm block.                          |
 +--------------------------------------------------------------------*/
 qpargs.nohead    =
 qpargs.noid      =
 qpargs.nowhitesp =
 qpargs.silent    =
 qpargs.htmlesc   =
 qpargs.lines     = 0;


/*--------------------------------------------------------------------+
 |  Parse the command line.                                           |
 +--------------------------------------------------------------------*/
 for (i = 1; i < argc; i++)
 {
     if (strncmp(argv[i],"-c",2) == 0)
     {
         i++;
         statrc = stat(CONFIG_OVERRIDE_FILE,&dummy);
         if (statrc == 0)
             handle.config_file = argv[i];
         else
         {
             if (!qpargs.silent)
                 printf("Config file override not allowed.\n");
             rc = QSAS_NOCONFIGOVERRIDE;
```

```
        goto ret;
    }
}
else if (strncmp(argv[i],"-u",2) == 0)
{
    i++;
    conp.userid = argv[i];
}
else if (strncmp(argv[i],"-p",2) == 0)
{
    i++;
    conp.pw = argv[i];
}
else if (strncmp(argv[i],"-sapw",5) == 0)
{
    i++;
    conp.sapw = argv[i];
}
else if (strncmp(argv[i],"-se",3) == 0)
{
    i++;
    if (c = strchr(argv[i], ':'))         /* =, not == ! */
    {
        *c++ = '\0';
        conp.serv_host = argv[i];
        conp.serv_port = c;
    }
    else if (c = strrchr(argv[i], '.'))   /* =, not == ! */
    {
        *c++ = '\0';
        conp.serv_host = argv[i];
        conp.serv_port = c;
    }
    else
    {
        conp.serv_host = strdup("localhost");
        conp.serv_port = argv[i];
    }
}
else if (strncmp(argv[i],"-noh",4) == 0)
{
    qpargs.nohead = 1;
}
else if (strncmp(argv[i],"-noid",5) == 0)
{
    qpargs.noid = 1;
}
else if (strncmp(argv[i],"-nowhi",6) == 0)
{
    qpargs.nowhitesp = 1;
}
else if (strncmp(argv[i],"-raw",4) == 0)
{
    qpargs.nohead    =
    qpargs.noid      =
    qpargs.nowhitesp = 1;
}
else if (strncmp(argv[i],"-si",3) == 0)
{
    qpargs.silent = 1;
}
```

```
    else if (strncmp(argv[i],"-ht",3) == 0)
    {
        qpargs.htmlesc = 1;
    }
    else
    {
        if (!conp.ds && !conp.serv_host)
            conp.ds = argv[i];
        else if (!query)
            query = argv[i];
        else
            if (!qpargs.silent)
                printf("Extra argument %s ignored.\n", argv[i]);
    }
 }

 if (!conp.ds && !conp.serv_host)
 {
    if (!qpargs.silent)
        printf("Data source name or SAS/SHARE server required.\n");
    rc = QSAS_NODSN;
    goto ret;
 }


/*-------------------------------------------------------------------+
 |  First things first.  Need to initialize our client, then worry |
 |  about connecting to it.                                        |
 |                                                                 |
 |  Pass in our get/set name/value pair routines and the handle    |
 |  used by those routines.                                        |
 +-------------------------------------------------------------------*/
rc = sasSQLinitialize(&env, (void *)&handle, usergnv, usersnv);
if (rc != SCA_OK)
   {
   if (!qpargs.silent)
       printf("scainit: returned rc=%X\n", rc);
   return(rc);
   }


/*-------------------------------------------------------------------+
 |  Connect to the data source passed in.                          |
 +-------------------------------------------------------------------*/
rc = sasSQLconnect(env, &con, "QSAS", &conp);
if (rc != SCA_OK)
   {
   if (!qpargs.silent)
   {
       (void)sasSQLreasonCode(env, &reason, &error_text);
       printf("sasSQLconnect: returned rc=%X, reas=%X, (%s)\n",
              rc, reason, error_text);
   }
   if (rc != SCA_WARN_NOEXEC)
      {
      (void)sasSQLterminate(env);
      return(rc);
      }
   }
```

```
/*---------------------------------------------------------------------+
 |  If no query was specified on the command line, read stdin.         |
 +---------------------------------------------------------------------*/
 if (!query)
 {
     remains = 32767;
     query=malloc(remains);
     if (query)
     {
         qp=query;
         do
         {
             read_it=fgets(qp,remains,stdin);
             len = strlen(qp);
             qp += len-1;
             if (*qp == '\n')
                 *qp = ' ';
             remains -= len;
             if (remains > 1)
                 qp++;
             else
                 read_it = NULL;
         }
         while (read_it);
         qp=query;
     }
     else
     {
         if (!qpargs.silent)
             printf("Input buffer (%d bytes) allocation failed.\n", remains);
         rc = QSAS_NOMEM;
         goto disc;
     }
 }


/*---------------------------------------------------------------------+
 |  White space for looks.                                             |
 +---------------------------------------------------------------------*/
 if (!qpargs.nowhitesp)
     printf("\n");


/*---------------------------------------------------------------------+
 |  Pass the query, fetch the records, and print the values.           |
 +---------------------------------------------------------------------*/
 rc = query_and_print(con,query,&qpargs);

 if (qpargs.lines == 0 && !qpargs.silent)
     printf("No records returned\n");


/*---------------------------------------------------------------------+
 |  White space for looks.                                             |
 +---------------------------------------------------------------------*/
 if (!qpargs.nowhitesp)
     printf("\n");


/*---------------------------------------------------------------------+
 |  If we had to read the query from stdin, free the buffer.           |
```

```
    +-------------------------------------------------------------------*/
    if (qp)
        free((void *)qp);


disc:
   if (con)
      (void)sasSQLdisconnect(con);
   (void)sasSQLterminate(env);

ret:
   return(rc);
}
```

```
/*-------------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ------------------------------------------------------------------------
*
*  NAME:       qpsas.c
*  DATE:       20 Sep 1995
*  SUPPORT:    *** sample client implementation ***
*  PURPOSE:    Pass a specified query, fetch and print results
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*-------------------------------------------------------------------------*/

#include "qpsas.h"

/*-------------------------------------------------------------------------*/
/* include stuff for the SAS SQL Library                                   */
/*-------------------------------------------------------------------------*/
#include "sasSQL.h"

#include <stdlib.h>
#include <stdio.h>
#include <string.h>


void
esc_txt(buffer)
   char *              buffer;
{
   while (*buffer) {
        switch (*buffer) {
                case '<': printf("&lt;");
                          break;
                case '>': printf("&gt;");
                          break;
                case '&': printf("&amp;");
                          break;
```

```
                case '\"': printf("&quot;");
                           break;
                default : putchar(*buffer);
        }
   buffer++;
   }
}



int
query_and_print(con,query,qpa)
   SCAcon              con;
   char *              query;     /* SQL query                        */
   struct QP *         qpa;       /* parm block                       */
{
   char               *api,
                      *buf = NULL,
                      *buf_ptr,
                      *c,
                      *error_text;
   int                 data_records = 0,
                       column_count,
                       drc,
                       i, j,
                       len,
                       rc = SCA_OK,
                       reason;
   SCAcol              columns = NULL,
                       cur_column;
   SCAstmt             stmt = NULL;
   struct SCAFDATE     fmtdate;
   struct SCAFTIME     fmttime;
   static char *       months[] = {"","JAN","FEB","MAR","APR","MAY","JUN",
                                   "JUL","AUG","SEP","OCT","NOV","DEC"};
   void               *sca_handle;


   /*-------------------------------------------------------------------+
    |  Make sure what we have is a query.                               |
    +-------------------------------------------------------------------*/
    c = query + strspn(query," ");
    if (strncasecmp(c,"SELECT",6))
    {
        if (!qpa->silent)
            printf("Query must begin with the keyword 'select'.\n");
        rc = QPSAS_NOQUERY;
        goto ret3;
    }


   /*-------------------------------------------------------------------+
    |  Now prepare and describe the query passed in.                    |
    +-------------------------------------------------------------------*/
   len = strlen(query);
   rc = sasSQLprepare(con,query,len,&stmt,&column_count);
   if (rc != SCA_OK)
      {
      api        = "sasSQLprepare";
      sca_handle = con;
      goto ret;
```

```
   }
else if (column_count <= 0)
{
     goto ret3;
}
sca_handle = stmt;
rc         = sasSQLdescribe(stmt, &columns);
if (rc != SCA_OK)
   {
   api = "sasSQLdescribe";
   goto ret;
   }


/*-------------------------------------------------------------------+
 |  Make a print buffer.                                             |
 +-------------------------------------------------------------------*/
 for (i = 0, cur_column = columns, len = 7;  /* "  OBS  ..." */
      i < column_count;
      i++, cur_column++)
 {
     if (cur_column->type == CHARACTER)
         len += cur_column->len > cur_column->sname_len ?
                    cur_column->len : cur_column->sname_len;
     else if (!cur_column->dtval)
         len += 12;
     else
     {
         if (cur_column->dtval & SCA_DATE_VALUE)
            len += 8;
         if (cur_column->dtval & SCA_TIME_VALUE)
            len += 8;
         len++; /* 2 values == a blank in middle */
     }
     len++; /* separator space between names/values */
 }
 len++; /* null terminator */
 buf = malloc(len);
 if ( !buf )
 {
     if (!qpa->silent)
         printf("Print buffer (%d bytes) allocation failed.\n", len);
     goto ret2;
 }


/*-------------------------------------------------------------------+
 |  Fetch all the records. The first time through, loop over the   |
 |  columns and print the name of each one in a header line. Then  |
 |  for each record print the value of each variable.              |
 +-------------------------------------------------------------------*/
api = "sasSQLfetch";
while (!(rc = sasSQLfetch(stmt)))
   {                           /* While more records              */
            data_records++;
            buf[0]  = '\0';
            buf_ptr = buf;
            if (data_records == 1 && !qpa->nohead)
            {
               if (!qpa->noid)
               {
```

```
            len = sprintf(buf_ptr, "%s  ", "  OBS");
            buf_ptr += len;
         }
         for (i = 0, cur_column = columns;
              i < column_count;
              i++, cur_column++)
            {
            len = sprintf(buf_ptr, "%.*s ",
                          cur_column->sname_len,
                          cur_column->sname);
            buf_ptr += len;
            if (cur_column->type == CHARACTER)
               {
               len = cur_column->sname_len > cur_column->len ?
                     0 : cur_column->len - cur_column->sname_len;
               }
            else if (!cur_column->dtval)
               {
               buf_ptr -= len;
               memset(buf_ptr, ' ', len);
               for (c = cur_column->sname+7, j = 0; *c == ' '; c--, j++ );
               for (c = cur_column->sname; j <= 8; j++, c++ )
                   buf_ptr[j] = *c;
               buf_ptr += 8;
               len = 4;
               }
            else
               {
               len = 0;
               if (cur_column->dtval & SCA_DATE_VALUE)
                  len += 8;
               if (cur_column->dtval & SCA_TIME_VALUE)
                  len += 8;
               if (len == 16) len++; /* 2 values == a blank in middle */
               len -= cur_column->sname_len;
               }
            if (len)
               {
               memset(buf_ptr, ' ', len);
               buf_ptr += len;
               }
            }
         *buf_ptr = '\0';
         if (qpa->htmlesc)
         {
             esc_txt(buf);
             printf("\n");
         }
         else
             printf("%s\n", buf);
         buf[0]   = '\0';
         buf_ptr  = buf;
      }
      for (i = 0, cur_column = columns;
           i < column_count;
           i++, cur_column++)
         {
         if (cur_column->type == CHARACTER)
            {
            len = cur_column->sname_len >= cur_column->len ?
                        cur_column->sname_len :
```

```
                    cur_column->len;
      len = sprintf(buf_ptr, "%-*.*s ",
                     len, cur_column->len, cur_column->data);
      }
   else if (!cur_column->dtval)
      {
      if (sasSQLmissingValue(*(double *)cur_column->data))
         len = sprintf(buf_ptr, "%s", "       .    ");
      else
         len = sprintf(buf_ptr, "%8.8g    ",
                       *(double *)cur_column->data);
      }
   else
      {
      len = 0;
      if (cur_column->dtval & SCA_DATE_VALUE)
         {
         drc = sasSQLformatDate(*(double *)cur_column->data,
                       &cur_column->format,
                       &fmtdate);
         if (!drc)
            len = sprintf(buf_ptr, "%02d%3s%02d  ",
                          fmtdate.day,
                          months[fmtdate.month],
                          (fmtdate.year%100));
         else
             {
             if (sasSQLmissingValue(*(double *)cur_column->data))
                 len = sprintf(buf_ptr, "%s", "   .     ");
             else
                 len = sprintf(buf_ptr, "%s", "*Bad Date");
             }
         }
      if (cur_column->dtval & SCA_TIME_VALUE)
         {
         buf_ptr += len;
         drc = sasSQLformatTime(*(double *)cur_column->data,
                       &cur_column->format,
                       &fmttime);
         if (!drc)
            len = sprintf(buf_ptr, "%02d:%02d:%02d ",
                          fmttime.hour,
                          fmttime.minute,
                          fmttime.second);
         else
             {
             if (sasSQLmissingValue(*(double *)cur_column->data))
                 len = sprintf(buf_ptr, "%s", "   .     ");
             else
                 len = sprintf(buf_ptr, "%s", "*Bad Time");
             }
         }
      }
   buf_ptr += len;
   }
*buf_ptr = '\0';
if (qpa->noid)
{
    if (qpa->htmlesc)
    {
        esc_txt(buf);
```

```
                    printf("\n");
                }
                else
                    printf("%s\n", buf);
            }
            else
            {
                if (qpa->htmlesc)
                {
                    printf("%5d  ", data_records);
                    esc_txt(buf);
                    printf("\n");
                }
                else
                    printf("%5d  %s\n", data_records, buf);
            }

      }                             /* End while more records           */
   if (rc == SCA_WARN_EOF)
      rc = SCA_OK;


ret:

   if (rc != SCA_OK)
   {
      if (!qpa->silent)
      {
          (void)sasSQLreasonCode(sca_handle, &reason, &error_text);
          printf("%s: returned rc=%X, reas=%X, (%s)\n",
                 api, rc, reason, error_text);
      }
   }


   /*-------------------------------------------------------------------+
    |  Free the print buffer.                                           |
    +-------------------------------------------------------------------*/

    free((void *)buf);


ret2:

   /*-------------------------------------------------------------------+
    |  Terminate the query.                                             |
    +-------------------------------------------------------------------*/
    if (stmt)
    {
        rc  = sasSQLdestroy(stmt);
        if (rc != SCA_OK)
        {
            if (!qpa->silent)
            {
                (void)sasSQLreasonCode(con, &reason, &error_text);
                printf("sasSQLdestroy: returned rc=%X, reas=%X, (%s)\n",
                       rc, reason, error_text);
            }
           rc = SCA_OK;
        }
    }
```

```
ret3:

    qpa->lines = data_records;

   return(rc);
}
```

```
/*---------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ---------------------------------------------------------------------
*
*  NAME:       sampnv.c
*  DATE:       27 Aug 1996
*  SUPPORT:    *** sample NV implementation for SAS SQL Library ***
*  PURPOSE:    Provide working sample of GET_NV & SET_NV
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*---------------------------------------------------------------------*/


#include "sampnv.h"

#ifdef MAC
#include <sys/errno.h>
#else
#include <errno.h>
#endif

#ifndef TRUE
#define TRUE  1
#endif
#ifndef FALSE
#define FALSE 0
#endif

static void   add_entry ANSI_PROTO((data_p, data_p));
static data_p find_nv ANSI_PROTO((data_p, char *));
static int    save_nv ANSI_PROTO((data_p, data_p));
static int    read_node_v1 ANSI_PROTO((data_p, FILE *));
static int    read_node ANSI_PROTO((data_p, FILE *));
static void   write_tree ANSI_PROTO((data_p, FILE *));


/*---------------------------------------------------------------------
*  NAME:     usergnv
*
*  PURPOSE:  Retrieve a value for the given name/attributes
*
*  PARMS:    vhandle:  Our handle that was passed to sasSQLinitialize().
```

```
*            name:     The entity whose attributes are being asked about.
*            attr_nv:  A caller-supplied array of attribute name/value
*                      structures (name ptr/len & value ptr/len). The
*                      names of the entity attributes for which we are
*                      to retrieve values is filled in by the caller;
*                      we fill in the value ptr/len's.
*            num_attrs:The number of elements in the attr_nv array.
*            privy:    Don't prompt for values that are not known.
*
*---------------------------------------------------------------------*/
int
usergnv(vhandle, name, attr_nv, num_attrs, privy)
   void               *vhandle;   /* Our NV handle (struct SAMP_HNDL) */
   char               *name;      /* Name of entity                   */
   SCAattrNV *         attr_nv;   /* Entity attribute name/value array*/
   int                 num_attrs; /* Number of elements in n/v array  */
   int                 privy;     /* 0:Public NV, 1:Private NV        */
{
   struct SAMP_HNDL *  handle = (struct SAMP_HNDL *) vhandle;
   char *              config_file = handle->config_file,
                       new_name[MAX_LINE+1],
                       read_buff[MAX_LINE+1];
   data_p              entry;
   FILE               *c_file = NULL;
   int                 a,
                       i,
                       j,
                       l,
                       read_it,
                       name_len,
                       new_prompt,
                       rc = SCA_OK;
   int                 first_time = 1;
   int                 prompt  = handle->config;
   data_p              nv_head = &handle->head;

   int               (*read_func) ANSI_PROTO((data_p, FILE *));
   struct DATA_PTR     node;

   /*-------------------------------------------------------------------+
    |  If this is our first time through, we will read in all of the  |
    |  configuration name/value pairs present in our config file.     |
    +-------------------------------------------------------------------*/
   if (!handle->file_loaded)
   {
       handle->file_loaded = TRUE;
      /*---------------------------------------------------------------+
       |  Open our config file, then read in all of the information   |
       |  in our config file.                                         |
       +---------------------------------------------------------------*/
      if (config_file)
          c_file = fopen(config_file, "r");
      else
          c_file = fopen(CONFIG_FILE, "r");

      if (c_file != NULL)
      {
         read_func = read_node_v1;

         i = getc(c_file);
         if (i == '*')
```

```
      {
         read_buff[0] = i;
         if (fgets(read_buff+1, MAX_LINE, c_file))
            if (strcmp(read_buff, V2_ID) == 0)
               read_func = read_node;
      }
      else
         ungetc(i, c_file);

      while ((read_it = (*read_func)(&node, c_file)) == 0)
      {
         if (node.name)
         {
            if ((rc = save_nv(nv_head, &node)))
               goto ret;
         }
      }

      fclose(c_file);

      if (read_it < 0)
      {
         rc = !SCA_OK;
         goto ret;
      }

   }
}


/*-------------------------------------------------------------------+
 |  Now that we know all of our configuration data, let's see if   |
 |  we got a name by itself.  A name by itself indicates that we   |
 |  should display it and pass back what the user enters.          |
 +-----------------------------------------------------------------*/
if (num_attrs == 1 && !attr_nv[0].name && !privy)
{
   attr_nv[0].value_len = 0;

   strcpy(read_buff, "\nEnter a ");
   strcat(read_buff, name);
   strcat(read_buff, " to configure: ");
   fputs(read_buff, stdout);
   fgets(read_buff, MAX_LINE, stdin);

   i = strlen(read_buff) - 1;
   if (read_buff[i] == '\n')
      read_buff[i] = '\0';
   for (j = 0; read_buff[j] == ' '; j++) ;
   i -= j;
   if (!(attr_nv[0].value = (char *) malloc(i+1)))
   {
      rc = SCA_ERROR_MEM;
      goto ret;
   }

   strcpy(attr_nv[0].value, &read_buff[j]);
   attr_nv[0].value_len = i;

   if (i == 1)
      if (read_buff[j] == 'e')
```

```
         rc = SCA_SETUP_END;
      else if (read_buff[j] == 'c')
         rc = SCA_SETUP_CANCEL;
   goto ret;
}

/*-------------------------------------------------------------------+
 |  Ok, if we get here, the caller is interested in attribute info |
 |  related to the name.  But, he could be asking for multiple     |
 |  attribute values at once.  So, we'll have to loop through them.|
 +-------------------------------------------------------------------*/
strcpy(new_name, name);
name_len             = strlen(new_name);
new_name[name_len++] = ':';
for (a = 0; !rc && a < num_attrs; a++)
{
   strcpy(&new_name[name_len], attr_nv[a].name);
   new_prompt = prompt;
   if ((entry = find_nv(nv_head, new_name)))
   {
      if (privy && !strcmp("Servers", name))
      {
         printf("\nDo you want to update configuration for server %s? ",
                attr_nv[a].name);
         read_buff[0] = '\0';
         fgets(read_buff, MAX_LINE, stdin);
         if (read_buff[0] != 'Y' && read_buff[0] != 'y')
         {
            attr_nv[a].value     = entry->value;
            attr_nv[a].value_len = entry->value_len;
         }
      }
      else
      {
         attr_nv[a].value     = entry->value;
         attr_nv[a].value_len = entry->value_len;
         new_prompt &= ~PROMPT_NOT_KNOWN;
      }
   }
   if (new_prompt && !privy)
   {
      if (first_time)
      {
         fputs("\nEnter information for: ", stdout);
         strcpy(read_buff, name);
         fputs(read_buff, stdout);
         fputs("\n \n", stdout);
         first_time = 0;
      }
   /*
      strcpy(read_buff, attr_nv[a].desc);
      l = strlen(read_buff);
      read_buff[l]     = ' ';
      read_buff[l + 1] = '[';
      i = 0;
      if (entry && entry->value_len)
      {
         strcpy(&read_buff[l+2], entry->value);
         i = entry->value_len;
      }
      read_buff[l + 2 + i] = ']';
```

```
         read_buff[l + 3 + i] = ':';
         read_buff[l + 4 + i] = ' ';
         read_buff[l + 5 + i] = '\0';
         fputs(read_buff, stdout);
      */
         if (entry && entry->value_len)
         {
            char *  value;

            value = (char *) malloc(entry->value_len+1);
            memcpy(value, entry->value, entry->value_len);
            value[entry->value_len] = '\0';
            printf("%s [%s]: ", attr_nv[a].desc, value);
            free((void *) value);
         }
         else
            printf("%s: ", attr_nv[a].desc);

         read_buff[0] = '\0';
         fgets(read_buff, MAX_LINE, stdin);
         i = strlen(read_buff) - 1;
         if (read_buff[i] == '\n')
            read_buff[i] = '\0';
         for (j = 0; read_buff[j] == ' '; j++) ;
         if (read_buff[j])
         {
            l = i - j;
            if (l == 1)
            {
               if (read_buff[j] == 'e')
                  rc = SCA_SETUP_END;
               else if (read_buff[j] == 'c')
                  rc = SCA_SETUP_CANCEL;
            }
            if (!rc)
            {
               if (attr_nv[a].value = malloc(l))
               {
                  memcpy(attr_nv[a].value, read_buff+j, l);
                  attr_nv[a].value_len = l;
               }
               else
                  rc = SCA_SETUP_CANCEL;
            }
         }
      }
   }

ret:

   if (c_file != NULL)
      (void)fclose(c_file);

   return(rc);
}


int
usersnv(vhandle, name, attr, value, val_len)
   void               *vhandle;   /* User's NV pair handle           */
   char               *name;      /* Name of containing key          */
```

```
   char               *attr;      /* Name of attribute values to set  */
   char               *value;     /* Caller provided value dest       */
   int                 val_len;   /* length of supplied value         */
{
   struct SAMP_HNDL *  handle = (struct SAMP_HNDL *) vhandle;
   char *              config_file = handle->config_file,
                      *new_name,
                      *new_value;
   char                work[MAX_LINE+1];
   data_p              entry;
   FILE               *c_file;
   int                 rc = SCA_OK;
   data_p              nv_head = &handle->head;

   /*-------------------------------------------------------------------+
    |  A NULL name pointer indicates that the client is being          |
    |  terminated and you may want to save any name/value information |
    |  collected during the client's execution.                        |
    +-------------------------------------------------------------------*/
   if (!name)
   {
      /*----------------------------------------------------------------+
       |  Open our config file, then write out all of the information |
       |  we've collected.                                            |
       +----------------------------------------------------------------*/
      if (config_file)
          c_file = fopen(config_file, "w");
      else
          c_file = fopen(CONFIG_FILE, "w");

      if (c_file == NULL)
      {
         if (errno == ENOENT)
            return(SAMPNV_NOFILE);
         else if (errno == EACCES)
            return(SAMPNV_NOAUTH);
         else
         {
            printf("\nOpen failed for data source definition file %s errno=%d\n",
                   config_file ? config_file : CONFIG_FILE, errno);
            return(SCA_SETUP_GENERR);  /* indicate we've already messaged */
         }
      }

      fputs(V2_ID, c_file);

      write_tree(nv_head->left,c_file);
      write_tree(nv_head->right,c_file);

      (void)fclose(c_file);

      return(rc);
   }

   /*-------------------------------------------------------------------+
    |  Ok, now we need to construct our name and save it with its     |
    |  associated value.  If we already have a 'name', replace its    |
    |  value.  Otherwise, create a new DATA_PTR and place it in our   |
    |  tree.                                                           |
    +-------------------------------------------------------------------*/
   strcpy(work,name);
```

```
   strcat(work,":");
   strcat(work,attr);
   if (value && val_len)
   {
      if (!(new_value = malloc(val_len)))
         return(!SCA_OK);

      memcpy(new_value, value, val_len);
   }

   if ((new_name = strdup(work)))
      {
      if ((entry = find_nv(nv_head, new_name)))
         {
         free((void *)new_name);
         if (entry->value)
            free((void *)entry->value);
         if (val_len)
            entry->value = new_value;
         else
            entry->value = NULL;
         entry->value_len = val_len;
         }
      else
         {
         if ((entry = (data_p) malloc(sizeof(struct DATA_PTR))))
            {
            entry->name     = new_name;
            entry->name_len = strlen(work);
            if (val_len)
               entry->value = new_value;
            else
               entry->value = NULL;
            entry->left      =
            entry->right     =
            entry->parent    = NULL;
            entry->value_len = val_len;
            add_entry(nv_head, entry);
            }
         else
            {
            if (val_len)
               free((void *)new_value);
            free((void *)new_name);
            rc = !SCA_OK;
            }
         }
      }
   else
      {
      if (val_len)
         free((void *)new_value);
      rc = !SCA_OK;
      }

   return(rc);
}


static void
add_entry(parent, node)
```

```
   data_p              parent;    /* NV (sub)tree head               */
   data_p              node;      /* NV node to insert               */
{

   int                 order;

   /*---------------------------------------------------------------+
    |  An equal condition should never happen since save_nv checks  |
    |  for duplicates.                                              |
    +---------------------------------------------------------------*/
   order = strcmp(node->name, parent->name);

   if (order < 0)
      {
      if (parent->left == NULL)
         {
         parent->left = node;
         node->parent = parent;
         }
      else
         add_entry(parent->left, node);
      }
   else
      {
      if (parent->right == NULL)
         {
         parent->right = node;
         node->parent = parent;
         }
      else
         add_entry(parent->right, node);
      }
}


static data_p
find_nv(node, name)
   data_p              node;      /* NV (sub)tree head               */
   char               *name;      /* Name in which to locate         */
{

   int                 order;

   /*---------------------------------------------------------------+
    |  If node is NULL, then we've reached the end of tree without  |
    |  locating the entry requested.                                |
    +---------------------------------------------------------------*/
   if (node == NULL)
      return(node);

   if (!(order = strcmp(name, node->name)))
      return(node);

   if (order < 0)
      return(find_nv(node->left,name));
   else
      return(find_nv(node->right,name));
}


static int
```

```
save_nv(nv_head, node)
   data_p              nv_head;   /* NV tree head                     */
   data_p              node;
{
   data_p              entry;
   int                 rc = SCA_OK;

   /*-------------------------------------------------------------------+
    |  Ok, now that we've constructed everything we need, let's see   |
    |  if we already have an item saved away with this name.  If so,  |
    |  replace its value.  If not, create a new DATA_PTR and place it |
    |  in our tree.                                                   |
    +-------------------------------------------------------------------*/
   if ((entry = find_nv(nv_head, node->name)))
   {
      if (entry->value)
         free((void *)entry->value);
      if (node->value_len)
         entry->value = node->value;
      else
         entry->value = NULL;
      entry->name_len  = node->name_len;
      entry->value_len = node->value_len;
   }
   else
   {
      if ((entry = (data_p) malloc(sizeof(struct DATA_PTR))))
      {
         entry->name     = node->name;
         entry->name_len = node->name_len;
         if (node->value_len)
            entry->value = node->value;
         else
            entry->value = NULL;
         entry->left      =
         entry->right     =
         entry->parent    = NULL;
         entry->value_len = node->value_len;
         add_entry(nv_head, entry);
      }
      else
         rc = !SCA_OK;
   }

   /*-------------------------------------------------------------------+
    |  Time to return to our caller.                                  |
    +-------------------------------------------------------------------*/
   if (rc)
   {
      if (node->name)
         free((void *)node->name);
      if (node->value)
         free((void *)node->value);
      if (entry)
      {
         if (entry->name)
            free((void *)entry->name);
         if (entry->value)
            free((void *)entry->value);
         free((void *)entry);
      }
```

```
   }

   return(rc);
}


static int
read_node_v1(node, c_file)
   data_p              node;      /* Current NV node                 */
   FILE               *c_file;    /* Configuration file stream       */
{
   char                read_buff[MAX_LINE+1],
                      *buff,
                      *attr,
                      *name,
                      *temp,
                      *value;
   int                 attr_len,
                       name_len,
                       val_len;


   node->name     = NULL;
   node->name_len = 0;


   if (!fgets(read_buff, MAX_LINE, c_file))
      return(1);


   buff = read_buff;
   if (*buff == '*')
      return(0);


   /*---------------------------------------------------------------------+
    |  First, let's skip leading blanks.  If we end up with a blank       |
    |  line, just act like everything went ok.                            |
    +---------------------------------------------------------------------*/
   name_len = strlen(buff);
   if (buff[name_len - 1] == '\n')
      buff[--name_len] = '\0';
   while (*buff == ' ' && name_len)
   {
      buff++;
      name_len--;
   }
   if (!name_len)
      return(0);


   /*---------------------------------------------------------------------+
    |  Ok, our configuration information is formatted as follows:         |
    |                       name:attr=value                               |
    |                                                                     |
    | first find the :, eliminate trailing blanks after the name          |
    +---------------------------------------------------------------------*/
   if (!(attr = strchr(buff, ':')))
      return(-1);
   name_len = attr - buff;
   if (!name_len)
```

```
      return(-1);
   for (temp = attr - 1; *temp == ' '; temp--);
   name_len = temp - buff + 1;

   /*-------------------------------------------------------------------+
    | now skip leading blanks before the attr, find the =, and get      |
    | rid of trailing blanks after the attr                             |
    +-------------------------------------------------------------------*/
   for (attr++; *attr == ' '; attr++);
   if (!(value = strchr(attr, '=')))
      return(-1);
   for (temp = value - 1; *temp == ' '; temp--);
   attr_len = temp - attr + 1;

   /*-------------------------------------------------------------------+
    | make a buffer for the (trimmed) name:attr and copy it there       |
    +-------------------------------------------------------------------*/
   if (!(name = malloc(name_len + 1 + attr_len + 1)))
      return(-1);
   strncpy(name,buff,name_len);
   name[name_len] = '\0';
   strcat(name,":");
   strncat(name,attr,attr_len);

   /*-------------------------------------------------------------------+
    | if there's a value there, make a copy of it to keep               |
    +-------------------------------------------------------------------*/
   temp     = value + 1;
   value    = NULL;
   val_len = strlen(temp);
   if (val_len)
      if (!(value = strdup(temp)))
      {
         free((void *)name);
         return(-1);
      }


   /*-------------------------------------------------------------------+
    | finally return the parsed info as a node                          |
    +-------------------------------------------------------------------*/
   node->name     = name;
   node->name_len = name_len;
   if (val_len)
      node->value = value;
   else
      node->value = NULL;
   node->value_len = val_len;
   node->left      =
   node->right     =
   node->parent    = NULL;


   return(0);
}


static int
read_node(node, c_file)
   data_p              node;      /* Current NV node                */
   FILE               *c_file;    /* Configuration file stream      */
```

```
{
   char                read_buff[MAX_LINE+1],
                      *buff,
                      *attr,
                      *name,
                      *temp,
                      *len,
                      *value;
   int                 line_len,
                       attr_len,
                       name_len = 0,
                       val_len = -1,
                       i, l;


   node->name     = NULL;
   node->name_len = 0;


   if (!fgets(read_buff, MAX_LINE, c_file))
      return(1);


   buff = read_buff;
   if (*buff == '*')
      return(0);


   /*---------------------------------------------------------------------+
    |  First, let's skip leading blanks.  If we end up with a blank   |
    |  line, just act like everything went ok.                            |
    +---------------------------------------------------------------------*/
   line_len = name_len = strlen(buff);
   while (*buff == ' ' && name_len)
   {
      buff++;
      name_len--;
   }
   if (!name_len)
      return(0);


   /*---------------------------------------------------------------------+
    |  Ok, our configuration information is formatted as follows:     |
    |                       name:attr=len value                           |
    |                                                                     |
    | first find the :, eliminate trailing blanks after the name      |
    +---------------------------------------------------------------------*/
   if (!(attr = strchr(buff, ':')))
      return(-1);
   name_len = attr - buff;
   if (!name_len)
      return(-1);
   for (temp = attr - 1; *temp == ' '; temp--);
   name_len = temp - buff + 1;

   /*---------------------------------------------------------------------+
    | now skip leading blanks before the attr, find the =, and get    |
    | rid of trailing blanks after the attr                               |
    +---------------------------------------------------------------------*/
   for (attr++; *attr == ' '; attr++);
```

```
if (!(len = strchr(attr, '=')))
   return(-1);
for (temp = len - 1; *temp == ' '; temp--);
attr_len = temp - attr + 1;

/*-------------------------------------------------------------------+
 | make a buffer for the (trimmed) name:attr and copy it there       |
 +-------------------------------------------------------------------*/
if (!(name = malloc(name_len + 1 + attr_len + 1)))
   return(-1);
strncpy(name,buff,name_len);
name[name_len] = '\0';
strcat(name,":");
strncat(name,attr,attr_len);

/*-------------------------------------------------------------------+
 | find the value length (len), read it, and skip over it            |
 +-------------------------------------------------------------------*/
temp     = len + 1;
value    = NULL;
for ( ; *temp == ' '; temp++ );
sscanf(temp, "%d", &val_len);
if (val_len < 0)
   return(-1);
for ( ; *temp != ' '; temp++ );
temp++;

/*-------------------------------------------------------------------+
 | if there's supposed to be a value there, read that many bytes     |
 +-------------------------------------------------------------------*/
if (val_len)
{
   /*----------------------------------------------------------------+
    | if we have the whole thing already, just make a copy to keep   |
    +----------------------------------------------------------------*/
   l = strlen(temp);
   if (l == val_len)
   {
      if (!(value = strdup(temp)))
      {
         free((void *)name);
         return(-1);
      }
   }
   /*----------------------------------------------------------------+
    | if we have the value and more (like a newline), make a         |
    | buffer of the right length and copy the value to it            |
    +----------------------------------------------------------------*/
   else if (l > val_len)
   {
      if (!(value = malloc(val_len+1)))
      {
         free((void *)name);
         return(-1);
      }
      memcpy(value, temp, val_len);
      value[val_len] = '\0';
   }
   /*----------------------------------------------------------------+
    | if there is a null or newline in the middle of the value so    |
    | we don't have it all (or we couldn't tell), copy characters    |
```

```
       | and get new lines as we need 'em                                  |
       +-----------------------------------------------------------------*/
      else /* (l < val_len) */
      {
         if (!(value = malloc(val_len)))
         {
            free((void *)name);
            return(-1);
         }
         for (i = 0; i < val_len; i++)
         {
             value[i] = *temp;
             if (*temp == '\n')
             {
                if (!fgets(read_buff, MAX_LINE, c_file))
                   return(-1);
                temp = read_buff;
             }
             else
                temp++;
         }
      }
   }


   /*-----------------------------------------------------------------+
    | finally return the parsed info as a node                        |
    +-----------------------------------------------------------------*/
   node->name     = name;
   node->name_len = name_len;
   if (val_len)
      node->value = value;
   else
      node->value = NULL;
   node->value_len = val_len;
   node->left      =
   node->right     =
   node->parent    = NULL;


   return(0);
}


static void
write_tree(node, c_file)
   data_p              node;      /* Current NV node                 */
   FILE               *c_file;    /* Configuration file stream       */
{
   char                write_buff[MAX_LINE+1];
   char                len_buff[6];
   int                 l;

   if (!node)
      return;

   /*-----------------------------------------------------------------+
    |  If we can still go down, keep going.  Otherwise, record our    |
    |  node's information and proceed down the greater than path.     |
    +-----------------------------------------------------------------*/
   write_tree(node->left, c_file);
```

```
   strcpy(write_buff,node->name);
   strcat(write_buff,"=");
   sprintf(len_buff, "%d ", node->value_len);
   strcat(write_buff, len_buff);
   l = strlen(write_buff);
   if (node->value)
   {
      memcpy(write_buff+l, node->value, node->value_len);
      l += node->value_len;
   }
   fwrite(write_buff, sizeof(char), l, c_file);
   putc('\n', c_file);

   write_tree(node->right, c_file);

   free((void *)node->name);
   if (node->value)
      free((void *)node->value);
   free((void *)node);

   return;
}
```

```
/*---------------------------------------------------------------------
*  Copyright (C), 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ---------------------------------------------------------------------
*
*  NAME:       dsfile.h
*  DATE:       27 Aug 1996
*  SUPPORT:    *** sample client implementation ***
*  PURPOSE:    define default data source definition file names
*              for an application that uses sample NV implementation
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*--------------------------------------------------------------------*/


#define  CONFIG_FILE            "/usr/local/src/qsas/qsas.config"
#define  CONFIG_OVERRIDE_FILE   "/usr/local/src/qsas/.qsas.config.override"
```

```
/*---------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ---------------------------------------------------------------------
```

```
*
*  NAME:       qpsas.h
*  DATE:       22 Sep 1995
*  SUPPORT:    *** sample client implementation ***
*  PURPOSE:    Header file for qpsas.c
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*-----------------------------------------------------------------------*/


/*-----------------------------------------------------------------------*/
/* a block for passing/returing info                                     */
/*-----------------------------------------------------------------------*/

struct QP
{
    int     nohead;         /* I - do not write header line           */
    int     noid;           /* I - do not write record number         */
    int     nowhitesp;      /* I - do not write blank lines bef./aft. */
    int     silent;         /* I - do not write error messages        */
    int     htmlesc;        /* I - escape special chars for HTML      */
    int     lines;          /* O - number of lines written            */
};


/*-----------------------------------------------------------------------*/
/* return codes                                                          */
/*-----------------------------------------------------------------------*/

#define QPSAS_NOMEM             -999
#define QPSAS_NOQUERY           -998
```

```
/*-----------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ----------------------------------------------------------------------
*
*  NAME:       qsas.h
*  DATE:       22 Sep 1995
*  SUPPORT:    *** sample client implementation ***
*  PURPOSE:    Header file for qsas
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*-----------------------------------------------------------------------*/

/*-----------------------------------------------------------------------*/
/* return codes                                                          */
```

```
/*---------------------------------------------------------------------*/

#define QSAS_NOMEM              -99
#define QSAS_NOCONFIGOVERRIDE   -98
#define QSAS_NODSN              -97
```

```
/*---------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ---------------------------------------------------------------------
*
*  NAME:       sampnv.h
*  DATE:       27 Aug 1996
*  SUPPORT:    *** sample NV implementation for SAS SQL Library ***
*  PURPOSE:    header file for sample GET_NV & SET_NV
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*---------------------------------------------------------------------*/


/*---------------------------------------------------------------------*/
/* define the SQL library structures, symbols, etc.                    */
/*---------------------------------------------------------------------*/
#include "sasSQL.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


/*---------------------------------------------------------------------*/
/* this file defines a symbol for the name of the default data source */
/* definition file for whatever application is being compiled          */
/*---------------------------------------------------------------------*/
#include "dsfile.h"


/*---------------------------------------------------------------------*/
/* this comment line identifies files written by v2 of sampnv.c       */
/*---------------------------------------------------------------------*/
#define  V2_ID "* SAS SQL Library for C (sasSQL) default NV routines (v2)\n"


/*---------------------------------------------------------------------*/
/* longest line of the input file we'll read in a single fgets()      */
/*---------------------------------------------------------------------*/
#define  MAX_LINE    1024


/*---------------------------------------------------------------------*/
/* a node of the tree we use to store entity attributes                */
```

```
/*     "name" points to "<entity-name>:<attribute name>"              */
/*---------------------------------------------------------------------*/
typedef struct DATA_PTR *data_p;

struct DATA_PTR
   {
   data_p              parent;
   data_p              left;
   data_p              right;
   char               *name;
   int                 name_len;
   char               *value;
   int                 value_len;
   };


/*---------------------------------------------------------------------*/
/* the handle for our sample GET_NV & SET_NV functions                 */
/*---------------------------------------------------------------------*/
struct SAMP_HNDL
   {
   int                 config;      /* flags for prompting behavior   */
#define  NO_PROMPT        0
#define  PROMPT_ALWAYS    1
#define  PROMPT_NOT_KNOWN 2

   char *              config_file; /* the name of the data source    */
                                    /* definition file (use default   */
                                    /* from dsfile.h if NULL)         */
   int                 file_loaded; /* T = dsrc file has been loaded */

   struct DATA_PTR     head;        /* the top node of the tree       */

   int                 flags;
#define  PRINT_TREE       0x01

   };


/*---------------------------------------------------------------------*/
/* prototypes for our functions                                        */
/*---------------------------------------------------------------------*/
#ifdef NO_PROTOTYPES
#define ANSI_PROTO(a) ()
#else
#define ANSI_PROTO(a) a
#endif

int   usergnv   ANSI_PROTO((
                  void *,           /* User's name/value pair handle  */
                  char *,           /* Name of containing key         */
                  SCAattrNV *,      /* Array of attribute name/values */
                  int,              /* Number of elements in n/v array*/
                  int               /* 0:Public NV, 1:Private NV      */
                ));

int   usersnv   ANSI_PROTO((
                  void *,           /* User's name/value pair handle  */
                  char *,           /* Name of containing key         */
                  char *,           /* Name of attribute w/in key     */
                  char *,           /* Value in which to set for name */
```

```
                int             /* Length of value              */
              ));

/*---------------------------------------------------------------------*/
/* return codes for our functions                                      */
/*---------------------------------------------------------------------*/
#define SAMPNV_NOFILE         351   /* data source def file not found */
#define SAMPNV_NOAUTH         352   /* not allowed to write ds def    */
```

```
/*---------------------------------------------------------------------
*  Copyright (C), 1994, 1995, 1996, 1997, 1998, 1999
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ---------------------------------------------------------------------
*
*  NAME:       sasSQL.h
*  PRODUCT:    SAS SQL Library for C  (Version 1.3)
*  PURPOSE:    Prototypes and interface structures
*
*---------------------------------------------------------------------*/
#ifndef SASSQL_H
#define SASSQL_H

/*---------------------------------------------------------------------*/
/*-- SCAconP is a parameter block for sasSQLconnect                 --*/
/*---------------------------------------------------------------------*/
typedef struct SCACONP
{
   char *  ds;                  /* data source name                */
   char *  serv_host;           /* server host IP name or addr     */
   char *  serv_port;           /* server service name or port #   */
   char *  userid;              /* server host userid              */
   char *  pw;                  /* server host password            */
   char *  sapw;                /* server access password          */
   int     defer_destroys;      /* # of destroys to defer          */
                                /*    -1 = MAX (v1.1 = 64)         */
                                /*     0 = none                    */
                                /*     n = min(n,MAX)              */
   int     bufrecs;             /* # of recs to ask for in a       */
                                /*    single request to server     */
} SCAconP;
typedef SCAconP * pSCAconP;
#define SCA_CP_LEN sizeof(SCAconP)

/*---------------------------------------------------------------------*/
/*-- handles returned to client applications by the SQL library     --*/
/*---------------------------------------------------------------------*/
typedef void *SCAenv;           /* Client environment handle       */
typedef void *SCAcon;           /* Connection handle               */
typedef void *SCAstmt;          /* SELECT statement handle         */

/*---------------------------------------------------------------------*/
/*--          SCAcol describes a column in a results set           --*/
/*---------------------------------------------------------------------*/
typedef struct SCACOL
```

```
   {
   int    type;                    /* NUMERIC or CHARACTER          */
#define NUMERIC         1
#define CHARACTER       2

   int    len;                     /* length MAX(200)!!!            */

   char   sname[32];               /* A Useful SAS name.            */
   int    sname_len;               /*    ...and its length          */
   char   dname[32];               /* The DBMS name                 */
   int    dname_len;               /*    ...and its length          */
   char   label[256];              /* A Useful label                */
   int    label_len;               /*    ...and its length          */

   struct SCAFORMAT
      {
      char     name[32];
      int      name_len;
      short    wid, ndec;
      } format, informat;

   char   dtval;                   /* Date and/or Time value present */
#define SCA_DATE_VALUE 0x01        /*  ...value represents date      */
#define SCA_TIME_VALUE 0x02        /*  ...value represents time      */
                                   /* both on for datetime values    */

   void  *data;                    /* Data associated w/column       */
   int    trunc;                   /* Value was truncated            */
   } *SCAcol;                      /* Column descriptor              */

/*---------------------------------------------------------------------*/
/*--          Descriptions of a formatted  SAS date/time            --*/
/*---------------------------------------------------------------------*/
typedef struct SCAFDATE
   {
   int                 month;      /* Month      (5)                 */
   int                 day;        /* Day        (9)                 */
   int                 year;       /* Year       (1994)              */
   int                 quarter;    /* Quarter    (2)                 */
   int                 julday;     /* Julian day (129)               */
   int                 weekday;    /* Weekday (1 for Sunday)         */
   } *SCAfdate;

typedef struct SCAFTIME
   {
   int                 hour;       /* Hour    (13)                   */
   int                 minute;     /* Minute  (45)                   */
   int                 second;     /* Second  (37)                   */
   } *SCAftime;                    /* Formatted time                 */

/*---------------------------------------------------------------------*/
/*--             USER-SUPPLIED NAME/VALUE ROUTINES                 --*/
/*--            (sample implementation in sampnv.c)                --*/
/*---------------------------------------------------------------------*/
typedef struct SCA_ATTR_NV
{
   char * name;
   int    name_len;
   char * desc;
   int    desc_len;
   char * value;
```

```
   int    value_len;
} SCAattrNV;
#define SCA_ANV_LEN sizeof(SCAattrNV)


#ifdef NO_PROTOTYPES
#define ANSI_PROTO(a) ()
#else
#define ANSI_PROTO(a) a
#endif

typedef
int   (*GET_NV) ANSI_PROTO((
                  void *,          /* User's name/value pair handle  */
                  char *,          /* Name of containing key         */
                  SCAattrNV *,     /* Array of attribute name/values */
                  int,             /* Number of elements in n/v array*/
                  int              /* 0:Public NV, 1:Private NV      */
                ));

typedef
int   (*SET_NV) ANSI_PROTO((
                  void *,          /* User's name/value pair handle  */
                  char *,          /* Name of containing key         */
                  char *,          /* Name of attribute w/in key     */
                  char *,          /* Value in which to set for name */
                  int              /* Length of value                */
                ));

/*------------------------------------------------------------------------*/
/*--          FUNCTIONS THAT MAKE UP THE SQL LIBRARY                --*/
/*------------------------------------------------------------------------*/

#ifdef WNC
#define U_EXPORT __declspec(dllexport)
#else
#define U_EXPORT
#endif

U_EXPORT
int   sasSQLconfigure
                ANSI_PROTO((
                  SCAenv           /* Environment handle             */
                ));

U_EXPORT
int   sasSQLconnect
                ANSI_PROTO((
                  SCAenv,          /* Environment handle             */
                  SCAcon *,        /* Connection handle, returned    */
                  char   *,        /* Application name, optional     */
                  pSCAconP         /* Parm block (see above)         */
                ));

U_EXPORT
int   sasSQLdescribe
                ANSI_PROTO((
                  SCAstmt,         /* SELECT statement handle        */
                  SCAcol  *        /* Column description, updated    */
                ));
```

```
U_EXPORT
int   sasSQLdestroy
                ANSI_PROTO((
                  SCAstmt           /* SELECT statement handle        */
                ));

U_EXPORT
int   sasSQLdisconnect
                ANSI_PROTO((
                  SCAcon            /* Connection handle              */
                ));

U_EXPORT
int   sasSQLexecute
                ANSI_PROTO((
                  SCAcon,           /* Connection handle              */
                  char   *,         /* EXECUTE statement address      */
                  int,              /* EXECUTE statement length       */
                                  int *                          /* rows updated, ins
                ));

U_EXPORT
int   sasSQLfetch
                ANSI_PROTO((
                  SCAstmt           /* SELECT statement handle        */
                ));

U_EXPORT
int   sasSQLformatDate
                ANSI_PROTO((
                  double,           /* SAS date or datetime value     */
                  struct SCAFORMAT*,/* Format descriptor for column   */
                  SCAfdate          /* Formatted date, returned       */
                ));

U_EXPORT
int   sasSQLformatTime
                ANSI_PROTO((
                  double,           /* SAS time or datetime value     */
                  struct SCAFORMAT*,/* Format descriptor for column   */
                  SCAftime          /* Formatted time, returned       */
                ));

U_EXPORT
int   sasSQLinitialize
                ANSI_PROTO((
                  SCAenv *,         /* Environment handle, returned   */
                  void   *,         /* User's name/value pair handle  */
                  GET_NV,           /* Get name/value function        */
                  SET_NV            /* Set name/value function        */
                ));

U_EXPORT
int   sasSQLmissingValue
                ANSI_PROTO((
                  double            /* Value to test for missing      */
                ));

U_EXPORT
int   sasSQLprepare
                ANSI_PROTO((
```

```
                  SCAcon,           /* Connection handle              */
                  char    *,        /* SELECT statement address       */
                  int,              /* SELECT statement length        */
                  SCAstmt *,        /* SELECT stmt handle, returned   */
                  int     *         /* Number of variables, returned  */
                ));

U_EXPORT
int   sasSQLreasonCode
                ANSI_PROTO((
                  void   *,         /* A handle                       */
                  int    *,         /* Reason code, returned          */
                  char  **          /* Error text, returned           */
                ));

U_EXPORT
int   sasSQLterminate
                ANSI_PROTO((
                  SCAenv            /* Environment handle             */
                ));

/*--------------------------------------------------------------------*/
/*--            BRING IN SQL LIBRARY RETURN/REASON CODES            --*/
/*--------------------------------------------------------------------*/
#include "sasSQLrc.h"


#endif   /* ifndef SASSQL_H */
```

```
/*--------------------------------------------------------------------
*  Copyright (C), 1994, 1995, 1996, 1997, 1998, 1999
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* -------------------------------------------------------------------
*
*  NAME:       sasSQLrc.h
*  PRODUCT:    SAS SQL Library for C  (Version 1.3)
*  PURPOSE:    Return/reason codes
*
*--------------------------------------------------------------------*/

/*--------------------------------------------------------------------*/
/*--                   SQL LIBRARY RETURN CODES                     --*/
/*--------------------------------------------------------------------*/
#define SCA_OK                  0   /* Successful completion          */
#define SCA_ERROR_API           1   /* Error came from SAS            */
#define SCA_ERROR_MEM           2   /* Unable to allocate memory      */
#define SCA_ERROR_PARM          3   /* Invalid input parameter        */
#define SCA_ERROR_SEQ           4   /* SQL lib API call out of order  */
#define SCA_ERROR_SERVER        5   /* Server is undefined            */
#define SCA_ERROR_COMM          6   /* Access method error, call      */
                                    /*   reasonCode() for reason code */
#define SCA_ERROR_TABLE         7   /* Table not found                */
#define SCA_ERROR_COLUMN        8   /* Column not found               */
#define SCA_ERROR_CHARS         9   /* Illegal character in data      */
                                    /*   source name                  */
```

```
#define SCA_ERROR_INTBUFF      10   /* Internal error in connect() or */
                                    /*   configure() -- please report */
                                    /*   it to SAS Institute          */
#define SCA_WARN_EOF           11   /* End of file was reached        */
#define SCA_WARN_DISC          12   /* terminate() caused disconnects */
#define SCA_WARN_NOEXEC        13   /* execute() not allowed for this */
                                    /*   connection or error exec'ing */
                                    /*   SQL statement                */
#define SCA_WARN_API           14   /* Warning came from SAS          */
#define SCA_ERROR_LIBREF       15   /* Libref longer than 8-char max  */

#define SCA_ERROR_DESCERR      21   /* describe() or fetch() called   */
                                    /* after prev. describe() error   */

#define SCA_CHAR_TRUNC        101   /* CHAR data was truncated        */
#define SCA_NUM_TRUNC         102   /* Conversion truncated NUM data  */

#define SCA_SETUP_END         300   /* End setup and save information */
#define SCA_SETUP_CANCEL      301   /* Cancel w/o saving configuration*/
#define SCA_SETUP_GENERR      399   /* Error; message written by NV   */
                                    /*    routines                    */

/*-------------------------------------------------------------------*/
/*--            COMMUNICATIONS ACCESS METHOD REASON CODES          --*/
/*-------------------------------------------------------------------*/
#define AM_ERROR_NOTINITED  -1001  /* AM not initialized             */
#define AM_ERROR_INITED     -1002  /* AM already initialized         */
#define AM_ERROR_REENTERED  -1003  /* AM re-entered illegally        */
#define AM_ERROR_LICENSE    -1004  /* Facility not licensed          */
#define AM_ERROR_NOSERVER   -1005  /* No server found                */
#define AM_ERROR_BADENV     -1006  /* Bad environment argument       */
#define AM_ERROR_BADCON     -1007  /* Bad connection argument        */
#define AM_ERROR_BADADDRESS -1008  /* Bad server address             */
#define AM_ERROR_BADSERVICE -1009  /* Bad service                    */
#define AM_ERROR_CLOSED     -1010  /* Connection closed              */
#define AM_ERROR_NOCONNECT  -1011  /* No connection                  */
#define AM_ERROR_BADPARM    -1012  /* Bad parameter                  */
#define AM_ERROR_MEMORY     -1013  /* No memory                      */
#define AM_ERROR_NOTIMP     -1014  /* Not implemented                */
#define AM_ERROR_NETFAIL    -1015  /* Network failure                */
#define AM_ERROR_NOTCOMP    -1016  /* Request not completed          */
#define AM_ERROR_TRUNC      -1017  /* Data truncation occured        */
#define AM_ERROR_SASTIMEOUT -1018  /* Start SAS timeout              */
#define AM_ERROR_SECURITY   -1019  /* Userid.password security failure*/
#define AM_ERROR_INTR       -1020  /* Connection was interrupted     */
#define AM_ERROR_SHELLFAIL  -1021  /* ShellExecute failure           */
#define AM_ERROR_BADHEADER  -1022  /* Bad header                     */
#define AM_ERROR_BADPROT    -1023  /* Bad AM protocol                */
#define AM_ERROR_ERRNO      -1024  /* AM Error, call scareas         */
#define AM_ERROR_CONNREF    -1025  /* Connection refused             */
#define AM_ERROR_ENCR       -1026  /* encryption err (details in msg) */
```

Instructions for Using qsasconfig

The qsasconfig program is a line-mode configuration program that defines data sources for use with qsas. The definition information is written to the definition file specified by the `-config` option. If `-config` is not specified, the definition is written to file `qsas.config` in the directory in which qsasconfig is run. If the definition file already exists, it is updated; otherwise, it is created.

Note: If you use a Windows operating system, you must run this program from an MS-DOS command prompt window.

Syntax

`qsasconfig` `[-config path-name]`

Parameters

`-config path-name`
: the qsas data source definition pathname. Data source names can be any length that your application allows them to be and can contain any character except a double quotation mark (").

  `-c` is an alias for `-config`.

Description of Program Execution

The qsasconfig program first prompts for a data source to configure, waiting until you enter a data source name, a null line, or the character `e`. For each data source name that you enter, qsasconfig prompts for SAS/SHARE server information and other general information. It then prompts for a library to configure.

For each library, qsasconfig further prompts for particulars such as the library pathname, the engine for the server to use, and options to be specified when the library is accessed at data source connect time. After collecting the library information, it prompts you to specify another library and waits for you to enter a library name, a null line, or the character `e`.

After you finish specifying libraries, qsasconfig prompts for another data source. If you have no more data sources to define, enter the character `e` to save the data source configuration file.

See qsasconfig Dialog to view the dialog that is generated by qsasconfig.

```
/*-----------------------------------------------------------------------
*  Copyright (C), 1995, 1996, 1997, 1998
*  SAS Institute Inc., Cary, N.C. 27513, U.S.A.  All rights reserved.
* ----------------------------------------------------------------------
*
*  NAME:       qsasconfig.c
*  DATE:       18 Oct 1995
*  SUPPORT:    sasbrp - Brian Perkinson
*  PURPOSE:    configure data sources for qsas
*
*  NOTES:      _
*
*  ALGORITHM:  _
*
*  END
*----------------------------------------------------------------------*/

/*----------------------------------------------------------------------*/
/* include stuff for the SAS SQL Library                                */
/*----------------------------------------------------------------------*/
#include "sssSQL.h"

/*----------------------------------------------------------------------*/
/* include stuff for the sample NV routines                             */
/*----------------------------------------------------------------------*/
#include "sampnv.h"


#ifndef FALSE
#define FALSE 0
#endif


int
main(argc, argv)
   int                 argc;      /* Count of arguments received      */
   char               *argv[];    /* Array of arguments received      */
{
   char               *api;
   int                 rc = SCA_OK,
                       i;
   SCAenv              env;
   struct SAMP_HNDL    handle;    /* Our sample's handle              */
   static char         name[] = "middle";

   /*-------------------------------------------------------------------+
    |  Initialize our head node                                         |
    +-------------------------------------------------------------------*/
   handle.head.left     =
   handle.head.right    =
   handle.head.parent   = NULL;
   handle.head.value    = NULL;
   handle.head.name     = name;
   handle.head.name_len = sizeof(name) - 1;
   handle.config        = PROMPT_ALWAYS;
   handle.config_file   = CONFIG_FILE;
   handle.file_loaded   = FALSE;


   /*-------------------------------------------------------------------+
    |  Parse the command line.                                          |
```

```
    +-------------------------------------------------------------------*/
    for (i = 1; i < argc; i++)
    {
        if (strncmp(argv[i],"-c",2) == 0)
            handle.config_file = argv[++i];
        else
            printf("Extra argument %s ignored.\n", argv[i]);
    }


   /*-------------------------------------------------------------------+
    |  Initiate the configuration process.                              |
    +-------------------------------------------------------------------*/
   api = "sasSQLinitialize";
   rc  = sasSQLinitialize(&env,(void *)&handle,usergnv,usersnv);
   if (rc == SCA_OK)
   {
      printf("\nConfigure data sources for qsas:");
      printf("\n (enter c to cancel w/o saving, e to end w/saving)\n");
      api = "sasSQLconfigure";
      rc  = sasSQLconfigure(env);
      (void)sasSQLterminate(env);
   }

   if (rc)
   {
      if (rc == SCA_SETUP_CANCEL)
         printf("\nChanges cancelled at your request.\n");
      else if (rc == SCA_SETUP_GENERR)
         ;
      else if (rc == SAMPNV_NOFILE)
         printf("\nData source definition file %s not found\n", handle.config_file);
      else if (rc == SAMPNV_NOAUTH)
         printf("\nYou are not allowed to write data source definition file %s\n",
                handle.config_file);
      else
         printf("\n%s: returned rc=%d\n", api, rc);
   }

   return(rc);
}
```

Error Codes and Messages

Return Codes

Return Code	Return-Code Mnemonic	Description
0	SCA_OK	The function completed successfully.
1	SCA_ERROR_API	A SAS software error was received.
2	SCA_ERROR_MEM	Memory could not be allocated.
3	SCA_ERROR_PARM	The input parameter is not valid.
4	SCA_ERROR_SEQ	An SQL library function call was not in sequence.
5	SCA_ERROR_SERVER	The server is not defined.
6	SCA_ERROR_COMM	An access method error occurred. Use the sasSQLreasonCode() function to determine the reason code.
7	SCA_ERROR_TABLE	The specified table was not found.
8	SCA_ERROR_COLUMN	The specified column was not found.
9	SCA_ERROR_CHARS	The data source name contains an illegal character.
10	SCA_ERROR_INTBUFF	An internal error occurred in sasSQLconnect() or sasSQLconfigure(). Please report the error to SAS Institute.
11	SCA_WARN_EOF	End of file was reached.
12	SCA_WARN_DISC	The sasSQLterminate() function disconnected active connections.
13	SCA_WARN_NOEXEC	The sasSQLexecute() function is not allowed for this connection, or an error occurred when the SAS SQL processor tried to execute the SQL statement.
14	SCA_WARN_API	A warning was issued by SAS software.
15	SCA_ERROR_LIBREF	The libref contains too many characters. The libref can contain a maximum of eight characters.
21	SCA_ERROR_DESCERR	Either the sasSQLdescribe() or sasSQLfetch() function was called after a sasSQLdescribe() error occurred.
101	SCA_CHAR_TRUNC	Data in a char variable was truncated.
102	SCA_NUM_TRUNC	Data in a num variable was truncated during conversion.
300	SCA_SETUP_END	Configuration setup ended and known values were saved.
301	SCA_SETUP_CANCEL	Configuration setup was canceled without saving data source information.
399	SCA_SETUP_GENERR	An error occurred and a message was displayed by the NV routines.

Reason Codes		
Reason Code	**Reason-Code Mnemonic**	**Description**
-1001	AM_ERROR_NOTINITED	Access method is not initialized.
-1002	AM_ERROR_INITED	Access method is already initialized.
-1003	AM_ERROR_REENTERED	Access method was re-entered illegally.
-1004	AM_ERROR_LICENSE	The facility is not licensed.
-1005	AM_ERROR_NOSERVER	No server was found.
-1006	AM_ERROR_BADENV	An environment argument is faulty.
-1007	AM_ERROR_BADCON	A connection argument is faulty.
-1008	AM_ERROR_BADADDRESS	The server address is faulty.
-1009	AM_ERROR_BADSERVICE	The service is faulty.
-1010	AM_ERROR_CLOSED	The connection is closed.
-1011	AM_ERROR_NOCONNECT	There is no connection.
-1012	AM_ERROR_BADPARM	A parameter is incorrect.
-1013	AM_ERROR_MEMORY	There is no memory.
-1014	AM_ERROR_NOTIMP	The function was not implemented.
-1015	AM_ERROR_NETFAIL	A network failure has occurred.
-1016	AM_ERROR_NOTCOMP	The request was not completed.
-1017	AM_ERROR_TRUNC	Data truncation occurred.
-1018	AM_ERROR_SASTIMEOUT	SAS timeout was started.
-1019	AM_ERROR_SECURITY	Userid and password security failure occurred.
-1020	AM_ERROR_INTR	The connection was interrupted.
-1021	AM_ERROR_SHELLFAIL	ShellExecute failed.
-1022	AM_ERROR_BADHEADER	The header was incorrect.
-1023	AM_ERROR_BADPROT	Access method protocol was incorrect.
-1024	AM_ERROR_ERRNO	An access method error occurred.
-1025	AM_ERROR_CONNREF	The connection was refused.
-1026	AM_ERROR_ENCR	An encryption error occurred.

Glossary

[A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M]

[N] [O] [P] [Q] [R] [S] [T] [U] [V] [W] [X] [Y] [Z]

A

access method

the communications protocol that the SAS SQL Library for C uses to exchange data with a SAS server. The SAS SQL Library for C uses the TCP/IP access method.

administrator-defined server library

a server library that is pre-defined to the server by a server administrator. An administrator-defined library remains defined to the server even when no users are accessing it. The administrator determines what the library's libref is going to be on the server's session.

All administrator-defined server libraries are automatically available through any data source that specifies the server; therefore, you do not have to explicitly specify the libraries in your data source definition.

See also user-defined server library.

API

See application programming interface.

application programming interface (API)

a set of software functions that facilitate communication between applications and other kinds of programs or services.

attribute

a property of a data source entity. An attribute consists of three parts: a name, a description, and a value.

C

CGI

See Common Gateway Interface .

client

a computer or application that requests services, data, or other resources from a server.

Common Gateway Interface

a standard for external gateway programs to communicate with information servers such as a World Wide Web HTTP server.

D

database management system (DBMS)

an integrated software application that enables you to create and manipulate data in the form of databases.

data file

See SAS data file.

data set

See SAS data set.

data source

a specification of a SAS/SHARE server and either the server libraries or the DBMS that a client application can access through the server.

See Defining a Data Source for information about defining data sources.

data view

See SAS data view.

DBMS

See database management system.

E

engine

a part of SAS software that reads from or writes to a file. Each engine enables SAS software to access files that exist either in a particular format or on a separate system. The two types of engines are library engine and view engine.

entity

a data source, SAS/SHARE server, or server library as specified in a data source definition.

L

library

1. See SAS data library or server library.
2. a collection of items. The SAS SQL Library for C is a collection of C functions that enables you to use SQL to access data in SAS data sets and DBMSs.

library engine

an engine that accesses groups of files and puts them into the correct form for processing SAS utility windows and procedures. A library engine also determines the fundamental processing characteristics of a SAS data library, presents lists of files for the library directory, and supports view engines.

libref

a library reference name. The logical name for a SAS data library . The libref can be up to eight 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.

S

SAS/ACCESS software

software that provides an interface between SAS software and DBMSs.

SAS data file

relational tables with columns (or variables) and rows (or observations). The SAS data file structure can have many of the characteristics of a database management system, including

indexing, compression, and password protection.

SAS data library

a collection of one or more SAS files that are recognized by SAS software. On some hosts, the SAS data library is a single file. On most hosts, however, a SAS data library is a directory. A single SAS data library can contain an unlimited number of SAS files.

Each SAS data library has two names:

- a physical name. The physical name of the library fully identifies the directory, file type or file mode, or host system data structure that contains the data sets. The name must conform to the rules for naming files within your host system.

- a logical name (libref). The logical name enables you to provide SAS software with a single identifier for a group of data sets.

See also server library.

SAS data set

a file that SAS software can access as if it were a physical object. SAS data sets consist of the following components:

- data values that are stored in tabular form
- a descriptor portion that defines the types of data to SAS software.

SAS data sets have two forms: data files and data views.

SAS data view

definitions or descriptions of data that reside elsewhere. The data views enable you to use SAS software to access data from many different sources, including flat files, VSAM files, DBMS structures, and native SAS data files.

SAS server

a SAS procedure that runs in its own SAS session and serves users by controlling and processing input and output requests to one or more server libraries or DBMSs.

Your client application reads and writes data through a SAS server instead of directly reading from and writing to a SAS server library or DBMS.

The two types of SAS servers are

- SAS/CONNECT servers, which are single-user servers
- SAS/SHARE servers, which are multi-user servers.

SAS/SHARE*NET software

the licensable right to send requests to a SAS/SHARE server from a client that is not a SAS application.

SAS/SHARE software

software that provides a multi-user client/server environment for SAS software. The multi-user server provides concurrent update access to SAS data for local and remote users across the enterprise. The server also provides remote users with low-overhead connectivity for reading SAS data.

server library

a SAS data library that is defined to a SAS server. Libraries are defined to a SAS server in two ways:

- by an administrator (administrator-defined server library)
- by the user (user-defined server library).

SQL

See Structured Query Language.

Structured Query Language (SQL)

the standardized, high-level query language used in relational database management systems to create and manipulate database management system objects.

T

TCP/IP

an abbreviation for a pair of networking protocols. Transmission Control Protocol (TCP) is a standard protocol for transferring information on local area networks such as Ethernets. TCP ensures that process-to-process information is delivered in the proper order. Internet Protocol (IP) is a protocol for managing connections between hosts. IP routes information through the network to a particular host, and fragments and reassembles information in host-to-host transfers.

U

user-defined server library

a server library that a user defines. A user-defined server library is dynamically defined to the server when a user accesses it. The library remains defined only while at least one user is accessing it.

The user specifies the physical files that make up the user-defined library. The user also specifies a libref for the library. Use this libref as the high-level qualifier for the library's table names.

To access a user-defined server library, you must explicitly specify it in your data source definition.

See also administrator-defined server library.

V

view engine

an engine that enables SAS software to process SAS data views . A view engine performs in a transparent manner. The SAS native view engine processes views that contain an SQL description of data. The SAS/ACCESS interface view engines are view engines that enable users to access data that are on SAS/ACCESS-supported DBMSs.
