Mini SQL Module

An Ember Module for Accessing Mini SQL Databases

Users Guide and Reference

Version 2.0

Feb 2017

Hughes Technologies

Although best efforts have been made to ensure the completeness and accuracy of the material presented in this document, Hughes Technologies Pty Ltd does not warrant the correctness of the information provided herein.

This software product is distributed under a dual-license scheme that provides the user with a choice of licenses. This software may be used either under the terms of the GNU Public License (GPL), or the Ember OEM License. A copy of both the GPL and the OEM license is included within the software distribution. The GPL is also available from the GNU Project's web site at http://www.gnu.org. The Ember OEM License is available from the Hughes Technologies web site at http://www.Hughes.com.au.

This document was designed to be printed on a duplex (double sided) device.

Preface

Intended Audience

This document describes the Ember implementation of the Mini SQL application programming interface. It is not intended to be a reference for mSQL nor is it a general purpose SQL tutorial. It is assumed that the reader is either familiar with mSQL or has access to mSQL's reference manual.

Document Conventions

This manual has been designed to be printed on US Letter paper. While many parts of the world utilise the A4 paper size (Australia included), it is not possible to print A4 formatted documents on US Letter paper without loss of information. However, printing of US Letter formatted documents on A4 will result in a correct representation of the document with somewhat larger margins than normal.

Throughout this manual, parts of the text have been flagged with the symbol that appears in the margin opposite this paragraph. Such pieces of text are viewed as being important. The reader should ensure that paragraphs marked as important are read even if the entire manual section is only being skimmed. Important sections will include information such as tips on improving the performance of your applications, or areas where mistakes are commonly made.

Contact Information

Further information about Ember and its related software can be found on the Hughes Technologies Web site. The web site includes the latest version of Ember, documentation, support, example software, references to customer sites, and much more. Our web site can be found at

http://www.Hughes.com.au

This page left intentionally blank

Table of Contents

Introduction

The Mini SQL module is a library of routines for communicating with a Mini SQL database server. The functions provided by this module mimic the functions provided by the mSQL C API. Please see the mSQL documentation for more information about the API and it's use. This manual provides a rough overview and concentrates on the Ember implementation of the API rather than the API itself.

Along with a detailed description of each API call, the document provides an overview for writing mSQL capable scripts, and also includes an example program.

Overview

Interacting with an mSQL database engine is usually a simple and painless task. The database engine runs as a separate "daemon" process on a server and the client application, in this case our Ember script, connects to the server to communicate with it. Communications can be either via a local UNIX socket (fast and safe) or via a TCP connection over a network (discouraged).

The basic design of a script that accesses an mSQL database is as follows

- Connect to the database engine
- Select the database you wish to access
- Submit the query
- If the query was a SELECT query, process the returned data
- Disconnect from the mSQL server

To connect to the database server you must make a call to the msqlConnect( ) function. If you wish to use a local connection then do not pass an argument to the function call. If you wish to connect via TCP/IP to a server running on a remote machine then pass the hostname or IP Address of the server as the argument. The function will return you a handle that you must store and use for later interaction with the server. If the connection attempt failed, a value of –1 will be returned and the $ERRMSG variable will contain a textual description of the problem. If all went well, a zero value is returned.

Once you are connected to the server you must inform it of the database you wish to access. To do so, simply call msqlSelectDB($sock, $db) where $sock is the handle returned by msqlConnect( ) and $db contains the name of the database you wish to access. If you are unable to access the requested database, a value of –1 will be returned and $ERRMSG will be set. Otherwise, a zero value will be returned.

SQL queries are submitted to the engine using the msqlQuery($sock, $query) function, where $sock is the connection handle and $query contains the SQL query. If the query fails, a value of –1 is returned and $ERRMSG is set. Otherwise the query will return a non-negative value indicating the number of table rows affected by the query (i.e. returned or edited or inserted etc).

If the query returned some data (i.e. it was a select query), then you must store the result if you wish to access it. The result is stored using the aptly named msqlStoreResult( ) function. No arguments are required but you must save the handle returned by the function. That handle is a query result handle.

A row of result data is accessed using the msqlFetchRow($result) function. On it's first invocation, the function will return the first row of result data. Each subsequent function call will move on to the next row of data. Each row is returned as an array of data fields. The order of the array elements reflects the order of the fields specified in the query. When the result data is exhausted, the function returns an empty array.

When the result data is no longer required it must be freed as it consumes valuable memory. To free the result you simply call msqlFreeResult($result)

You may submit as many queries as you like to the database during the execution of your script. Once you have performed all the queries you need, you should close the database connection. To do so, simply call msqlClose($sock) where $sock is the database handle returned by msqlConnect( ).

API Reference

msqlConnect ( )

```
int msqlConnect ( host )
        text host
```

msqlConnect() connects to the mSQL server on the specified host. If no host is specified it connects to the local mSQL server. A negative return value indicates an error. Error messages are returned via the $ERRMSG variable. Note: if the database is located on the same machine as the script then it is safer and much faster to connect via a local socket. To do so simply omit the host argument (i.e do not pass any arguments to the function call)

Example :

```
$sock = msqlConnect("research.Hughes.com.au");
if ($sock < 0)
{
        echo("ERROR : $ERRMSG\n");
}
```

msqlClose ( )

```
msqlClose ( sock )
        int sock
```

msqlClose() closes a connection made using msqlConnect(). The connection should be closed once the application is finished using the database.

msqlSelectDB ( )

```
int msqlSelectDB ( sock , db )
        int sock
        text db
```

msqlSelectDB() tells the mSQL server which database you wish to use. Pass it the server socket and the name of the desired database as its arguments. A negative return value indicates that an error was encountered and that the error message is available in the $ERRMSG variable.

Example :

```
if (msqlSelectDB($sock,"my_db") < 0)
{
        echo("ERROR : $ERRMSG\n");
}
```

msqlQuery ( )

```
int msqlQuery ( sock , query )
        int sock
        text query
```

msqlQuery() submits a query to the mSQL server connected to the specified socket. The query may be any valid SQL statement including INSERT, DELETE, UPDATE, SELECT, CREATE and DROP queries. A negative return value indicates an error. The cause of the error is described in the $ERRMSG variable.

Example :

```
if (msqlQuery($sock, "select * from foo") < 0)
{
        echo("ERROR : $ERRMSG\n");
}
```

msqlStoreResult ( )

```
msqlStoreResult ( )
```

msqlStoreResult() stores any data that was a result of the previous query and provides a result handle for later use. The result handle must be stored by the script.

Example :

```
$res = msqlStoreResult( );
```

msqlFreeResult ( )

```
msqlFreeResult ( res )
        int res
```

msqlFreeResult() frees any memory allocated to the specified result. If result handles are not freed once they are no longer required, the script will waste valuable memory.

Example :

```
msqlFreeResult( $res );
```

msqlFetchRow ( )

```
msqlFetchRow ( res )
        int res;
```

msqlFetchRow() returns a single row of the data stored in the specified result. The first call to msqlFetchRow( ) will return the first row of result data. Each subsequent call will return the next row until all result data has been returned. When no further data is available, an empty array is returned.

Example :

```
$row = msqlFetchRow($res);
while( # $row > 0)
{
        echo("Field 0 is $row[0]\n");
        $row = msqlFetchRow($res);
}
```

msqlDataSeek ( )

```
msqlDataSeek ( res , location )
        int res
        int location
```

msqlDataSeek() allows you to move the data pointer within the result table. Specifying a location of 0 will rewind the result to the start of the result table. The next call to msqlFetchRow() would return the first row of the result table again. An attempt to seek beyond the end of the table or before the start of the table will result in an error. In such a situation a negative value is returned and the data pointer is left unchanged.

Example :

```
msqlDataSeek( $res, 0);
```

msqlListDBs ( )

```
msqlListDBs ( sock )
        int sock
```

msqlListDBs() returns an array of the names of the databases available on the specified server.

Example :

```
$dbs = msqlListDBs($sock);
$index = 0;
while ($index < # $dbs)
{
        printf("Database = %s\n", $dbs[$index]);
        $index = $index + 1;
}
```

msqlListTables ( )

```
msqlListTables ( sock )
        int sock
```

msqlListTables() returns an array of the names of all the tables available in the current database of the specified server. The database must have been previously specified using the msqlSelectDB( ) function.

Example :

```
msqlSelectDB($sock, “my_db”);
$tabls = msqlListTables($sock);
$index = 0;
while ($index < # $tabls)
{
        printf("Table = %s\n", $tabls[$index]);
        $index = $index + 1;
}
```

msqlInitFieldList ( )

```
msqlInitFieldList ( sock ,  table )
        int sock
        char *table
```

msqlInitFieldList() generates an internal result handle containing details of all the fields in the specified table of the currently selected database database. The result handle is stored within the API library and is not returned to the program. It is held as a static variable inside the mSQL module and further calls to msqlInitFieldList() will free the result. The contents of the result handle are accessed using the API functions outlined below.

Example :

```
msqlSelectDB($sock, “my_db”);
msqlInitFieldList($sock , “my_table”);
```

msqlListField ( )

```
msqlListField ( )
```

msqlListField() returns an array of information about a single field of the current field list result table. The result table must have been previously created by calling the msqlInitFieldList() function. The elements of the array contain the following information.

Element	Description
0	Field Name
1	Table Name
2	Type
3	Length
4	Flags

Example :

```
msqlSelectDB($sock, “my_db”);
msqlInitFieldList($sock,"my_table");
$field = msqlListField( );
while( # $field > 0)
{
        $name = $field[0];
        $table = $field[1];
        echo("Name $name]\n");
        $field = msqlListField( );
}
```

msqlFieldSeek ( )

```
msqlFieldSeek ( location )
        int location
```

msqlFieldSeek() acts upon the internal result of a call to msqlInitFieldList() in the same way msqlDataSeek() acts upon the result of a call to msqlStoreResult(). It allows you to move the internal result pointer to the specified location.

Example :

```
msqlSelectDB($sock, “my_db”);
msqlInitFieldList($sock,"my_table");
msqlFieldSeek( 3 );
$field = msqlListField( );
echo(“The name of the third field is $field[0]\n”);
```

msqlNumRows ( )

```
int msqlNumRows ( res )
        int res
```

msqlNumRows() returns the number of rows contained in the result handle res. The result handle must have been the result of a call to msqlStoreResult( )

Example :

```
msqlQuery($sock, "select * from foo");
$res = msqlStoreResult();
printf("There are %d rows in foo\n",
msqlNumRows($res);
```

msqlEncode ( )

msqlEncode (string)
text string

msqlEncode() is passed a string value that may contain characters that can cause errors in mSQL query strings (such as the ' character in text values). It returns a modified version of the string with all such characters escaped.

Example :

```
$name = "O'Reilly";
$newName = msqlEscape($name);
```

Example

```
/*
** Connect to the local database server
*/
$sock = msqlConnect();
if ($sock < 0)
{
        echo("mSQL Connect failed :$ERRMSG\n");
        exit(1);
}

/*
** Select the test database
*/
if (msqlSelectDB($sock, "test") < 0)
{
        echo("Couldn't select DB : $ERRMSG\n");
        exit(1);
}

/*
** Retrieve a list of staff names
*/
$query = "select first_name, last_name from staff
        order by last_name, first_name";
if (msqlQuery($sock, $query) < 0)
{
        echo("Query failed : $ERRMSG\n");
        exit(1);
}

/*
** Store the result and output the info
*/
$res = msqlStoreResult();
$row = msqlFetchRow($res);
while( # $row > 0)
{
        echo("Name = $row[0] $row[1]\n");
        $row = msqlFetchRow($res);
}

/*
** Clean up
*/
msqlFreeResult($res);
msqlClose($sock);
```