Study of a New Database Engine
mSQL/W3-mSQL and its Application for the Web

Technical Report 97-01

Guirong Qin

Submitted to Committee Members

Dr. W. Homer Carlisle (Chair)
Dr. Kai-Hsiung Chang
Dr. Loretta A. Moore

Department of Computer Science & Engineering
Auburn University
Auburn University, Alabama 36849-5347

May 12, 1997
Abstract

mSQL is a light-weighted database engine, which supports a subset of ANSI SQL. mSQL is easy to use and provides fast access to stored data with low memory requirements while still meeting many needs of the real world. It can run on many platform (UNIX, Linux, Windows, etc.). mSQL 2.0 is bundled with a couple of tools to aid in the development of applications. W3-mSQL 2.0 is included in the mSQL distribution as a standard tool and provides a complete scripting language -- Lite script, with full access to mSQL API within the HTML tag. This tool can be used to develop sophisticated GUI based applications. W3-mSQL also provides an enhanced and flexible authentication system via a built-in authorization scheme – w3-auth. This project extensively explores the features of mSQL and W3-mSQL, from language specification, application programming interface, system security to script language and module, and presents a practical, “real-world” application using mSQL/W3-mSQL: student registration and course management forms for the web.
Contents

I. Introduction

II. mSQL 2.0 Installation and Configuration

2.1 Installation
2.2 Runtime configuration

III. mSQL Database Package Specification

3.1 mSQL language specification
3.2 mSQL programming API
3.3 Standard mSQL programs and utilities

IV. W3-mSQL Interface Package Specification

4.1 W3-mSQL -- WWW interface to mSQL
4.2 System security -- access control and authentication
4.3 Lite -- mSQL's own scripting language
4.4 Lite's standard module, mSQL module and custom module

V. Application of mSQL/W3-mSQL -- Development of Student Registration and Course Management Forms for the Web

5.1 Database construction and development
5.2 Development of dynamic and interactive Registration/Management forms for the web

VI. Limitations of mSQL/W3-mSQL and its Future Development

References
I. Introduction

From time to time, lists appear defining the "hot technologies" of the Internet\textsuperscript{6}. Nearly all those lists include "database access" as one of their entries. Yet database access continues to be among the most mysterious techniques of the web.

For many applications the website must allow user to search much information. Database technology is ideally suited for this sort of task\textsuperscript{6}. Database technology comes in various flavors, each suitable for a particular class of problems.

Flat files are considered as the simplest database. A query could be answered with a brute force search of the file. But this search strategy and disk space utilization are inefficient. Also database maintenance is difficult.

The indexed sequential access method (ISAM) can significantly improve this kind of search. ISAM involves balancing disk blocking and track size to build a partial index. Access to the records is through a primary index (memory), then a secondary index (with a single disk fetch), then to a block (requiring one more disk fetch), which must be searched sequentially. If file changes regularly, the problem becomes more complex. ISAM relies upon the file being stored on disk in the same order as the indexed field. When records are added or deleted from the database, the file and its indexes must be rebuilt.

Many real-world problems get more complex than a single table and index can handle. When databases and queries become complex, most people turn to the relational database management system (RDBMS). Many vendors offer an RDBMS solution; Oracle, Sybase, and Informix are among the best known. Most RDBMSs are accessed using the Structured Query Language, or SQL.

High-end RDBMS products routinely cost tens of thousands of dollars. For many purposes on the web, a much simpler product will suffice. High-end products are often used to produce reports, which may take many minutes to run. Most web queries need to complete within a few seconds to satisfy user's real-time requirements. To fill this need David Hughes wrote miniSQL\textsuperscript{6}.

Mini SQL, or mSQL, is a lightweight database engine designed to provide fast access to stored data with low memory requirements and runs on a wide variety of UNIX systems.
mSQL was developed by David J. Hughes at Bond University, Australia and can be used without cost by universities, non-commercial research groups, and not-for-profit organizations. It is also offered under a commercial license which costs around US $200. As its name implies, mSQL offers a subset of SQL as its query interface[3]. Although it only supports a subset of SQL (no views, subqueries), everything it supports is in accordance with the ANSI SQL specification[3].

mSQL version 2 is the second generation of the mSQL product. mSQL 1.x has been available in various forms since June 1994 and has undergone many enhancements to become a very popular and stable database system for small databases. mSQL 2.0 goes beyond the initial design goals of mSQL 1 and provides functionality suited to larger applications. It does this without reducing its performance on the simple tasks that mSQL 1 handled so well[11]. The result is a lightweight database system suited to most data storage requirements.

mSQL 2.0 is bundled with a couple of tools to aid in the development of applications. W3-mSQL 2.0, the second generation WWW interface package, is included as a standard tool. The new W3-mSQL code provides a complete scripting language, Lite, with full access to the mSQL API, within an HTML tag. This tool can be used to develop sophisticated GUI based applications that are platform independent and available as shared resources on a network[11]. Along with the mSQL API, a library of nearly 60 other functions, including file I/O, strings handling and date/time manipulation are available to the scripts within a W3-mSQL enhanced web page.

This project extensively explores the features of mSQL and W3-mSQL, from language specification, application programming interface, system security to script language and module. Finally, it presents a practical, “real-world” application using mSQL/W3-mSQL: student registration and course management forms for the web.
II. mSQL 2.0 Installation and Configuration

2.1 Installation

To install mSQL server to UNIX-like platform, first we need to get a latest version (now mSQL-2.0-B7.1) of mSQL package via anonymous ftp from ftp://bond.edu.au/pub/Minerva/mssql. Then unpack the package with gzip -cd (or gunzip) and tar xvf -. The version of mSQL for Windows platform is now still 1.0.16 and W3-mSQL 1.0 was released in a separated package. Once the package is unpacked, then follow the instruction below step by step[1]:

(1) In the root (or any other specified) directory of the distribution type "make target". This will create a "symbolic link tree" for the particular platform under the targets/directory. For example, on a Linux machine it may end up with a replica of the source code directory in targets/Linux-2.0.30-i586/.

(2) Change directory to the target directory for this machine, e.g., targets/Linux-2.0.30-i586/

(3) Run the setup utility (Note: use ./setup to ensure getting the mSQL setup utility and not a system utility).

(4) After setup has completed, you may need to edit a couple of default configuration values in file site.mm. This is required to specify the installation directory if the user is non-root. When everything is OK type "make all".

(5) If the compilation of mSQL completes properly, type "make install". This will install all the mSQL related files under /usr/local/Hughes (or whatever directory it was set as the install directory in site.mm).

(6) Edit /usr/local/Hughes/msql.conf to see if everything is OK. The only thing it might change in a normal installation is the username of the mSQL_User or the Admin_User. By default, mSQL is configured to run as a user called 'msql' with root being the admin user (i.e. the one who can create databases, shutdown the server etc.).

(7) Ensure that the installation directory (and everything under it) is owned by the user that mSQL will run as (e.g. the msq1 user). To do this, issue a command like "chown -R msql /usr/local/Hughes". Substitute the correct username and installation directory if it is running as a non-default installation.

5
(8) If you wish to use the WWW interface package, W3-mSQL, first you should start the mSQL server (by running /usr/local/Hughes/bin/msqld2d & ) and then execute misc/setup_www (or /usr/local/Hughes/bin/lite misc/setup_www). This will setup a couple of databases and install the Hughes Technologies related web page graphics etc. into the web tree. You will need to be running the mSQL server as the mSQL Root_User as this creates several databases. You may also need to run the server as a specific user to modify your web tree. Note: setup_www expects to find Lite installed in the default location. If the Lite was installed into anywhere other than /usr/local/Hughes, edit the related lines of the setup_www script to reflect the location of the Lite binary.

(9) If you want to use W3-mSQL you will need to copy w3-auth and w3-msql from /usr/local/Hughes/bin to the cgi-bin directory (or provide a symbolic link to them from the cgi-bin). It may need root access to perform this step.

Note: To use the W3-mSQL authentication/access control features you will need a web server that passes authentication headers to CGI scripts. The CERN server does this and a patch is available from Hughes Technologies to add this feature to the Apache server. The Apache patch is available for both 1.1 and 1.2 versions of Apache at http://Hughes.com.au

2.2 Runtime configuration

mSQL 1.x offered several configuration options, including such details as the user the server should run as, the location of the TCP and UNIX sockets for client/server communications, the location of the database files, etc. The problem with configuring mSQL 1.x was that all these details were hard-coded into the software at compile time. Once the software was compiled and installed you couldn’t easily change those settings.

To overcome this problem, mSQL 2.0 utilizes an external run-time configuration file for definition of all these values[11]. The file is called msq1.conf and is located in the installation directory (usually /usr/local/Hughes). An application can choose to use a different configuration file by calling the new msq1LoadConfigFile( ) API function. All standard mSQL applications and utilities provide a command line flag, -f Conffile, that allows to specify a non-standard configuration file. When an application first calls the mSQL API library, a check is made to see if a configuration file has been loaded via a call to the msq1LoadConfigFile( ) function. If no such call has been made, the API library loads the default config file. Any values that are specified in that file will override the normal operating parameters used by mSQL.
III. mSQL Database Package Specification

The mSQL distribution includes the mSQL server, client program, a C programming interface for client database, and several tools. User contributed software is also available including interfaces to mSQL from Perl, Tcl, Java and Python, WWW interfaces, a Windows port of the client library and much more.

3.1 mSQL language specification[1]

The mSQL language offers a significant subset of the features provided by ANSI SQL. It allows a program or user to store, manipulate and retrieve data in table structures. It does not support some relational capabilities such as views and nested queries.

The mSQL 2.0 database provides for storage of data types: UINT, INT, REAL, CHAR, TEXT, DATE, TIME and MONEY. It supports the CREATE, DROP, INSERT, SELECT, DELETE and UPDATE operations.

(1) The CREATE Clause

CREATE clause can be used to create tables, indices, and sequences. It cannot be used to create other definitions such as views. The syntax of CREATE clause is as:

```
CREATE TABLE table_name (col_name col_type [not null] [,col_name col_type [not null]] ...)  
CREATE [UNIQUE] INDEX index_name ON table_name(field_name [,field_name] ...)  
CREATE SEQUENCE ON table_name [STEP step_val] [VALUE initial_val]
```

mSQL version 2.0 has removed support for the primary key construct within the table creation syntax although the same result can be achieved with an index. A compound index (more than one field) would improve performance. Sequences provide a mechanism via which a sequence value can be maintained by the mSQL server.

(2) The DROP Clause

The DROP clause is used to remove a definition from the database. It is most commonly used to remove a table from a database but can also be used for several other constructs. In mSQL 2.0, it can be used to remove the definition of an index, a sequence, or a table. The syntax of the DROP clause are given below:
DROP TABLE table_name
DROP INDEX index_name FROM table_name
DROP SEQUENCE FROM table_name

(3) INSERT Clause

Unlike ANSI SQL, mSQL cannot nest a SELECT within an INSERT (i.e. you cannot insert the data returned by a select). If the query does not specify the field names they will be used in the order they were defined, and the statement must specify a value for every field. The Syntax is:

    INSERT INTO table_name [( column [, column ]** )] VALUES (value [, value]** )

(4) SELECT Clause

The SELECT offered by mSQL lacks some of the features provided by the standard SQL specification. Development of mSQL 2 is continuing and some of this missing functionality will be made available in the next beta release. At this point in time, mSQL's SELECT does not provide: Nested SELECTs, Implicit functions (e.g. count(), avg() ). It does however support: Joins - including table aliases, DISTINCT row selection, ORDER BY clauses, Regular expression matching, Column to Column comparisons in WHERE clauses, Complex conditions. The formal definition of the syntax for mSQL's SELECT clause is:

    SELECT [table.]column [, [table.]column ]**
    FROM table [ = alias] [, table [ = alias] ]**
    [ WHERE [table.]column OPERATOR VALUE
    [ AND | OR [table.]column OPERATOR VALUE]** ]
    [ ORDER BY [table.]column [DESC] [, [table.]column [DESC] ]

OPERATOR can be <, >, =, <=, >=, <=>, LIKE, RLIKE or CLIQUE. VALUE can be a literal value or a column name. WHERE clauses may contain '(' , ')' to nest conditions.

mSQL provides three regular expression operators for use in WHERE comparisons. The standard SQL syntax provides a very simplistic regular expression capability that does not provide the power nor the flexibility UNIX programmers or users will be accustomed to. mSQL supports the "standard" SQL regular expression syntax, via the LIKE operator, but also provide further functionality if it is required. The available regular expression operators are: LIKE - the standard SQL regular expression operator. CLIQUE - a standard LIKE operator that ignores case. RLIKE - a complete UNIX regular expression operator.
(5) DELETE Clause

The SQL DELETE construct is used to remove one or more entries from a database table. The selection of rows to be removed from the table is based on the same WHERE construct as used by the SELECT clause. The syntax for mSQL's DELETE clause is

```
DELETE FROM table_name
WHERE column OPERATOR value
[ AND | OR column OPERATOR value ]**
```

OPERATOR can be <, >, =, <=, >=, <>, LIKE, RLIKE, or CLIKE

(6) UPDATE Clause

The SQL UPDATE clause is used to modify data that is already in the database. The operation is carried out on one or more rows as specified by the WHERE construct. The value of any number of fields on the rows matching the WHERE construct can be updated. mSQL places a limitation on the operation of the UPDATE clause in that it cannot use a column name as an UPDATE value (i.e. you cannot set the value of one field to the current value of another field). Only literal values may be used as an UPDATE value. The syntax supported by mSQL is

```
UPDATE table_name SET column=value [, column=value ]**
WHERE column OPERATOR value
[ AND | OR column OPERATOR value ]**
```

OPERATOR can be <, >, =, <=, >=, <>, LIKE, RLIKE or CLIKE.

3.2 mSQL programming API[1]

Included in the distribution is the mSQL API library, libmsql.a. The API allows any C program to communicate with the database engine. The API functions are accessed by including the msq1.h header file into program and by linking with the mSQL library (using -lmsq1 as an argument to C compiler). The library and header file are installed by default into /usr/local/Hughes/lib and /usr/local/Hughes/include respectively.

Like the mSQL engine, the API supports debugging via the MYSQL_DEBUG environment variable. Three debugging modules are currently supported by the API: query, api, and malloc. Enabling "query" debugging will cause the API to print the contents of queries as they are sent to the server. The "api" debug modules causes internal information, such as
connection details, to be printed. Details about the memory used by the API library can be obtained via the "malloc" debug module. Information such as the location and size of malloced blocks and the addresses passed to free() will be generated. Multiple debug modules can be enabled by setting MYSQL_DEBUG to a colon separated list of module names. For example setenv MYSQL_DEBUG api:query.

Query or Schema related functions in mSQL API are almost same as those in Lite’s mSQL modules.

3.3 Standard mSQL programs and utilities

mSQL embodies a simple client-server architecture. The server maintains the databases, receives commands from client programs and sends replies. The standard client programs are:

(1) The monitor - msql

Usage:  **msql [-h host] [-f confFile] database**

Options -h Specify a remote hostname or IP address on which the mSQL server is running. The default is to connect to a server on the localhost using a UNIX domain socket rather than TCP/IP (which gives better performance).
-f Specify a non-default configuration file to be loaded. The default action is to load the standard configuration file located in INST_DIR/msql.conf (usually /usr/local/Hughes/msql.conf)

Description The mSQL monitor is an interactive interface to the mSQL server. It allows you to submit SQL commands directly to the server. Any valid mSQL syntax can be entered at the prompt provided by the mSQL monitor. Control of the monitor itself is provided by 4 internal commands. Each command is comprised of a backslash followed by a single character. The available command are: \q – Quit, \g -- Go (Send the query to the server), \e -- Edit (Edit the previous query), \p -- Print (Print the query buffer).

(2) Schema viewer - relshow

Usage      **relshow [-h host] [-f confFile] [database [rel [idx] ] ]**

Option -h Specify a remote hostname or IP address on which the mSQL server is running. The default is to connect to a server on the localhost using a UNIX domain socket rather than TCP/IP (which gives better performance)
-f Specify a non-default configuration file to be loaded. The default action is to
load the standard configuration file located in INST_DIR/mysql.conf (usually
/usr/local/Hughes/mysql.conf).

Description Relshow is used to display the structure of the contents of mSQL databases.
If no arguments are given, relshow will list the names of the databases
currently defined. If a database name is given it will list the tables defined in
that database. If a table name is also given then it will display the structure of
the table (i.e. field names, types, lengths etc.).

(3) Admin program - mysqladmin

Usage mysqladmin [-h host] [-f confFile] [-q] Command

Option -h Specify a remote hostname or IP address on which the mSQL server is
running. The default is to connect to a server on the localhost using a UNIX
domain socket rather than TCP/IP (which gives better performance)
-f Specify a non-default configuration file to be loaded. The default action is to
load the standard configuration file located in INST_DIR/mysql.conf (usually
/usr/local/Hughes/mysql.conf)
-q Put mysqladmin into quiet mode. If this flag is specified, mysqladmin will not
prompt the user to verify dangerous actions (such as dropping a database).

Description mysqladmin is used to perform administrative operations on an mSQL
database server. Such tasks include the creation of databases, performing
server shutdowns etc. The available commands for mysqladmin are
create db_name -- Creates a new database called db_name
drop db_name -- Removes the database called db_name from the server.
This will also delete all data contained in the database!
shutdown -- Terminates the mSQL server.
reload -- Forces the server to reload ACL information.
version -- Displays version and configuration information about the
currently running server.

Note: most administrative functions can only be executed by the user specified in the run-
time configuration as the admin user. They can also only be executed from the host on
which the server process is running (e.g. cannot shutdown a remote server process).

(4) Data dumper - mysqldump

Option -h Specify a remote hostname or IP address on which the mSQL server is running. The default is to connect to a server on the localhost using a UNIX domain socket rather than TCP/IP (which gives better performance).
-f Specify a non-default configuration file to be loaded. The default action is to load the standard configuration file located in INST_DIR/msql.conf (usually /usr/local/Hughes/msql.conf)
-c Include column names in INSERT commands generated by the dump.
-v Run in verbose mode. This will display details such as connection results etc.

Description **msqldump** produces an ASCII text file containing valid SQL commands that recreate the table or database dumped when piped through the mSQL monitor program. The output will include all CREATE TABLE commands required to recreate the table structures, CREATE INDEX commands to recreate the indices, and INSERT commands to populate the tables with the data currently contained in the tables.

Note: msqldump does not recreate sequences at this time.

(5) Data exporter - msqlexport

Usage **msqlexport** [-h host] [-f conf] [-v] [-s Char] [-q Char] [-e Char] database table

Option -h Specify a remote hostname or IP address on which the mSQL server is running. The default is to connect to a server on the localhost using a UNIX domain socket rather than TCP/IP (which gives better performance)
-f Specify a non-default configuration file to be loaded. The default action is to load the standard configuration file located in INST_DIR/msql.conf (usually /usr/local/Hughes/msql.conf)
-v Verbose mode
-s Use the character Char as the separation character. The default is a comma.
-q Quote each value with the specified character
-e Use the specified Char as the escape character. The default is \

Description **msqlexport** produces an ASCII export of the data from the specified table. Output produced can be used as input to other programs such as spreadsheets. It has been designed to be as flexible as possible allowing the user to specify the character to use to separate the fields, the character to use to escape the separator character if it appears in the data, and whether the data should be quoted and if so what character to use as the quote character. The output is sent to stdout with one data row per line.
(6) Data importer - msqimport

Usage msqimport [-h host] [-f conf] [-v] [-s Char] [-e Char] [-c col,col...] database table

Option  
-h Specify a remote hostname or IP address on which the mSQL server is running. The default is to connect to a server on the localhost using a UNIX domain socket rather than TCP/IP (which gives better performance)
-f Specify a non-default configuration file to be loaded. The default action is to load the standard configuration file located in INST_DIR/msql.conf (usually /usr/local/Hughes/msql.conf)
-v Verbose mode
-s Use the character Char as the separation character. The default is a comma.
-e Use the specified Char as the escape character. The default is \n-c A comma separated list of column names into which the data will be inserted.
Note: there can be no spaces in the list.

Description msqimport loads a flat ASCII data file into an mSQL database table.
The file can be formatted using any character as the column separator. When passed through msqimport, each line of the text file will be loaded as a row in the database table. The separation character as specified by the -s flag, will be used to split the line of text into columns. If the data uses a specific character to escape any occurrence of the separation character in the data, the escape character can be specified with the -e flag and will be removed from the data before it is inserted.
IV. W3-mSQL Interface Package Specification

4.1 W3-mSQL -- WWW interface to mSQL

mSQL 2.0 is bundled with a couple of new tools to aid in the development of applications. W3-mSQL 2.0, the second generation WWW interface package, is included as a standard tool. The new W3-mSQL code provides a complete scripting language, with full access to the mSQL API, within an HTML tag\(^1\). This tool can be used to develop sophisticated GUI based applications that are platform independent and available as shared resources on a network. Along with the mSQL API, a library of nearly 60 other functions, including file I/O, strings handling and date/time manipulation are available to the scripts within a W3-mSQL enhanced web page.

To solve another problem associated with delivering "real" applications over the web, W3-mSQL provides an enhanced and flexible authentication system\(^1\). Any page that is accessed via W3-mSQL is subjected to the new W3-auth access scrutiny. Access can be restricted via a combination of username/passwd and requesting host. Configuration of the security system, including management of user groups, definition of secure areas, and creation of authorized users, is via a graphical interface accessed via a web page.

Access to mSQL from scripting languages has become popular and virtually all major scripting languages provide an interface to the original mSQL server. Support for script based access to mSQL becomes standard in mSQL 2.0 with the inclusion of its own scripting language\(^1\). The language, called Lite, is a stand-alone version of the language provided by W3-mSQL (i.e. the language that W3-mSQL offers inside the special HTML tags is Lite) and includes access to the mSQL API and the other functions mentioned above. Lite, as it's name implies, is a lightweight language yet provides a powerful and flexible programming environment. The syntax of the language will be very familiar to C programmers (and ESL programmers) and provides shell-like ease of use. A future release of Lite will include support for ASCII forms to provide a rapid development environment for non-graphical mSQL-based applications.

Using W3-mSQL and the embedded Lite language, you can generate HTML code "on-the-fly" in the same way you do when you write custom CGI programs. What's more, you can mix normal HTML code with W3-mSQL code so that you only need to use the CGI styled approach where you actually have to.

To facilitate the W3-mSQL extensions to normal web pages, Lite code is included in your HTML code. It is differentiated from normal HTML code by including it inside <! > tags.
As an example, a W3-mSQL version of the legendary **Hello World** program is provided below.

```html
<HTML>
  <HEAD><TITLE>Hello World from W3-mSQL</TITLE></HEAD>
  <BODY>
    <CENTER><H1>Introduction to W3-mSQL</H1><P>
      <! echo("Hello World\n"); >
    </CENTER>
  </BODY>
</HTML>
```

As you can see, there is a line of code in the middle of the HTML page, enclosed in `<! >` tags. When the page is loaded through the W3-mSQL CGI program, anything enclosed in `<! >` tags is parsed and executed as an embedded program. Any output generated by the program is sent to the user's browser. In this case, the string "Hello World" would be sent as part of the HTML page to the browser. The remainder of the page is sent to the browser unmodified. There can be any number of W3-mSQL tags within a single page and there can be any number of lines of code within a single W3-mSQL tag.

To execute the above script it must not just specify the path to the file in the URL as it would normally do. If doing that, the browser will just be sent the unprocessed HTML document. To execute the script properly it must specify a URL that executes the W3-mSQL binary and tells it to load and process the script. The W3-mSQL binary is called w3-msql and will usually be located in the /cgi-bin directory (or other specified cgi-bin directory). If the normal URL of a W3-mSQL enhanced web page is `/staff/lookup.html`, it would be loaded using the following URL:

`/cgi-bin/w3-msql/staff/lookup.html`

This URL instructs the web server to execute the W3-mSQL binary and tells it to load the `/staff/lookup.html` script file.

One thing virtually all CGI type programs have in common is that they process the contents of an HTML form. The form data is passed to the CGI program via either a GET or a POST method by the http server. It is then the responsibility of the CGI script to decipher and decode the data being passed to it. W3-mSQL simplifies this process greatly by converting any form data passed to a script into global Lite variables within the Lite Virtual Machine[^1]. These variables can then be accessed by your script code.

[^1]: Reference to additional material or explanation.
When an HTML form is defined, a field name is given to each of the elements of the form. This allows the CGI to determine what the data values being submitted actually mean. When the data is passed to W3-mSQL, the field names are used as the variable names for the global variables. Once a set of variables has been created for each form element, the values being passed to the script are assigned to the variables. This is done automatically during start-up of the W3-mSQL program.

As an example, consider the following form defined in an HTML page

```html
<FORM ACTION="/cgi-bin/w3-msql/my_stuff/test.html METHOD=POST"
<INPUT NAME=username SIZE=20>
<INPUT NAME=password SIZE=20 TYPE=PASSWORD>
<SELECT NAME=quarter>
  <OPTION VALUE=sp97>Spring 97
  <OPTION VALUE=su97>Summer 97
  <OPTION VALUE=fa97>Fall 97
</SELECT>
</FORM>
```

In the example there are three fields within the form, two text entry fields called username and password, and a menu called quarter. It has also been specified that the action for the form is to call W3-mSQL and tell it to process /my_stuff/test.html passing the form data via the POST method. When the data is submitted, the values entered for the three form fields are passed to W3-mSQL. It then creates three global variables called $username, $password and $quarter, and assigns the user's data to those variables. The values can then be accessed within the Lite script code embedded in test.html by referencing the variables.

4.2 System security -- access-control and authentication

W3-mSQL tries to address security related issues from several points of view. The most obvious security problem is the management of access to data contained in web pages. A good solution to this problem provides both authentication of the users and access control to restrict access based on where the user is located. W3-mSQL solves this problem with a built-in authorization scheme known as w3-auth.

w3-auth provides a facility by which it can restrict access to file accessed via W3-mSQL based on username/password and also by the location of the client machine. It should be noted that it only restricts access to pages processed by the W3-mSQL CGI program, not to every page on the web server.
w3-auth provides a web based interface to the management and configuration of the security policy. Access is via the w3-auth CGI program usually located at /cgi-bin/w3-auth on the web server.

As hosting of web pages by ISP's and web providers becomes the norm, it is quite probable that a single machine may house W3-mSQL based applications for several companies. If the administration and management of usernames for access to web based applications is stored in a single place, there is a potential for "clashes" in the usernames required by the various customers. For example, if both ABC Corp. and XYZ Inc. had W3-mSQL based applications on a single machine, there exists a potential that both ABC and XYZ will want to have a user called 'bill'. If both companies have to share a username/password facility then it is obvious that there can only be one user called 'bill' and that the company that requested the username second would have to be rejected (i.e. can't have two users with the same username on a single UNIX machine).

To overcome this problem, w3-auth supports multiple "namespaces" for the definition of usernames[1]. In the situation above, a namespace would be created for both ABC and for XYZ. The fact that both companies want to have a user called 'bill' is no longer a problem as one 'bill' will be in the ABC namespace and the other 'bill' will be in the XYZ namespace. W3-auth treats these users as totally different users (i.e. they can have different passwords and access levels etc.).

The concept of namespaces is used throughout W3-auth to distinguish between users and web pages "owned" by different organizations[1].

When building "real" applications with a scheme such as W3-mSQL, other security related issues become apparent[1]. Because the actual program code is embedded in the HTML code anyone wishing to obtain a copy of the source code would just need to access the W3-mSQL enhanced web page directly rather than accessing it via the W3-mSQL CGI program. If users save the source of the page from their browser they would have a full copy of the source code on their machine. Naturally, this is a major problem for people who write proprietary applications.

To overcome this problem, W3-mSQL provides two features, private scripts and pre-compiled libraries[1]. The web server may also provide a feature that can overcome this problem.

A problem associated with embedding the source code in an HTML document is that by pure definition an HTML document is a public document (available to anyone via the web
server). The software written with W3-mSQL is safe as long as the user only accesses it via the W3-mSQL CGI program (because it will be processed and removed from the HTML source before it is sent to the browser). So the problem is not that the source code is in the HTML file, it is that a user may access the HTML file directly by specifying the URL and bypass the W3-mSQL CGI program.

The obvious solution to this problem would be if the HTML file was not available directly from web server. If that was the case the user couldn't specify the URL directly and as such could not download the source code. But, how is this possible if the W3-mSQL expects to find the enhanced HTML file in the web document space? The solution is to install the enhanced web pages as private scripts.

A private script is an HTML file that is installed outside the web document tree[1] (i.e. it is not directly available through the web server). When a page is requested via W3-mSQL, it looks for the file based on the URL specified. For example, if page /cgi-bin/w3-msql/test/myfile.html is requested, W3-mSQL would try to load and process WEB_ROOT/test/myfile.html, where WEB_ROOT is the directory in which web pages (such as /usr/local/etc/htdocs or similar) was installed. If the file is found at that location, W3-mSQL will load and process it. If it doesn't find the file at that location, W3-mSQL assumes it must be a private script.

When W3-mSQL determines that the request references a private script (i.e. it didn't find the page in the web tree) it looks in an external directory for the page. The default location for private scripts is /usr/local/Hughes/www. In the example above, W3-mSQL will try to load /usr/local/Hughes/www/test/myfile.html and process it. In short, it will use the private script directory as a second web document tree. The web server does not know that documents are stored in that directory so it is not able to send them without the help of W3-mSQL.

In the above example, if someone tried to load /test/myfile.html directly, the web server would report an error because the file does not exist in the web tree. If the user then requested it using the W3-mSQL CGI program, a check for the file in the web tree would fail so the file installed in the private script directory would be loaded, processed and sent back to the user. This eliminates the possibility of a user directly accessing the file and downloading the source code.

Lite libraries are pre-compiled version of Lite functions that are loaded into Lite scripts and W3-mSQL enhanced web pages at run-time[1]. From a security point of view, libraries can be used to hide the Lite source code from a user. A library is a binary version of the Lite code in the same way that an object file is a binary version of C code after it has been
compiled. If all of the "sensitive" functions are placed in a library then they are totally hidden from the remote users (the binary file will be of no use to anyone as it cannot reverse the process and turn the library back into source code).

Using libraries in this way also increases the performance of the W3-mSQL applications because the source code does not need to be compiled every time the page is requested (it was compiled once and the binary version is then loaded directly into the Lite Virtual Machine when needed)\(^1\).

4.3 Lite -- mSQL's own scripting language

The fact that mSQL can be accessed from virtually every popular scripting language used on UNIX systems has been one of the factors in its popularity. Quite often, however, it is also one of the greatest causes of frustration for users. Adding mSQL support to an existing language, such as Perl or Tcl, requires compilation of a modified version of the language including the mSQL specific code\(^1\). For an average user this can often be a time-consuming and error prone process.

To help overcome this, mSQL 2.0 includes its own scripting language, preconfigured with support for the mSQL API. It is also the same language used by W3-mSQL, the WWW to mSQL interface package. People wishing to access mSQL from scripts and via the web now have to learn only one simple yet powerful language.

Lite has been designed to mimic the syntax and semantics of the C language while reducing some of the complexities and error prone features of C. This is intentional as most programmers working on UNIX machines have a working knowledge of C but look for a more "easy to use" language for scripting\(^1\). The main changes from C are

- All memory management (i.e. allocation and deallocation of memory for variables) is taken care of by the Lite Virtual Machine. The script does not need to perform any memory management routines.

- A variable has no fixed type. It will contain whatever is stored in it (e.g. char value, numeric value). When an operation is performed on a variable, such as a mathematics operator, the contents of the variable are checked to ensure they are of the correct type.

- There is a dynamic array type. Each element of the array is a variable as described above. The elements are accessed as they are in C, i.e. variable[offset], but they need not be declared before use. The array element is created when a value is stored in it.
without an pre-definition of the array.

- Variables are not pre-declared. They are created when they are first used.

- Variable names must start with a $ character. This will be familiar to shell script programmers.

(1) Variables, Types and Expressions

Variables are constructed from a $ sign followed by alpha-numeric characters and the '_' character. The only restriction placed upon the name of a variable is that the first character of a user defined variable must not be an upper case character. There is no need to pre-declare variables as does in a language such as C. A variable is created the first time when a value is assigned to it. Similarly, the type of the variable is defined by the value that is assigned to it. There are four types of scalar variables, char, integer, unsigned integers and real number. At any point in time, the type of a value can be changed by using the type cast notation from the C language.

Array variables are supported by Lite but there is no fixed type for the array. Each element of the array can hold data from any of the available data types. An array is created by assigning a value to one of the array elements such as

```plaintext
$arrayval[3] = "Foo";
$arrayval[4] = 5;
$arrayval[6] = 1.23 + 5.38;
```

Lite expressions are formed from mathematical equations incorporating the values of variables and values returned from function calls. Lite is a little more flexible than other languages such as C. It will allow mathematical operations on all data types including the char type. Adding two char values together results in the concatenation of the two strings. It can also perform operations on values of different types by casting the value to the correct type within the expression. Example are given below:

```plaintext
$charval = "Hello" + " there!";
$intval = 8 + 1;
$charval = (char)$intval + " green bottles";
```

Math expressions of any complexity, including any number of sub expressions enclosed in ( ) characters, are supported. The available math operators, +, -, *, / may be applied to int or real data types, and operator + may also be applied to text type. A special operator
supported by Lite is the count operator written as the # sign. The count operator is used to determine the size of certain variables. If applying the count operator to a char value it will evaluate the number of characters in the string. Applying it to an array will evaluate the number of elements in that array.

(2) Conditions and Loops

Conditions are provided by Lite using the same syntax as C. That is, the conditional block is started by an 'if (condition)'. The blocks of code are defined using the { and } character. Unlike C, it must always wrap code blocks in { } characters (in C don't have to if the code block is only one line long). After the initial code block, an optional 'else' block may be defined.

Multiple parts of the conditional expression may be linked together using logical AND and OR. Like C, the syntax for an AND is && while the syntax for an OR is ||. However, Lite provides more flexibility than C in conditions containing text values. Two text values can be compared using the '==' equality test or the '!=' inequality test rather than having to use a function such as strcmp().

Lite supports only one form of looping -- a 'while' loop. The syntax and operation of the while loop is identical the while loop offered by the C language. This includes the use of 'continue' and 'break' clauses to control the flow of execution within the loop.

(3) User Defined Functions

As with most modern programming languages, Lite allows users to write functions. In a Lite script a function is defined as follows:

    funct functName ( type arg, type arg ...)
    {
        statements
    }

As the definition dictates, a function must be started with the funct label. The remainder looks like a C function declaration in that there is a function name followed by a list of typed arguments. Any type may be passed to a function and any type may be returned from a function. All values passed to a function are passed by value, not by reference.

It must be noted that function declarations can only be made before any of the actual script code of the file. That is, all functions must be defined before the main body of the script is reached.
(4) User Defined Libraries

To help provide an efficient programming environment, Lite (and W3-mSQL) allows to build a library of functions and load the library into the script at run-time. This allows for effective re-use of code in the same way the languages such as C allows to re-use code by linking against libraries. The main difference is that the library is not "linked" into the script, it is loaded on request at run-time (a little like a C shared library).\(^1\)

The power and convenience of Lite libraries is most obvious when writing large WWW based applications using W3-mSQL. Like any application, there will be actions that will need to be performed several times. Without the aid of libraries, the code to perform those actions would need to be re-coded into each W3-mSQL enhanced web page (because each HTML file is a stand-alone program). By placing all these commonly used functions into a library, each web page can simply load the library and have access to the functions. This also provides a single place at which modifications can be made that are reflected in all web pages that load the library.

Library files are not like normal Lite script files. A Lite script file is a plain ASCII text file that is parsed at run-time by Lite. A library file contains the pre-compiled version of the Lite functions. These will load faster as they do not need to be re-parsed every time they are used. A Lite library file is created by using the -l flag of the Lite interpreter. If a set of functions is placed in a file called mylib.lite, a compiled version of the library is created using the syntax shown below.

\[
\text{lite -lmylib.lib mylib.lite}
\]

The -l flag tells Lite to compile the functions and write the binary version of the functions to a file called mylib.lib. This is similar to the concept of using the C compiler to create an object file by using the -c flag of the compiler.

There are three points that should be noted about the use of Lite libraries. Firstly, it should be noted that a Lite library can only contain functions (i.e. it cannot contain any "main body" code that you would normally include in a script file). Secondly, like functions themselves, a library can only be loaded into a Lite script prior to the start of the main body code (using load "mylib.lib"). Finally, the path given to the load command within the script does not enforce a known location for the library file. If you specify the library file as "mylib.lib" the Lite will expect the library file to exist in the current directory. You can of course provide a complete pathname rather than just a filename to the load command.
4.4 Lite's standard module, mSQL module and custom module

Lite provides two kinds of modules. One is standard module which is UNIX system related module like standard C library. The other is mSQL related module which is used to access and manipulate mSQL database.

(1) Lite's standard module

The standard module is to Lite as the standard C library is to C. It is a library of functions that are available to all Lite programs. It provides basic functionality for string manipulation, file I/O and other normal expectations of a programming language. Outline below is a description of each of the functions available within the standard module.

(a) Input Output Routines

echo ( char *string)

echo() outputs the content of string to be included in the generated HTML. Any variables that are included in string are evaluated and expanded before the output is generated.

printf ( char *format [ , arg ... ])

printf() produces output that is included in the HTML sent to the browser. It works in the same way as printf() in C. It should be noted that unlike echo(), any variables included in the format string passed to printf() are not expanded before the output is generated. The only way to include variable values in the output is to use C styled format definitions (such as "%s" for a string value etc).

fprintf ( int fd , char *format [ , arg ... ])

Like printf(), fprintf() produces text output based on the content of the format string and the args passed to the function. Unlike printf(), fprintf() sends the output to a file rather than including it in the HTML sent to the browser. The first arg is a file descriptor as returned by the open() function. See the description of open() below for more information.

int open ( char *path , char *access )

open() opens the object (usually a file) pointed to by path for reading and/or writing as specified by the access arg, and returns a file descriptor for that newly opened file. The possible values for the access flags are:
Flag | Description
---|---
"<" | File is opened for reading
">" | File is opened for writing
"<>" | File is opened for reading and writing
"<P" | Create a named pipe in the file system and open it for reading
">P" | Create a named pipe in the file system and open it for writing
"<I" | The contents of path is a shell command. Execute the command and allow reading from the process.
">I" | The contents of path is a shell command. Execute the command and allow writing to the process.

An error is indicated by a returned value of -1. In such a case, the system variable ERRMSG will contain the error message. NOTE : both the pipe related modes, i.e. "<P" and ">P", create the pipe prior to accessing it. If the pipe exists in the file system prior to the call, open() will fail.

**close ( int fd )**
close() closes an open file descriptor. If the descriptor relates to a file or a pipe, the file or pipe is closed. If the descriptor is a process, the stdin of the process is closed (and the process should terminate when it reads an EOF from its input). NOTE : If not close all file descriptors that opened then it will eventually run out of file descriptors.

**read ( int fd , int numBytes )**
read() reads numBytes bytes of data from the specified file descriptor and returns the data. It returns the empty string "" when on end of file or error. $ERRMSG will be set if an error occurred.

**readln ( int fd )**
readln() reads a line of text from the nominated file descriptor and returns the data. The newline value is not removed from the data returned. Like read(), the return of an empty string indicates EOF or an error. $ERRMSG will be set to a non-empty string on error.

**readtok ( int fd , char *token )**
readtok() reads data from the file descriptor until it finds the character specified as the token in the input data. The data read prior to the token is returned, the token is not. NOTE : The token is a single value character. If more than one character is passed in the token arg, only the first character is used.
(b) String Manipulation Routines

**split (char *str, char *token)**

`split()` splits the contents of a variable into multiple substrings using the value of `token` as the separator character. The result of splitting the string is returned as an array. If more than one character is passed as the token, all but the first character is ignored.

**strseg (char *str, int start, int end)**

`strseg()` returns a segment of the string passed as the `str` arg. The segment starts at `start` characters from the start of the string and ends at `end` characters from the start of the string. In the example below, `$sub` will contain the string "is a".

**char * chop (char *str)**

`chop()` removes the last character from the text string `str` and returns the new value. The primary use of this function is for chopping end-of-line characters off strings read from files with `readln()`.

**char * tr (char *str, char *expr1, char *expr2)**

`tr()` performs text translations on the string arg `str` based on the contents of `expr1` and `expr2` and returns the modified string value. `expr1` and `expr2` are sets of characters. Any character that is found in `str` that matches a character in `expr1` is translated to the corresponding character from `expr2`. The character sets can be defined by listing individual characters or by providing character ranges (such as A-Z to indicate all characters between A and Z).

**char * sub (char *str, char *expr1, char *expr2)**

`sub()` performs string substitutions on the string arg `str` based on the contents of `expr1` and `expr2`. If the string value passed as `expr1` is found anywhere in `str` it is substituted for the value if `expr2`.

**char * substr (char *str, char *regexp, char *pattern)**

`substr()` extracts substrings from `str` based on the regular expression `regexp` and the extraction pattern pattern. Any parts of the string that are matched by parts of the regular expression enclosed in parenthesis are made available to the extraction pattern. The first such substring is available as `$1`, the second as `$2` and so on. The string value created by expanding any such variables in pattern is returned.
( c ) File Manipulation Routines

**test (char *test, char *filename )**
test() offers functionality similar to the test program provided by the shell. Given a filename and a test, it will determine if the file matches the test specification. If it matches, 1 is returned otherwise 0 is returned.

The available tests are

<table>
<thead>
<tr>
<th>Test</th>
<th>File Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;b&quot;</td>
<td>Block mode device</td>
</tr>
<tr>
<td>&quot;c&quot;</td>
<td>Character mode device</td>
</tr>
<tr>
<td>&quot;d&quot;</td>
<td>Directory</td>
</tr>
<tr>
<td>&quot;p&quot;</td>
<td>Named pipe</td>
</tr>
<tr>
<td>&quot;s&quot;</td>
<td>Non-empty regular file</td>
</tr>
<tr>
<td>&quot;f&quot;</td>
<td>Regular file</td>
</tr>
<tr>
<td>&quot;u&quot;</td>
<td>File is setuid</td>
</tr>
<tr>
<td>&quot;g&quot;</td>
<td>File is setgid</td>
</tr>
</tbody>
</table>

**unlink (char *path )**
unlink() removes the named file from the file system.

**umask (int mask )**
umask() sets the umask for the current process. As with any numeric value, the mask can be given in decimal, hex or octal.

**chmod (char *path , int mode)**
chmod() changes the mode of the specified file to the specified mode.

**mkdir (char *path )**
mkdir() creates the directory specified by path.

**chdir (char *path )**
chdir() changes directory to the specified path.

**rmdir (char *path )**
rmdir() removes the specified directory from the file system.

**rename (char *old , char *new )**
rename() renames the specified file from the old name to the new name.
**truncate (char *path, int length)**

truncate() will set the length of the file to the specified length.

**link (char *path, char *new)**

link() will create a new link named new to the file specified by path.

**symlink (char *path, char *new)**

symlink() will create a symbolic link called new to the file specified by path.

**stat (char *path)**

stat() provides an interface to the stat() system call. The information from stat() is returned as an array. The elements of the array are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Inode number</td>
</tr>
<tr>
<td>1</td>
<td>File mode</td>
</tr>
<tr>
<td>2</td>
<td>Number of links to file</td>
</tr>
<tr>
<td>3</td>
<td>UID</td>
</tr>
<tr>
<td>4</td>
<td>GID</td>
</tr>
<tr>
<td>5</td>
<td>Size of file</td>
</tr>
<tr>
<td>6</td>
<td>atime</td>
</tr>
<tr>
<td>7</td>
<td>mtime</td>
</tr>
<tr>
<td>8</td>
<td>ctime</td>
</tr>
<tr>
<td>9</td>
<td>Block size of file system</td>
</tr>
<tr>
<td>10</td>
<td>Number of file system blocks used</td>
</tr>
</tbody>
</table>

(d) Process Oriented Routines

System facilities such as fork and exec are not available in the standard module. As this module is shared by both Lite and W3-mSQL it is not appropriate for such calls to be included here (having web pages fork child processes is not a sound idea). A supplementary module called mod_proc will be made available to provide these facilities.

**sleep (int time)**

sleep() will suspend operation of the script for time seconds.
system (char *command )

system() will execute the command line specified by command in a subshell. Any output generated by the command is included in the HTML output. The exit status of the command is returned to the caller.

getpid ( )

getpid() returns the process ID of the process running Lite.

getppid ( )

getppid() returns the process ID of the process that is the parent of the process running Lite.

kill (int pid, int signal )

kill() sends the specified signal to the specified process.

(e) Date/Time Related Routines

time ( )

time() returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds as an integer value.

ctime (int time )

ctime() converts a value returned by time() into the standard UNIX text representation of the date and time.

time2unixtime (int sec, int min, int hour, int day, int month, int year )

time2unixtime() provides a facility by which you can create a standard UNIX time value (i.e. the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds) for any specified date/time.

unixtime2year ( )

unixtime2month ( )

unixtime2day ( )

unixtime2hour ( )

unixtime2min ( )

unixtime2sec ( )

unixtime2* (int time )

The above functions take a UNIX time value (i.e. seconds since Jan 1, 1970) and return an integer value representing part of the time information.
**strftime (char *fmt, int time )**

`strftime()` returns a text representation of the UNIX time value `time` based on the format string passed as `fmt`. The available formatting options are:

- `%a` day of week, using locale's abbreviated weekday names
- `%A` day of week, using locale's full weekday names
- `%b` month, using locale's abbreviated month names
- `%B` month, using locale's full month names
- `%d` day of month (01-31)
- `%D` date as `%m/%d/%y`
- `%e` Day of month (1-31 with single digits preceded by a space)
- `%H` hour (00-23)
- `%I` hour (00-12)
- `%j` Day of year (001-366)
- `%k` hour (0-23, blank padded)
- `%l` hour (1-12, blank padded)
- `%m` month number (01-12)
- `%M` minute (00-59)
- `%p` AM or PM
- `%s` seconds (00-59)
- `%T` time as `%H:%M:%S`
- `%u` week number in year (01-52)
- `%w` day of week (0-6, Sunday being 0)
- `%y` year within the century (00-99)
- `%Y` year including century (e.g. 1999)

(f) Passwd file Related Routines

**getpwnam (char *uname )**

Returns the passwd file entry for the user specified by `uname`. The result is returned as an array with the array elements defined as below.

```
[ 0 ] = username
[ 1 ] = password
[ 2 ] = UID
[ 3 ] = GID
[ 4 ] = GECOS
[ 5 ] = home directory
[ 6 ] = shell
```
getpwuid (int UID)
getpwuid() returns the same information as getpwnam() but uses a UID to identify the user rather than a username. See the definition of getpwnam() above for details of the return format and usage.

(g) Network Related Routines

gethostbyname (char *host)
gethostbyname() returns an array of information about the specified host. Element 0 of the array contains the hostname while element 1 contains the hosts IP address.

gethostbyaddr (char *addr)
gethostbyaddr() returns an array of information about the specified host. Element 0 of the array contains the hostname while element 1 contains the hosts IP address.

(h) Routines available only in W3-mSQL

urlEncode (char str)
urlEncode() returns a URL encoded version of the specified string. The returned data can then be used in GET method operations without any potential problems of not conforming to the data encoding standard.

setContentType (char *str)
setContentType() can be used to override the default content type sent to in the HTML header of the generated HTML output. If it is to be used, it must be the first line of the script. Note: not even a blank line may proceed a call to setContentType().

includeFile (char *filename)
includeFile() may be used to include the contents of the specified file in the HTML output sent to the browser. The contents of the file are not modified or parsed in any way. If the first character of the file name is a / then the filename is an absolute path name from the root directory of the machine. If it is note, the filename is a relative path from the location of the script file.

(2) Lite's mSQL related module

The Mini SQL module is a library of routines for communicating with a Mini SQL database. The functions provided by this module mimic the functions provided by the mSQL C API. Outlined below is a description of each of the functions available within the Mini SQL module.
int mysqlConnect (char *host)
mysqlConnect() connects to the mSQL server on the specified host. If no host is specified it connects to the local mSQL server.

mysqlClose (int sock)
mysqlClose() closes a connection made using mysqlConnect().

int mysqlSelectDB (int sock, char *db)
mysqlSelectDB() tells the mSQL server which database to use.

int mysqlQuery (int sock, char *query)
mysqlQuery() submits a query to the mSQL server connected to the specified socket.

mysqlStoreResult ()
mysqlStoreResult() stores any data that was a result of the previous query.

mysqlFreeResult (int res)
mysqlFreeResult() frees any memory allocated to the specified result.

mysqlFetchRow (int res)
mysqlFetchRow() returns a single row of the data stored in the specified result.

mysqlDataSeek (int res, int location)
mysqlDataSeek() allows to move the data pointer within the result table. Specifying a location of 0 will rewind the result. The next call to mysqlFetchRow() will return the first row of the result table again.

mysqlListDBs (int sock)
mysqlListDBs() returns an array of the names of the databases available on the specified server.

mysqlListTables (int sock, char *db)
mysqlListTables() returns an array of the names of all the tables available in the current database of the specified server.

mysqlInitFieldList (int sock, char *db, char *table)
mysqlInitFieldList() generates an internal result handle containing details of all the fields in the specified table of the specified database. The result handle is used in conjunction with the functions below to access the field structure information. Note that the result handle is
held as a static variable inside the mSQL module and further calls to msqlInitFieldList() will free the result.

**msqlListField()**
msqlListField() returns an array of information about a single field of the current field list result that was generated using msqlInitFieldList(). The elements of the array contain the following information.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Field Name</td>
</tr>
<tr>
<td>1</td>
<td>Table Type</td>
</tr>
<tr>
<td>2</td>
<td>Type</td>
</tr>
<tr>
<td>3</td>
<td>Length</td>
</tr>
<tr>
<td>4</td>
<td>Flags</td>
</tr>
</tbody>
</table>

**mSQLFieldSeek (int res, int location)**
mSQLFieldSeek() acts upon the result of a call to msqlInitFieldList() in the same way mSQLDataSeek() acts upon the result of a call to mSQLStoreResult(). It allows to move the internal result pointer to the specified location.

**int mSQLNumRows (int res)**
mSQLNumRows() returns the number of rows contained in the result handle res.

**mSQLEncode (char *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.

(3) Writing Lite/W3-mSQL custom module

The Lite scripting language (which incorporates W3-mSQL) provides a mechanism for accessing external C language functions from within Lite scripts. This functionality is used to provide the Standard Module (access to many standard UNIX functions) and also the mSQL Module (access to the mSQL API functions). If there is a C library that would like to access from Lite/W3-mSQL scripts then it can write a custom module to provide this access.

Inside the Lite Virtual Machine (VM), that is the software environment that actually manages the execution of Lite code, there is a table that provides a mapping between Lite function names and external C code functions from the modules. Each entry in the table
includes the Lite name for the function, a pointer to the C function, and details about the number and type of parameters. If details of the parameters are given, the VM will ensure that the function is called with the correct number and type of functions. For example, the lite function open() may map to a C function in the standard module called doOpen(). The VM uses this table whenever the Lite script calls a function. If the script calls the open() function the VM understands that it must actually call an external C function called doOpen().

The other aspect of calling external functions is the passing of data to the function and returning data from the function. A function will usually require some parameters (such as a file name in the case of open ( ) ) and will also usually return some information (such as a file descriptor). Data that is being passed to the script as a parameter must be taken from the internal data structures used by the VM and provided in a useable form to the C code function in the module. Similarly, the C function must be able to take a normal C variable value and give it back to the VM in a useable form so that it can then be manipulated within the Lite script (assigned to a variable for example).

Passing data into the module function is achieved via a parameter list. A parameter list is a linked list of symbol structures where each element of the list represents one of the parameters passed to the function from inside the Lite script. The C function can then traverse this list and access the data values contained within the symbol structures. The parameters are presented to the C function in the same order in which they were passed to the Lite function. So, if a call was made inside a lite script as follows

```c
open ("/tmp/test.txt", "<")
```

The C function doOpen ( ) would be called with a parameter list containing 2 symbols. The first symbol would contain "/tmp/test.txt" while the second would contain "<".

Returning data to the script from the C function is achieved in a similar manner. It is the responsibility of the C function to create a valid symbol structure containing the return data. A pointer to this symbol structure is then assigned to a global, external variable called externReturn so that the VM can access this newly created symbol when execution resumes.

The writing of custom modules is used to extend the functionality of the language beyond what can be achieved using the Lite Library facilities. The applications of this are endless and some extra modules for Lite will be provided in the future mSQL package release.
V. Application of mSQL/W3-mSQL -- Student Registration and Course Management Forms on the Web

A practical web-to-database processing application should provide fast access to the database via a network with different level access control for various users. The front-end users (e.g. Student and Management here) should have full access to those related database tables for browsing/reading but have some restricted access for writing back to table. This restriction is generally specified by developer or administrator of the application system. The back-end users (e.g. Developer or Administrator) are responsible for development and maintenance of the system, so they have full access to the system for reading and writing from or to database. To collect data from the user and send data back to the user through the web server, CGI (common gateway interface) script is used for this communication between the gateway and web server[3].

mSQL provides fast access to stored data and is especially good for these web purposes. The example application we have implemented constructs two kinds of database table using mSQL. One, at management-side (e.g. quarter_Course, quarter_Grade tables), is for storing courses and student grades information. The other, at student-side (e.g. quarter_Schedule), is for storing student course schedule information.

HTML and W3-mSQL/Lite (CGI script) can be used to develop sophisticated GUI based application that is platform independent and available as shared resources on the network. They are used in this application to provide a user interface and process a user’s request for both management and student sides. The system diagram is as Figure 1.

![Diagram](image-url)

**Figure 1. Web-Database Integration System**
5.1 Database construction and development

Two kinds of database tables, student-side and management-side, are built for development of this application. Management-side database tables, (quarter_Course, quarter_Grade), are used for storing course information and student grades each quarter. These tables have restricted access to the read/write operations for system management but a student user can access to read courses available. The user (Management) can store, manipulate and retrieve data in a table via the web interface or by using an mSQL script file via an msql interactive interface. Student-side database tables are used to keep a student's schedule information. The read/write operations to database table is only through the web interface. Both kinds of tables can only be created and dropped by mSQL server superuser (root) or by a specified mSQL user in a configuration file.

(1) Management-side database table

quarter_Course Table – which contains all courses available for this quarter (e.g. sp97_Course). The structure of Quarter_Course table is:

<table>
<thead>
<tr>
<th>code</th>
<th>title</th>
<th>description</th>
<th>hours</th>
<th>classtime</th>
<th>instructor</th>
<th>location</th>
<th>labtime</th>
<th>labroom</th>
</tr>
</thead>
</table>

which is created by following code: (e.g. sp97_Course)

```sql
create table sp97_Course ( 
    code     char(4)  not null,
    title    char(8)  not null,
    description  char(18) not null,
    hours    int,
    classtime  char(18),
    instructor char(15) not null,
    location  char(8),
    labtime   char(18),
    labroom   char(8) 
) 
```

The field code is created as a unique index to uniquely identify each course.

```sql
create unique index sp97cursidx on sp97_Course (code) 
```
quarter_Grade table – which contains all student course grades in this quarter (e.g. sp97_Grade). The table structure is:

<table>
<thead>
<tr>
<th>coursecode</th>
<th>username</th>
<th>grade</th>
</tr>
</thead>
</table>

which is created by:

```sql
create table sp97_Grade (  
coursecode       char(4) not null,  
username         char(8) not null,  
grade            char(3) not null  
)
```

Both fields coursecode and username are defined as a unique index:

```sql
create unique index sp97gradeidx on sp97_Grade (coursecode, username)
```

Here, username is used to uniquely identify each student, since it’s a unique login name on UNIX system.

(2) Student-side database table

quarter_Schedule Table – which keeps all student schedule for this quarter (e.g. sp97_schedule). The table structure is:

<table>
<thead>
<tr>
<th>username</th>
<th>coursecode</th>
</tr>
</thead>
</table>

which is defined by:

```sql
create table sp97_Schedule (  
username       char(8) not null,  
coursecode     char(4) not null  
)
```

Both fields username and coursecode are used as a unique index.

```sql
Create unique index sp97schidx on sp97_Schedule (username, coursecode)
```
5.2 The development of interactive Registration/Management forms for the web

Along with advanced features of HTML (e.g. table, form, etc.), CGI/W3-mSQL/Lite can provide a good mean to design GUI on the web with a security and authentication mechanism. The development of the application for the web consists of three major parts: how to restrict access to this system, a management form for managing the course database, a registration form for registering for a class and displaying schedule and grades.

(1) Restricted access

The basic security policy requires that only authorized users of the system have accounts. Right now, this system is developed on the engineering LAN and restricted to access by engineering users. The prompts to input username and password are provided for user to login the system (Figure 2).

Due to security issues associated with system encrypted passwords, a password can not be processed currently. The password can’t be decrypted without the knowledge of the original transformation keys. To use this application practically, a system administrator can add this feature. The W3-mSQL/Lite code to control access is:

```bash
<!
/* process user login request */
if ($user == "") /* prevent user access directly without username */
{
    printf("<center><h2>Username can't be null. Please login again!</h2><center><\n";
    exit(1);
}
$pzinfo = getpwnam($user); /* get user information from system passwd file entry */
if ($pzwinfo[0] == $user)
{
    printf("<center><h2>Welcome to Registration Form</h2><center><\n";
}
else
{
    printf("User %s does not exist! Please check your account and re-login! \n", $user);
    printf("<p>BACK <a href="/projects/w3-mSQL/test-mSQL/Login.html">Re-Login</a>\n";
    exit(1);
}
>
(2) Registration form

Once the username is validated, the Registration form page (Figure 3) is displayed and a user can choose to browse the course database, register for a class, display a personal schedule or grades for a particular quarter. First the user needs to select a quarter for the registration request.

A user can browse the course database to get course availability information. Figure 4 shows the result of browsing Fall 97 courses.

W3-mSQL/Lite provides an mSQL related module to connect to a database and process queries. The Figure 4 is created by following code:

```php
<!>
$host = "banana.eng.auburn.edu"; /* hostname of mSQL server */
$DBName = "Manage_Database";
$courseTable = $quarter + ".Course";

$sock = mysqlConnect($host); /* establish the connection b/w web server and mSQL server */
if($sock < 0)
{
    fatal("Error: $ERRMSG\n"); /* $ERRMSG is system environment variable */
}

if(mysqlSelectDB($sock, $DBName) < 0)
{
    fatal("Error: $ERRMSG\n");
}

$query = "select * from $courseTable order by code"; /* mSQL query */
if(mysqlQuery($sock, $query)<0)
{
    fatal("Error: $ERRMSG\n");
}

/* print browse course database result header */
echo("<table border=2><th colspan=9 align= center>Course Schedule for $quarter</p>");
echo("<tr align=center><td>Code<td>Title<td>Description<td>Hours<td>Instructor
<td>Class Time<td>Location<td>Lab Time<td>Lab Room</center><tr><p>");
```
$result = mysqliStoreResult();
$NumRows = mysqliNumRows($result);
$row = mysqliFetchRow($result);
$count = 0;
while($count < $NumRows) /**< search tuple one by one */
{
    $count = $count + 1;
    $row = mysqliFetchRow($result);
}
mysqliFreeResult($result);
>

If a selection is to register for a class, a Register Class Form (Figure 5) will be created with the help of CGI/W3-mSQL. The related user input data needed by the form is attached as hidden fields:

    echo("<form action="/cgi-bin/w3-msql/projects/w3-mSQL/test-mSQL/Change_Course.html">\n")
    echo("<input type = hidden name = host value = \"$host\">\n")
    echo("<input type = hidden name = sock value = \"$sock\">\n")
    echo("<input type = hidden name = user value = \"$user\">\n")
    echo("<input type = hidden name = DBName value = \"$DBName\">\n")
    echo("<input type = hidden name = scheduleTable value = \"$scheduleTable\">\n<br>")
    echo("<input type = hidden name = courseTable value = \"$courseTable\">\n<br>")

To register for a class, a user needs to input a course code. If the course code is available, this course is then added to the personal schedule immediately. To drop a class a user simply types in the course code. The personal schedule of a student is limited up to 18 total credit hours for a quarter.

    /* to add a class to schedule */
    $query = "insert into $scheduleTable(username, coursecode) values ('$user', '$coursecode')";
    mysqliQuery($sock, $query);
    /* to drop a class from schedule */
    $query = "delete from $scheduleTable where(username='$user') and (coursecode='$coursecode')";

A user can browse their personal schedule (Figure 6) or grade (Figure 7) for some quarter at any time. Since the Schedule table or Grade table only contains coursecode, username
or grade information, to display details about personal schedule and grades, two or more table joins are required. mSQL provides this feature and this can be embedded in W3-mSQL/Lite program:

```php
/* query for browsing personal schedule */
$query = "select $courseTable.code, $courseTable.title, $courseTable.description,
           $courseTable.hours, $courseTable.instructor, $courseTable.classtime,
           $courseTable.location, $courseTable.labtime, $courseTable.labroom
from $scheduleTable, $courseTable
where ($scheduleTable.username = '$user') and
      ($scheduleTable.coursecode = $courseTable.code)
order by $courseTable.code";
mysqliQuery($sock, $query);

/* query for browsing personal grade */
$query = "select $courseTable.code, $courseTable.title, $courseTable.description,
           $gradeTable.grade, $courseTable.instructor
from $gradeTable, $courseTable
where ($gradeTable.username = '$user') and ($gradeTable.coursecode = $courseTable.code)
order by $courseTable.code";
mysqliQuery($sock, $query);

(3) Management form

The management can only be accessed by an administrator of the application or specified users in an msqql.acl access control file. This form (Figure 8) gives a user two selections, input course or input grade.

From the input page, a user can insert a course or delete a course from/to the course database for a quarter. A user can also display all the courses. The interface to input a course is shown in Figure 9.

The W3-mSQL/Lite program for inserting, deleting or browsing courses is as follows:

```
/* to delete a course from database */
$query = "delete from $courseTable where (code = '$code') or (title = '$title')";
mysqliQuery($sock, $query);

/* to browse courses available in database */
$query = mysqliQuery($sock, "select * from $courseTable order by code");

The database table or index etc. can also be created via a W3-mSQL/Lite program when the user sends a request. If the table does not exist, it is created. The following code shows this:

/* check how many Course Tables exist in database */
$stableList = mysqliListTables($sock, $DBName);
$num=0;
while($num< #stableList)
{
    if($courseTable == $stableList[$num])
    {
        break;
    }
    $num = $num + 1;
}

if($num == #stableList)    /* table not exist */
{
    /* create course table */
    $query = "create table $courseTable (code char(4) not null,
         title char(8) not null,
         description char(18) not null,
         hours int,
         instructor char(15) not null,
         classtime char(18),
         location char(8),
         labtime char(18),
         labroom char(8))";
    if(mysqliQuery($sock, $query) < 0)
    {
        fatal("Error: $ERRMSG\n");
    }
}
/* create index for the table */
$indexname = $quarter+"cursidx";
$query = "create unique index $indexname on $courseTable (code)";
if(mysqlQuery($sock, $query) < 0)
{
    fatal("Error: $ERRMSG\n");
}
echo("<h2>Table $courseTable created. \n</h2>"};

Inputting student grades (Figure 10) has same procedure as above. Figure 11 shows the result of browsing all student grades for the quarter Fall 97.
Login to Registration Form Page

This test Registration Form is developed with mSQL20 and w3-mSQL/Lite script. You can browse whole available course database, register classes for a particular quarter, check your schedule or grades. To access this test form, please type in your username and password below or email guiromg@eng.auburn.edu to get a temporary username and password.

Would you like to try it?

Your Username: [Blank]
Your Password: [Blank]

[Login] [Cancel]

---

Figure 2. Login Form for Registration

Registration Form Page

Welcome to Registration Form. First, please check a particular quarter. If you want to browse the course database, please click "Browse Courses". To display personal schedule for a particular quarter, click "Personal Schedule". To register a class, click "Register Class". Click "Personal Grades" to display your Grades for a particular quarter.

Check Quarter: Spring 97

Browse Courses  Register Class

Personal Schedule  Personal Grade

---

Figure 3. Student Registration Form
Register Result Form Page

Welcome to TEST ONLY Registration Form

(host at banana.eng.auburn.edu)

<table>
<thead>
<tr>
<th>Code</th>
<th>Title</th>
<th>Description</th>
<th>Hours</th>
<th>Instructor</th>
<th>Class Time</th>
<th>Location</th>
<th>Lab Time</th>
<th>Lab Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>2213</td>
<td>CSE100A</td>
<td>PERSONAL COMP APPL</td>
<td>3</td>
<td>STAFF</td>
<td>0800-0930</td>
<td>MW</td>
<td></td>
<td>SB101</td>
</tr>
<tr>
<td>2214</td>
<td>CSE100B</td>
<td>PERSONAL COMP APPL</td>
<td>3</td>
<td>STAFF</td>
<td>0800-0930</td>
<td>TH</td>
<td></td>
<td>SB101</td>
</tr>
<tr>
<td>2215</td>
<td>CSE100C</td>
<td>PERSONAL COMP APPL</td>
<td>3</td>
<td>STAFF</td>
<td>0930-1100</td>
<td>MW</td>
<td></td>
<td>SB101</td>
</tr>
<tr>
<td>2216</td>
<td>CSE100D</td>
<td>PERSONAL COMP APPL</td>
<td>3</td>
<td>STAFF</td>
<td>0830-1100</td>
<td>TH</td>
<td></td>
<td>SB101</td>
</tr>
<tr>
<td>2217</td>
<td>CSE100E</td>
<td>PERSONAL COMP APPL</td>
<td>3</td>
<td>STAFF</td>
<td>1100-1230</td>
<td>MW</td>
<td></td>
<td>SB101</td>
</tr>
<tr>
<td>2218</td>
<td>CSE100F</td>
<td>PERSONAL COMP APPL</td>
<td>3</td>
<td>STAFF</td>
<td>1100-1230</td>
<td>TH</td>
<td></td>
<td>SB101</td>
</tr>
</tbody>
</table>

Figure 4. Course Database Information Form
Register Result Form Page

Welcome to TEST ONLY Registration Form

(host at banana.eng.auburn.edu)

Please fill in Course Code (e.g. 1234):

Add Course    Drop Course    Clear

Figure 5. Register for Class Form

Register Result Form Page

Welcome to TEST ONLY Registration Form

(host at banana.eng.auburn.edu)

guirong's Schedule for fa97

<table>
<thead>
<tr>
<th>Code</th>
<th>Title</th>
<th>Hours</th>
<th>Instructor</th>
<th>Class Time</th>
<th>Location</th>
<th>Lab Time</th>
<th>Lab Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>2215</td>
<td>CSE1000</td>
<td>3</td>
<td>STAFF</td>
<td>MW 0900-1100</td>
<td>SB101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2220</td>
<td>CSE1010</td>
<td>3</td>
<td>STAFF</td>
<td>TH 1200-1400</td>
<td>SB101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. Personal Schedule Form
Register Result Form Page

Welcome to TEST ONLY Registration Form

(host at banana.eng.auburn.edu)

guirong's Course Grades for fa97

<table>
<thead>
<tr>
<th>Code</th>
<th>Title</th>
<th>Description</th>
<th>Grade</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>2215</td>
<td>CSE100C</td>
<td>PERSONAL COMPAPPL</td>
<td>A</td>
<td>STAFF</td>
</tr>
<tr>
<td>2220</td>
<td>CSE100H</td>
<td>PERSONAL COMPAPPL</td>
<td>A</td>
<td>STAFF</td>
</tr>
</tbody>
</table>

Figure 7. Personal Grade Form

Course Database Management Form

Welcome to Course Database Management Page. If you want to input the course table, please click "Input Course". To input course grades, click "Input Grade".

Input Course Table

Input Grade Table

Figure 8. Course Management Form
Figure 9. Input Course Database Form
Figure 10. Input Student Grade Form

Figure 11. Display Student Grade Form
VI. Limitations of mSQL and its Future Development

During mSQL's life, people have used it for applications far beyond the scope of the original design. These high-end applications, containing up to 1 million rows of data, showed a need for better handling of complex queries and large data sets if the package was to be used in this way. The second generation of the mSQL server has been designed to suit these high-end applications while maintaining the original design goals of mSQL 1. It has been designed to meet three main criteria\(^1\):

- Provide comparable performance for simple operations as mSQL 1.x.
- Provide rapid access to large databases and complex operations.
- Provide more of the functionality outlined in the ANSI SQL specification.

However, mSQL/W3-mSQL still has some limitations:

1. Like other Freeware or Shareware, it does not provide any technical support. There is only one mSQL mailing list\(^2\) and unofficial FAQ\(^3\) for discussing or exchange idea using mSQL.

2. The official released version is mSQL 1.0.16 now, which only supports a limited data types with low performance. Though mSQL 2.0B6 has been posted for trial, it is a beta version and has some obvious or potential bugs.

3. Even mSQL 2.0 does not support the whole set of Standard SQL. It does not support views or virtual tables. No nested selects, no aggregate functions (e.g. count, avg, min, max, etc.) are supported. mSQL does not support transaction commands, begin, commit, rollback, etc.

4. Although W3-mSQL provides an authentication system – w3-auth, the current beta version still has bugs. The practical application of this utility will be available at the official version mSQL 2.0.

5. Lite is a simple script language. It only provides some limited modules/functions so far.

mSQL/W3-mSQL is still under developed and the official new version mSQL 2.0 will be released soon. It should be expected that further functionality will be made available as the development of mSQL 2 continues.
References

1. mSQL-2.0B6 package documentation – included in the mSQL package

2. mSQL mailing-list (msql-list-request@bunyip.com)

3. mSQL FAQ (ftp: //bond.edu.au/pub/Minerva/msql/)

