CET W/32
MySQL support module

REVISIONS

Revision	Date	Changes	Changed by
0.1	02/17/03	First Draft	GK
0.2	02/18/03	Proofread and made minor syntax editing	RG
0.3	02/21/03	Extended list of supported BASIC statements	GK
0.4	02/24/03	Added Native API	GK
0.5	02/27/03	Added Unsupported Features List and References	GK
0.6	02/27/03	Added Example configuration script	GK

Table of Contents

1 Introduction

This document describes the optional MySQL extension module for the W/32 BASIC runtime.

1.1 New features

The optional MySQL extension module allows the BASIC programmer to access MySQL tables as if they were regular isam files.
The module exports a set of MySQL API functions similar to MySQL C API. These functions allow the BASIC programmer to use the full power of the MySQL database in BASIC programs.

1.2 Unsupported features

The following components of the W/32 compiler do not yet support MySQL:

- The cAccess function (FREEDOM Report Writer).
- The blist, bcreate, bcopy and other standalone utilities.

1.3 References

- Paul DuBois. MySQL, Second Edition. (Pearson Education, January 2003. ISBN 0735712123)
- Michael Widenius, David Axmark. MySQL Reference manual (O'Reilly & Associates, June 2002. ISBN 0596002653)
- Downloadable docs on the MySQL web site http://www.mysql.com/documentation/index.html

2 Working with MySQL database.

The MySQL module offers BASIC programmers two ways for working with MySQL databases:
- Indexed files emulation
- Native API

2.1 Indexed files emulation

The extension module supports access to MySQL database tables using regular BASIC file input/output statements:
- OPEN
- CLOSE
- READ
- READNEXT
- READPREV
- WRITE
- DELETE
- UNLOCK
- MAT READ
- MAT READNEXT
- MAT READPREV
- MAT WRITE

and the function:
- EOF

2.1.1 Configuration

The configuration information for the emulator is set through w32appw32 variable B_MYSQL:

B_MYSQL = <server>:<user>:<password>:<database>

Where:

<server>	MySql server name or IP address
<user>	user name to log on the MySQL server
<password>	password for the above user name
<database>	database used by the emulation module

The emulation module works with a single database only. This database must contain a configuration table named _cet_files_. This table contains information about BASIC indexed files converted to MySQL tables, such as the original file name, MySQL table name, key and record lengths, etc. The table format is described below in the “Installing and Configuring the MySQL Server” chapter.

2.1.2 File access methods and formats

The module supports indexed binary formatted files only. Valid options for OPEN statement are:
- INPUT INDEXED
- OUTPUT INDEXED
- UPDATE INDEXED
- INPUT INDEXED LOCK
- OUTPUT INDEXED LOCK
- UPDATE INDEXED LOCK

2.1.3 File names

The configuration table _cet_files_ contains information about mapping indexed file names to table names. The emulator uses the file name string passed to the OPEN statement as a search key to the _cet_files_ table to locate table name and other information required for i/o emulation.
The emulator does not support file system tree and does not parse the file name string. Instead, it uses the file name as is. This means that if you had two or more indexed files of the same name but located in different directories on your disk then when you move these files to the MySQL database you must select different names for the tables representing these files. The name mapping will look like:
Directory1\Data -> Data1
Directory1\Data -> Data2
That is, the statement OPEN #1:”Directory1\Data” will open the table Data1 and
the statement OPEN #1:”Directory2\Data” will open the Data2 table.

The emulator also uses the configuration table to determine whether some particular OPEN statement opens a table in the database or a native ISAM file. This allows the BASIC programmer to move his isam data to the MySQL database file by file.

2.1.4 File and record locking

The emulation module supports file locking (LOCK option of the OPEN statement) and record locking when the file is open for UPDATE.

2.2 Native API

The emulation module exports a set of BASIC-callable functions with functionality similar to the C API of the MySQL client library. This API can be used to access the MySQL database in the most efficient way using the native language of the MySQL server – SQL.

2.2.1 Native API function reference

The following table lists MySQL C API functions and their BASIC equivalents. Refer to MySQL API documentation for details.

MySQL C Function	`cetmysql Function`
mysql_affected_rows	`call cetmysql.affected_rows ( mysql%, addrof( rows% ) )`
mysql_close	`call cetmysql.close ( mysql% )`
mysql_connect	`Not implemented, use real_connect`
mysql_change_user	`call cetmysql.change_user ( mysql%, user$, passwd$, db$ | db%, addrof( errno% ) )`
mysql_character_set_name	`call cetmysql.character_set_name ( mysql% addrof( name$ ) )`

mysql_create_db	`Not implemented, use CREATE DATABASE`

mysql_data_seek	`call cetmysql.data_seek`
	`(`
	`     mysqlres%,`
	`     offset%`
	`)`

mysql_debug	`Not implemented`
mysql_drop_db	`Not implemented, use DROP DATABASE`
mysql_dump_debug_info	`Not implemented`
mysql_eof	`Not implemented, fetch_row_and_lengths returns 0 in mysqlrow% when eof condition detected`
mysql_errno	`Called implicitly by all functions which may set errno; result is returned in errno% function argument.`

mysql_error	`call cetmysql.error`
	`(`
	`     mysql%`
	`     addrof( error$ )`
	`)`

mysql_escape_string	`Not implemented`
mysql_fetch_field	`Not implemented`

mysql_fetch_field_direct	`call cetmysql.fetch_field_direct`
	`(`
	`     mysqlres%,`
	`     fieldnum%,`
	`     addrof( name$ ),`
	`     addrof( table$ ),`
	`     addrof( def$ ),`
	`     addrof( type% ),`
	`     addrof( length% ),`
	`     addrof( max_length% ),`
	`     addrof( flags% ),`
	`     addrof( decimals% )`
	`)`

mysql_fetch_fields	`Not implemented`
mysql_fetch_lengths	`Called implicitly by fetch_row_and_lengths function.`

mysql_fetch_row	`call cetmysql.fetch_row_and_lengths`
	`(`
	`     mysqlres%,`
	`     addrof( mysqlrow% ),`
	`     addrof( lengths% ),`
	`     addrof( errno% )`
	`)`
	
	`Use fetch_row_and_lengths function to fetch next row from the result set.`
	
	`call cetmysql.fetch_field_value`
	`(`
	`     mysqlrow%,`
	`     lengths%,`
	`     field%,`
	`     addrof( value$ )`
	`)`

	Use fetch_field_value function to fetch value of the field of current row.
mysql_field_seek	Not implemented
mysql_field_count	Not implemented
mysql_field_tell	Not implemented
mysql_free_result	`call cetmysql.free_result ( mysqlres% )`
mysql_get_client_info	`call cetmysql.get_client_info ( addrof( info$ ) )`
mysql_get_host_info	`call cetmysql.get_host_info ( mysql% addrof( info$ ) )`
mysql_get_proto_info	`call cetmysql.get_proto_info ( mysql% addrof( info% ) )`
mysql_get_server_info	`call cetmysql.get_server_info ( mysql% addrof( info$ ) )`
mysql_info	`call cetmysql.info ( mysql% addrof( info$ ) )`
mysql_init	`call cetmysql.init ( addrof( mysql% ) )`
mysql_insert_id	Not implemented
mysql_kill	Not implemented
mysql_list_dbs	Not implemented, use SHOW DATABASES
mysql_list_fields	Not implemented, use SHOW COLUMNS
mysql_list_processes	Not implemented
mysql_list_tables	Not implemented, use SHOW TABLES
mysql_num_fields	`call cetmysql.num_fields ( mysqlres%, addrof( fields% ) )`
mysql_num_rows	`call cetmysql.num_rows ( mysqlres%, addrof( rows% ) )`
mysql_options	Each option setting is implemented as a separate

	BASIC function. call cetmysql.opt_connect_timeout (mysql%, timeout%, addrof(errno%)) call cetmysql.opt_compress (mysql%, addrof(errno%)) call cetmysql.opt_named_pipe (mysql%, addrof(errno%)) call cetmysql.init_command (mysql%, command$, addrof(errno%)) call cetmysql.read_default_file (mysql%, file$, addrof(errno%)) call cetmysql.read_default_group (mysql%, group$, addrof(errno%))
mysql_ping	call cetmysql.ping (mysql%, addrof(errno%))
mysql_query	Not implemented, use real_query

mysql_real_connect	call cetmysql.real_connect
	(
	     mysql%,
	     host$ | host%,
	     user$ | user%,
	     passwd$ | passwd%,
	     db$ | db%,
	     port%.
	     unix_socket$ | unix_socket%,

	`flags%`
	`addrof( errno% )`
	`)`

mysql_real_query	`call cetmysql.real_query`
	`(`
	`mysql%,`
	`query$,`
	`addrof( errno% )`
	`)`

mysql_reload	`Not implemented`
mysql_row_seek	`Not implemented`
mysql_row_tell	`Not implemented`

mysql_select_db	`call cetmysql.select_db`
	`(`
	`mysql%,`
	`db$,`
	`addrof( errno% )`
	`)`

mysql_shutdown	`Not implemented`

mysql_stat	`call cetmysql.stat`
	`(`
	`mysql%,`
	`addrof( stat$ ),`
	`addrof( errno% )`
	`)`

mysql_store_result	`call cetmysql.store_result`
	`(`
	`mysql%,`
	`addrof( mysqlres% ),`
	`addrof( errno% )`
	`)`

mysql_thread_id	`Not implemented`
mysql_thread_safe	`Not implemented`

Mysql_use_result	`call cetmysql.use_result`
	`(`
	`mysql%,`
	`addrof( mysqlres% ),`
	`addrof( errno% )`
	`)`

2.2.2 Example of using native API in BASIC programs.

The example program mysqlapi.b is included in the distribution.
Compile the program:

```
bl mysqlapi.b
```

and run it:

```
mysqlapi _cet_files_
```

If the B_MYSQL variable is set correctly, the program connects to the MySQL server and prints the content of the _cet_files_ table. You can pass any table name to the program. If the table name does not exist or if the program encounters any error, it prints a message and exits.

The program uses settings from the B_MYSQL variable for MySQL host name, user name, user password and the database. The program expects the name of the table to dump to be passed through the command line.

3 Installing and configuring the MySQL server

3.1 Installing the MySQL server.

The MySql server binaries for different operating systems are available from the MySQL web site www.mysql.com. Download version 4.0 which is recommended by MySQL to use in new development. During this module development version 4.0.10-gamma was used. The InnoDB support is required for the isam file emulation so select server configuration with InnoDB support included.
Please follow instructions on the web site to download and install the server. The server can be installed into any directory. The next “Configuring the MySQL server” chapter assumes that it is installed on a Windows machine into C:\MYSQL directory
You will probably want to add C:\mysql\bin to your PATH so you could easily run MySQL command line tools.
It is also recommended to download and install the MySQL Control Center.

3.2 Configuring the MySQL server

3.2.1 Server-specific configuration

The MySQL server uses the configuration file my.cnf. On Windows machines this file is normally located at C:\my.cnf.

- The MySQL installer puts four example configuration files into the root installation directory, C:\MySQL in our case. The files are my-small.cnf, my-medium.cnf, my-large.cnf and my-huge cnf. Select the appropriate template for your machine configuration and database size. Refer to MySQL documentation for additional details.
- Copy the selected configuration file to C:\my.cnf
- Open the c:\my.cnf file with any text editor and uncomment all settings following the line saying “Uncomment the following if you are using Innobase tables”.
- Note the directories set to innodb_data_home_dir, innodb_log_group_home_dir, and innodb_log_arch_dir variables. These variables set the location for InnoDB data files. You can replace default directories with anything you want or you can just leave it as is.
- Create the above directories. The server does not create these directories, you must create them yourself.
- Open the command prompt window and start the mysql server for the very first time:
  ```
  mysqld-max –console
  ```
  The server will create the InnoDB dataspace and print some messages while doing that:
  ```
  InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
  InnoDB: a new database to be created!
  ……………………………………….
  011024 10:58:25 InnoDB: Started
  ```
- Open the second command prompt window and use the mysqladmin utility to shut down the server:
  ```
  mysqladmin –user=root shutdown
  ```
- Install the server as a Windows Service and start it:
  ```
  mysqld-max –install
  net start mysql
  ```

3.2.2 File I/O emulator configuration

Next you should configure users and databases. While everything can be done with the mysqladmin utility, it’s easier to use the Control Center for this. Please refer to MySQL and MyCC documentation for details.
You should create:

- New database to use by the emulator. Call it as you wish. This name then must be set in the B_MYSQL variable.

- New user with granted access to the emulator database. The user name and password then must be set into the B_MYSQL variable.
- New table _cet_files_ in the newly created database. The table can be of any type. The MyISAM type is recommended. This table must contain the following columns:
    - Name char(255)
    - Type char(1)
    - Reclen int unsigned
    - Keylen int unsigned
    - Table char(100)
    - Key char(100)
    - Record char(255)
- Set both Name and Type fields as a primary index

3.2.3 Example configuration script.

The example MySQL configuration script w32init.sql is included for your convenience. The script creates a database named w32 and creates an empty _cet_files_ table in this database. The script does not create a new user as it is not necessary for the example configuration. You can use user root while testing the file i/o emulation. However, for your production environment it is strongly recommended to create a user name specific for your BASIC application and grant this user only access to the emulator database. The main reason for that is that the user name and the password appear as plain text in your w32app.w32 file. The easiest way to create a user is using the MyCC utility.

To run the script, start the mysql utility as root:

```
mysql ––host=<host> --user=root ––password=<password>
```

and run the w32init.sql script

```
mysql> source w32init.sql;
Query OK, 0 rows affected (0.01 sec)

Query OK, 1 row affected (0.00 sec)

Database changed
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

Note that the script destroys the ‘w32’ database if it existed. Please be careful when running the script if you already have some files imported into the database.

4 Moving data from ISAM files to MySQL tables

The procedure of moving data from an isam file to a MySQL table can include the following steps:

- Analyzing the isam file with bsqlconv utility
- Splitting the isam file to two or more files, if required
- Creating tables in the emulator database and records in the _cet_files_ table
- Exporting data from the isam file to a text file using enhanced blist utility
- Importing data from a text file to MySQL table

4.1 Tools

4.1.1 The bsqlconv utility

The bsqlconv utility has been specially written for helping in isam->mysql data moving.
The command line is:

```
bsqlconv [-f fromkey] [-t tokey] [-c counter] <filename>

filename    Specifies the name of the file in a DOS or THEOS format.
            For example, either of the following names may be used to
            list cp\data\custname:
               .\cp\data\custname (or cp\data\custname)
               cp.data.custname
-f fromkey  Analyzes starting from the record with the tokey, a key in an
            indexed file or a record number in a direct file.  The default
            is to start at the beginning.
-t tokey    Analyzes the file up to (but not including) the record specified
            by the tokey. The default is to stop at the end of the file.
-c counter  Specifies the number of records to analyze.
-v          Displays the program version
```

The utility reads records from an indexed file in the given key range and analyzes the structure of each record. The main goal of the utility is to detect inconsistencies in records formatting. Only regularly-formatted files can be transparently converted to MySQL tables. If the file contains records formatted differently, such file must be split to two or more files of regular format. Such file can only be moved into two or more MySQL tables and the BASIC program must be modified accordingly.
If the utility found no inconsistencies then it creates and prints a sequence of SQL statements which then can be used to create the MySQL table, add a record to the _cet_files_ table and import data to the table.
If the utility found any inconsistency, it prints a warning message. The developer then must use –f, -t and –c command line parameters to select only a part of the original file for analyzing.

4.1.2 Enhanced blist utility

The blist utility is enhanced with the –e command line option. When used, this option forces the blist to generate output suitable for import into a MySQL database. It omits all page headers and pagination, encloses field values into double quotes and separates them with commas.
The –f, -t and –c options can be used to list only part of the file. It can be required if it was detected that file splitting is necessary. In this case these options in general should be set to the same values which were used for the bsqlconv.

4.2 Example

Following is the example conversion procedure for non-regular isam files. The procedure assumes that files will be moved into tables of database named ‘w32’ which must be already created. The _cet_files_ table must also be already created as documented above.

4.2.1 Creating the isam file for conversion

In the example we use an isam file with the first two records formatted differently than the rest of the file. First, create the file with bcreate:

bcreate example indexed reclen 100 keylen 20

Then write some data to the file using the following BASIC program:

```
open #1:"example", update indexed
rem write initial records, each record contains numeric and integer
     fields
write #1,"0001" : 1234.5678, 1234
write #1,"0002" : 8765.4321, 8765
rem write rest of the file with records containing string, integer and
numeric fields
for i%=1 to 9
    write #1,"Record" & STR$(i%) : "Record number " & STR$(i%), i% *
          100 + i%, i% * 1111.1111
next i%
close #1
```

Print the file content using blist:

```
blist example
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
example (Indexed)
Key    Length = 20
Record Length = 200
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
0001                :1234.5678,1234
0002                :8765.4321,8765
Record1             :Record number 1,101,1111.1111
Record2             :Record number 2,202,2222.2222
Record3             :Record number 3,303,3333.3333
Record4             :Record number 4,404,4444.4444
Record5             :Record number 5,505,5555.5555
Record6             :Record number 6,606,6666.6666
Record7             :Record number 7,707,7777.7777
Record8             :Record number 8,808,8888.8888
Record9             :Record number 9,909,9999.9999
```

4.2.2 Analyzing the isam file

Run the bsqlconv utility to discover structure of the file:

```
bsqlconv example
```

It should print the following:

```
Input file: example
# Indexed file 'example' Key Length 20 Record Length 200
Record 'Record1' :Inconsistent type 4 of field 1. Expected 2
```

The message printed by the utility means that the file record with key ‘Record1’ is formatted differently than all previous records of the file.
If the utility outputs such a message then the file cannot be converted to a single MySQL table. Two or more tables must be used. The first table will receive the records from the beginning of the file to the record preceding the record with ‘Record1’ key. Let's analyze only this part of the input file:

```
bsqlconv example -t Record1
```

The utility will print:

```
Input file: example
  To: Record1
# Indexed file 'example' Key Length 20 Record Length 200
No inconsistencies found
drop table if exists example;
create table example
(
        Key CHAR(20) BINARY NOT NULL PRIMARY KEY,
        Fld1 FLOAT,
        Fld2 SMALLINT
) TYPE=InnoDB;
replace `_cet_files_`
(`Name`,`Type`,`Reclen`,`Keylen`,`Table`,`Key`,`Record`)
  values ('example','I','200','20','example','`Key`','`Fld1`,`Fld2`');
load data infile 'c:/tmp/example.txt' into table example
  fields terminated by ',' enclosed by '"' lines terminated by '\r\n';
```

As you can see, the utility says that there are no inconsistencies in this part of the input file and creates sql statements for create SQL table and import data. Run the bsqlconv again and redirect its output into a file:

```
bsqlconv example -t Record1 >example1.sql
```

This will create an example1.sql file in a format suitable for the mysql utility. You will need this file later.

Now we must analyze the rest of the input file. Run the bsqlconv again using ‘Record1’ key as a start point for analyzing:

```
bsqlconv example -f Record1

Input file: example
From: Record1
# Indexed file 'example' Key Length 20 Record Length 200
No inconsistencies found
drop table if exists example;
create table example
(
        Key CHAR(20) BINARY NOT NULL PRIMARY KEY,
        Fld1 CHAR(15) BINARY,
        Fld2 SMALLINT,
        Fld3 FLOAT
) TYPE=InnoDB;
replace `_cet_files_`
(`Name`,`Type`,`Reclen`,`Keylen`,`Table`,`Key`,`Record`)
```

```
  values
('example','I','200','20','example','`Key`','`Fld1`,`Fld2`,`Fld3`');
load data infile 'c:/tmp/example.txt' into table example
  fields terminated by ',' enclosed by '"' lines terminated by '\r\n';
```

As there were no inconsistencies found, we can save the sql statements in another text file:

```
bsqlconv example -f Record1 >example2.sql
```

Now we have finished analyzing the file and found that it contains records of two formats. So two MySQL tables are required to move data from this file to the MySQL database. We created two SQL scripts to create these tables and to import data. However, both scripts use same file and table name – ‘example’. Open the script files with any text editor and replace all occurrences of ‘example’ with, say, ‘example1’ in the example1.sql file and with ‘example2’ in the example2.sql file. You can also replace field names with something more meaningful reflecting what data are kept in this field. For example, if the first field of the record is used to keep a customer name, replace ‘Fld1’ with ‘CustomerName’ etc. Be careful while editing the files and keep all single, double and back quotes as they are.

4.2.3 Exporting isam data to text files

The enhanced blist utility is used to export data from isam files to text files. The new –e command line option tells the blist to create a file in format suitable for import into a MySQL database with LOAD DATA statement.
Remember that you must create two files, using ‘Record1’ key as a split point:

```
blist example –e –t Record1 >example1.txt
blist example –e –f Record1 >example2.txt
```

4.2.4 Creating MySQL tables and importing data

First you must move text files with exported data to the computer where you run the MySQL server. You must put the files into the location used in the LOAD data statement in the .sql files. By default the bsqlconv utility uses the C:/tmp directory but you can change it to whatever better fits your server and network configuration.

Start mysql utility as user root and set w32 as the default database. If all parameters are correct and MySQL server is running you should see the following:

```
mysql --host=<host> --user=root --password=<pw> w32

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 13 to server version: 4.0.10-gamma-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Now run the example1.sql script:

```
mysql> source example1.sql
Query OK, 0 rows affected (0.03 sec)
Query OK, 0 rows affected (0.06 sec)
Query OK, 1 rows affected (0.00 sec)
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

The mysql utility executes the script and prints results. If you see four OK messages then the table was created and the data were imported into the database. Now you can run the example2.sql script and import the rest of the file data:

```
mysql> source example2.sql
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.05 sec)
Query OK, 1 row affected (0.00 sec)
Query OK, 9 rows affected (0.00 sec)
Records: 9  Deleted: 0  Skipped: 0  Warnings: 0
```

Now you have all data from the original isam file moved to the MySQL database and it is time to modify your BASIC program which uses the file.

4.2.5 Modifying BASIC programs

Note that BASIC program modification is only required if the isam file was split to two or more MySQL tables during data moving. If the isam file formatting is regular then it can be moved to a single MySQL table and no modifications to the BASIC program are required.