

MySQL University

Building MySQL Client Applications

Topic and Scope

- Building MySQL Client Applications
 - Using the MySQL C API
(no other connectors/languages covered)
 - Single and multi threaded
 - Covers regular clients and embedded server lib
 - Some parts cover UNIXoid systems only

Outline

- Prerequisites
- Build infrastructure
- Life cycle of a client application
- Life cycle of a regular query
- Life cycle of a prepared statement query
- Life cycle of stored procedure calls

Prerequisites

To build mysql client applications you need:

- MySQL public header files
- The MySQL client library(s)
- The `mysql_config` tool

If you installed from source or a tarball binary package these are always there, when installing from other package formats you may need to install extra “-dev” or “-devel” packages

Build Infrastructure

There are several ways to build client apps:

- Do it all manually
- Get a little help from **mysql_config**
- Automate with GNU autotools
- ... or with Cmake?

Build manually

- Compile:

```
$CC -I$prefix/include/mysql -c myapp.c
```

- Link:

```
$LD -L$prefix/lib/mysql -lmysqlclient \  
-o myapp myapp.o
```

- Extra flags may be needed depending on your platform

Using `mysql_config`

- Compile:

```
$CC `mysql_config -cflags` -c myapp.c
```

- Link:

```
$LD `mysql_config -libs` -o myapp myapp.o
```

- Or for multi threaded applications:

```
$LD `mysql_config -libs_r` -o myapp myapp.o
```

- Or to embed the server:

```
$LD `mysql_config -libmysqld_libs` \  
-o myapp myapp.o
```

Using GNU autotools & libtool

The Idea: automate as much as possible

- Detect installed things and features
- Create full featured Makefiles with lots of extra convenience targets
- Support almost any unixoid build environment
- Hide platform specific build differences by using libtool

Autotools project template

I created a `mysql.m4` macro library that performs all necessary MySQL 'magic' in just three macros:

- `WITH_MYSQL()` enables the general `--with-mysql` configure option
- `MYSQL_USE_CLIENT_API()` specifies the API to use
- `MYSQL_SUBST()` makes settings by the previous two macros permanent

Required autotools input files

- `configure.in` – takes project name and version
- `Makefile.am` – takes source and binary names
- `mysql.m4` – macro library
- `ax_compare_version.m4` – an included macro
- `acinclude.m4` – includes `mysql.m4`
- `README`, `NEWS`, `AUTHORS`, `ChangeLog`
- required by automake, can be empty

acinclude.m4

```
m4_include([./mysql.m4])
```

configure.in

```
AC_INIT(api-examples, 1.0)  
AM_INIT_AUTOMAKE  
AC_CONFIG_HEADERS(config.h)  
AC_PROG_LIBTOOL  
WITH_MYSQL()  
MYSQL_USE_CLIENT_API()  
MYSQL_SUBST()  
AC_OUTPUT(Makefile)
```

Makefile.am

```
AM_CFLAGS=@MYSQL_CFLAGS@
```

```
AM_CXXFLAGS=@MYSQL_CXXFLAGS@
```

```
AM_LDFLAGS=@MYSQL_LDFLAGS@
```

```
AM_LIBS=@MYSQL_LIBS@
```

```
bin_PROGRAMS = my_app
```

```
my_app_SOURCES = my_app.c
```

Prepare for build

To generate a working configure file out of configure.in and Makefile.am you need to run

```
touch NEWS README AUTHORS ChangeLog  
autoreconf -v -i
```

This will analyze the input files and then invoke all necessary autotools commands in the right order.

Configure and make

The generated **configure** file provides

- **--with-mysql**: either the MySQL installation prefix or the full path to **mysql_config**
- **--enable-embedded-mysql**: build embedded server application instead of regular client
- Plus all standard configure options like **--prefix**

To build use the typical

```
./configure . . . ; make ; make install
```

Extra make targets

- `make install`
- `make uninstall`
- `make dist`
- `make distcheck`
- `make clean / distclean`
`/maintainerclean`
- ...

Life cycle of a client application

- Include `mysql.h`
- Initialize the library
- Connect to a MySQL server
- Perform queries
- Close connection
- Shut down library

Library init and shutdown

```
#include <mysql.h>

int main(int argc, char **argv)
{
    MYSQL *mysql = NULL;
    mysql = mysql_init(mysql);
    ...
    mysql_close(mysql);
}
```

Connect to a server

```
mysql_real_connect (  
    mysql, /* MYSQL structure to use */  
    "localhost", /* server hostname or IP address */  
    "root",      /* mysql user */  
    "secret",    /* password */  
    "test",     /* default database to use */  
    3306,       /* port number, 0 for default */  
    "/tmp/mysql.sock", /* socket file */  
    CLIENT_FOUND_ROWS /* connection flags */ );  
  
...  
mysql_close (mysql) ;
```

Extras for embedded server

```
int main (int argc, char** argv)
{
    mysql_library_init (argc, argv, NULL);
    ...
    mysql_library_end();
}
```

Life cycle of a regular query

- Issue the query
- Fetch result handle
- Fetch rows one by one
- Close result handle

Issue the query

```
if (mysql_query(mysql,  
    "SELECT * FROM City LIMIT 10"))  
{  
    printf("Query failed: %s\n",  
        mysql_error(mysql));  
    exit(3);  
}
```

Fetch the result

```
MYSQL_RES *result  
    = mysql_(store|use)_result(mysql) ;  
  
...  
mysql_free_result(mysql) ;
```

Store vs. Use result

- **mysql_store_result ()** fetches all result rows from the server and stores them locally before returning, so freeing the connection for new commands
- **mysql_use_result ()** does only prepare result fetching, the actual fetch functions will then wait for data to arrive

Fetch rows

```
MYSQL_ROW row;

unsigned int i, num_fields;

mysql_num_fields(result);

while ((row = mysql_fetch_row(result))) {
    for (i = 0; i < num_fields; i++) {
        printf("%s, ", row[i]);
    }
    putchar('\n');
}
```

Life cycle of a prepared statement query

- Prepare query
- Bind parameter and result variables
- Execute prepared statement
- Store or use result
- Fetch rows one by one
- Close result
- Execute again with changed variable values
- Or close statement

Life cycle of a stored procedure call

Stored procedure calls can return multiple result sets:

- Execute CALL query
- Store or use result
- Fetch rows one by one
- Close result
- Move to next result, then go to “Store or use”
- ... until all results are processed