

Introduction to the Plugin Code

(thanks for getting up)

Why should I care?

- Single execution point on startup.
- Greater modularity
- “One way to do it” means its easier for new people to pick up the code.
- Much easier to try and test/remove as needed new code.
- Because all of the cool kids are doing it...

Relevant Code?

- `include/mysql/plugin.h`
- `sql/sql_plugin.cc`
- `sql/sql_plugin.h`
- `plugins/`

Plugin Declaration

```
/*  
  The allowable types of plugins  
*/  
#define MYSQL_UDF_PLUGIN          0 /* User-defined function */  
#define MYSQL_STORAGE_ENGINE_PLUGIN 1 /* Storage Engine */  
#define MYSQL_FTPARSER_PLUGIN    2 /* Full-text parser plugin */  
#define MYSQL_DAEMON_PLUGIN      3 /* The daemon/raw plugin type */  
#define MYSQL_INFORMATION_SCHEMA_PLUGIN 4 /* The I_S plugin type */  
#define MYSQL_MAX_PLUGIN_TYPE_NUM 5 /* The number of plugin types */
```

Plugin Structure

```
struct st_mysql_plugin
{
    int type;          /* the plugin type (a MYSQL_XXX_PLUGIN value) */
    void *info;       /* pointer to type-specific plugin descriptor */
    const char *name; /* plugin name */
    const char *author; /* plugin author (for SHOW PLUGINS) */
    const char *descr; /* general descriptive text (for SHOW PLUGINS) */
    int license;      /* the plugin license (PLUGIN_LICENSE_XXX) */
    int (*init)(void *); /* the function to invoke when plugin is loaded */
    int (*deinit)(void *); /* the function to invoke when plugin is unloaded */
    unsigned int version; /* plugin version (for SHOW PLUGINS) */
    struct st_mysql_show_var *status_vars;
    void * __reserved1; /* placeholder for system variables */
    void * __reserved2; /* placeholder for config options */
};
```

An Example Structure

```
mysql_declare_plugin(daemon_example)
{
    MYSQL_DAEMON_PLUGIN,
    &daemon_example_plugin,
    "daemon_example",
    "Brian Aker",
    "Daemon example, creates a heartbeat beat file in mysql-heartbeat.log",
    PLUGIN_LICENSE_GPL,
    daemon_example_plugin_init, /* Plugin Init */
    daemon_example_plugin_deinit, /* Plugin Deinit */
    0x0100 /* 1.0 */,
    NULL, /* status variables */
    NULL, /* system variables */
    NULL /* config options */
}
```

And how do they init?

```
plugin_type_init plugin_type_initialize  
[MYSQL_MAX_PLUGIN_TYPE_NUM]=  
{  
    0,ha_initialize_handlerton,0,0,initialize_schema_table  
};
```

The main execute...

```
static int plugin_initialize(struct st_plugin_int *plugin)
{
    DEBUG_ENTER("plugin_initialize");

    if (plugin_type_initialize[plugin->plugin->type])
    {
        if ((*plugin_type_initialize[plugin->plugin->type])(plugin))
        {
            sql_print_error("Plugin '%s' registration as a %s failed.",
                plugin->name.str, plugin_type_names[plugin->plugin->type].str);
            goto err;
        }
    }
    else if (plugin->plugin->init)
    {
        if (plugin->plugin->init(plugin))
        {
            sql_print_error("Plugin '%s' init function returned error.",
                plugin->name.str);
            goto err;
        }
    }

    plugin->state= PLUGIN_IS_READY;
```

How does this work?

- Lets look at information schema.
 - initialize_schema_table()
 - schema_tables_add()
 - find_schema_table()
 - finalize_schema_table()

```

int initialize_schema_table(st_plugin_int *plugin)
{
    ST_SCHEMA_TABLE *schema_table;
    DBUG_ENTER("initialize_schema_table");

    if (!(schema_table= (ST_SCHEMA_TABLE *)my_malloc(sizeof(ST_SCHEMA_TABLE),
        MYF(MY_WME | MY_ZEROFILL))))
        DBUG_RETURN(1);
    /* Historical Requirement */
    plugin->data= schema_table; // shortcut for the future
    if (plugin->plugin->init)
    {
        schema_table->create_table= create_schema_table;
        schema_table->old_format= make_old_format;
        schema_table->idx_field1 = -1,
        schema_table->idx_field2= -1;

        if (plugin->plugin->init(schema_table))
        {
            sql_print_error("Plugin '%s' init function returned error.",
                plugin->name.str);
            goto err;
        }
        schema_table->table_name= plugin->name.str;
    }

    DBUG_RETURN(0);
err:
    my_free((gptr)schema_table, MYF(0));
    DBUG_RETURN(1);
}

```

initialize_schema_table()

schema_tables_add()

```
add_data.files= files;
add_data.wild= wild;
if (plugin_foreach(thd, add_schema_table,
                  MYSQL_INFORMATION_SCHEMA_PLUGIN, &add_data))
    DEBUG_RETURN(1);

DEBUG_RETURN(0);
}
```

find_schema_table()

```
ST_SCHEMA_TABLE *find_schema_table(THD *thd, const char* table_name)
{
    schema_table_ref schema_table_a;
    ST_SCHEMA_TABLE *schema_table= schema_tables;
    DEBUG_ENTER("find_schema_table");

    for (; schema_table->table_name; schema_table++)
    {
        if (!my_strcasecmp(system_charset_info,
                           schema_table->table_name,
                           table_name))
            DEBUG_RETURN(schema_table);
    }

    schema_table_a.table_name= table_name;
    if (plugin_foreach(thd, find_schema_table_in_plugin,
                      MYSQL_INFORMATION_SCHEMA_PLUGIN, &schema_table_a))
        DEBUG_RETURN(schema_table_a.schema_table);

    DEBUG_RETURN(NULL);
}
```

finalize_schema_table()

```
int finalize_schema_table(st_plugin_int *plugin)
{
    ST_SCHEMA_TABLE *schema_table= (ST_SCHEMA_TABLE *)plugin->data;
    DEBUG_ENTER("finalize_schema_table");

    if (schema_table && plugin->plugin->deinit)
    {
        DEBUG_PRINT("info", ("Deinitializing plugin: '%s'", plugin->name.str));
        if (plugin->plugin->deinit(NULL))
        {
            DEBUG_PRINT("warning", ("Plugin '%s' deinit function returned error.",
                plugin->name.str));
        }
        my_free((gptr)schema_table, MYF(0));
    }

    DEBUG_RETURN(0);
}
```

SQL?

- LOAD PLUGIN
- UNLOAD PLUGIN
- show plugins;
- select * from information_schema.plugins;

plugin.in

`MYSQL_STORAGE_ENGINE(myisam,no,
[MyISAM Storage Engine],`

`[Traditional non-transactional MySQL tables])`

`MYSQL_PLUGIN_DIRECTORY(myisam, [storage/myisam])`

`MYSQL_PLUGIN_STATIC(myisam, [libmyisam.a])`

`MYSQL_PLUGIN_MANDATORY(myisam) dnl Default`

`MYSQL_PLUGIN_DEPENDS_ON_MYSQL_INTERNALS
(myisam, [ha_myisam.cc])`

plugin.in (another)

```
MYSQL_PLUGIN(daemon_example,[Daemon Example  
Plugin],
```

```
    [This is an example plugin daemon.]
```

```
MYSQL_PLUGIN_DYNAMIC(daemon_example,  
[libdaemon_example.la])
```