MVProc-FCGI is a Model-View-Controller web framework that uses MySQL stored procedures as the controller element. It is implemented as a FastCGI executable for stability and portability.

There are currently no plans to implement this for any other database, as no other database has enough flexibility of functionality in the context of stored routines. (At least as far as I'm aware.)

Installation

MVProc-FCGI has the following prerequisites:

- libfcgi
- libmysqlclient_r
- libcrypto

There are a few #define statements in the mvproc.h file that can be edited, including:

- CONFIG_LOCATION [default /etc/mvproc.conf]
- INIT_ERR_PATH [default /tmp/mvproc_init.err]
- MEM_PAGE_SIZE [default 10240] – while a 10K page size should be a good balance for most usage, some sites may generally require more memory per request. While pages are added as needed (and larger allocs get their own custom size “page”), if your transactions routinely use 25K, for example, increasing MEM_PAGE_SIZE to 30K might be a good idea. Request use of memory does not include raw POST data, which is malloc'd and free'd separately, but does include parsed input (names, filenames, NOT files) and database output.
- DEFAULT_MAX_CONTENT_LENGTH [this is set at 25MB] – completely unnecessary to edit this, as it can be overridden in the config file (see below).

Configuration

Please consult your webserver's documentation for implementing FastCGI executables. Below are some suggested configurations:

Apache using mod_fcgid

```
FcgidWrapper /path/to/executable/mvproc_fcgi virtual

<Location "/cgi">
        SetHandler fcgid-script
</Location>

<Directory /path/to/docroot/>
        Options +FollowSymLinks +ExecCGI
        AllowOverride None
        Require all granted
        RewriteEngine On
        RewriteCond %{REQUEST_FILENAME} -f
        RewriteRule ^.*$ - [NC,L]
        RewriteRule ^.*$ cgi [NC,L]
</Directory>
```

Lighttpd

```
server.error-handler-404    = "/all.mvproc"
static-file.exclude-extensions = ( ".mvproc" )
```

and in the 10-fastcgi.conf file:

```
fastcgi.server = ( ".mvproc" =>
  ((
      "host" => "127.0.0.1" ,
      "port" => 1026 ,
      "bin-path" => "/path/to/mvproc_fcgi",
      "check-local" => "disable",
      "max-procs" => 10
    ))
)
```

Nginx
I couldn't get things to work fully on nginx (some parameters would not come through, some did); maybe someone else can?

```
location / {
        try_files $uri =404;
}

error_page 404 /all.mvproc;

location ~ \.mvproc$ {
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_param   QUERY_STRING            $query_string;
        fastcgi_param   REQUEST_METHOD          $request_method;
        fastcgi_param   CONTENT_TYPE            $content_type;
        fastcgi_param   CONTENT_LENGTH          $content_length;
        fastcgi_param   REQUEST_URI             $request_uri;
        fastcgi_param   REMOTE_ADDR             $remote_addr;
        fastcgi_param   SERVER_NAME             $server_name;
}
```

The above configurations send all requests not matching a file to mvproc_fcgi. This is important for implementing MVProc-FCGI because the URI must match the name of a stored procedure (ie. http://yoursite.org/MyProc).

There are twelve (12) configuration directives for mod_mvproc:

session (Y or N) – sets session behavior. If sessions are enabled (Y), MVProc will supply called stored procedures with a user variable @mvp_session, which will contain a 32 character session id. A cookie will be produced and maintained with this id as its value. The procedure may set the @mvp_session value, which will update the value of the cookie. The value is also included in the output of the call in the PROC_OUT section.

dbGroup (yourDBgroup) – this is the group in mysql's my.cnf file which MVProc will use to authenticate and connect to the database. An example:

```
[mydb]
host     = localhost
port     = 3306
user     = myuser
password = mypass
database = webdb
```

Please refer to the MySQL documentation for my.cnf subtleties.
Do consider that my.cnf is accessible. Best practice is to create a user (let's say 'foo'@'localhost') for an MVProc database (let's say `bar`) and grant only minimal access. Most of the time, this would be:
GRANT SELECT (`name`,`param_list`,`db`,`type`) ON `mysql`.`proc` TO 'foo'@'localhost';
(The above is the minimum required permission for MVProc to work at all.)
GRANT EXECUTE ON `bar`.* TO 'foo'@'localhost';
The procedure will run with the permissions of its DEFINER (see mysql documentation).

templateDir (/your/template/directory) – the location of the templates. This is best placed outside the doc root – in fact, if the template files are in the doc root Apache (and maybe others) will get confused, thinking it should serve the file instead of allowing MVProc to handle the transaction. If templateDir is not set, or set with an empty string, templates and layouts will not be used. More on templates later.

defaultLayout (yourLayoutTemplate – leave off the '.tpl') – If set, this will set the @mvp_layout session variable to a default value. It's not required to set this in the configuration, as procedures can set it as well. More on layouts later.

defaultProc (yourDefaultProc) – If set, this will be the procedure that's called when the request uri is '/' or if the uri does not match a procedure. A non-proc uri is allowed so pages can be given interesting urls like http://my.site.org/i-like-cheese. Do remember to handle possible non-proc uris in your default proc.

cache (Y or N) – Y enables caching of procs AND templates. Set to N during development and Y for production use. Also available are T – cache templates only, and D – cache database only.

outputStyle (MIXED | PLAIN | JSON | EASY_XML | EASY_JSON) – see the Output section below.

errTemplate – If templateDir is configured, this is the name of the template to render when there's a database error. The error can be accessed as `<# status.error #>`.

allowSetContent (Y or N) – provides @mvp_content_type session variable, which allows a procedure to tell the webserver how to set the Content-Type header.

allowHTMLfromDB (Y or N) – default N, Y allows HTML output from the database to be rendered as HTML. By default, MVProc will convert '<' to '<', '&' to '"', etc. to prevent XXS and CSRF attacks.

uploadDirectory - Sets the path to a directory where uploaded files are written. It is highly recommended that this path be OUTSIDE the docroot. If the path doesn't exist, MVProc will attempt to create it. Default value is /tmp

maxPostSize – Sets the maximum size for POST requests. Because FastCGI handler modules for various webservers generally will be configurable to filter over-large requests, it's better to let them handle that filtering. That being said, if you want to exceed the default value of 25MB, this will need to be set. Note: Currently, this value needs to be set in number of bytes. (25MB = 26214400 bytes)

Requests

Requests are typical “pretty” web format, like so: http://mysite.com/procname. If the url points to a file that exists, the module will decline handling, which means the file will be served as normal. If the uri is empty (ie. nothing or '/' after the url) or the uri does not match a stored procedure, the procedure specified by the mvprocDefaultProc configuration directive or 'landing' will be looked for. See mvprocDefaultProc config directive.

GET and POST requests are supported, as well as file uploads. A file upload will be written to the configured uploadDirectory [default '/tmp'] with a pseudo-random name plus the original file extension. This filename will be reported to the procedure through the uploaded file's variable name. So

<input type=”file” name=”myfile”>

will send something like '/tmp/F228FA.jpg' into the procedure as the IN or INOUT argument 'myfile', but ONLY IF THE ARGUMENT EXISTS in the procedure's definition.

Procedures

All aspects of MySQL stored procedures are supported. IN, INOUT, OUT parameters and multiple result sets are available to the template parser and are shown in xml or json output. User variables available to procedures are currently:

- `@mvp_servername` – the server hostname (mysite.com)
- `@mvp_requestmethod` – GET, POST, or REQUEST
- `@mvp_uri` – the unparsed uri of the request
- `@mvp_template` – by default, the procedure name – this value can be changed in the procedure to tell the module to use a different template file. This can be 'procname' or 'myother_template' or even (under the template directory) 'my_subdirectory/my_other_sub/even_farther/as_deep_as_you_like/template_name'
  Remember to leave off the '.tpl'
- `@mvp_layout` – the layout template to use – this value can be set or changed in the procedure. Handling is essentially the same as for @mvp_template.
- `@mvp_agent` – the Browser's User-Agent header value
- `@mvp_remoteip` – the requestor's ip address
- `@mvp_session` – (if mvprocSession = 'Y') the current value of the MVPSESSION cookie
  This can be set in the procedure, but the module will set it up and provide a value by default. Once overridden, the set value will be returned with each subsequent request. (Just like a session... heh.) MVProc will only recognize an alphanumeric MVPSESSION cookie.
- `@mvp_content_type` – (if mvprocAllowSetContent = 'Y') – This is passed in as an empty string. It's only used in output if set by the procedure.

Known Issues

Returning multiple result sets with the same table name causes libmysqlclient to seg fault. This includes calculated selects. So if you want to do this:

```
SELECT 'a value' AS one_value; SELECT 'more value' AS too_value;
```

Instead, do this:

```
SELECT 'a value' AS one_value, 'more value' AS too_value;
```

And if you want to return multiple result sets from the same table, use aliases.

If libmysqlclient starts to allow multiple result sets from the same table, be cautious about using EASY_XML and EASY_JSON output. The results are, at this point, undefined.

Output

XML MIXED is the default output of a request. If no template exists, the module goes with this. Result sets are rendered in xml as a <table> element with child <row> elements.

For the MIXED output type, columns whose declared size is 32 or less (for example `VARCHAR(24)`) are shown as attributes of rows, while blobs and columns of greater size (like `VARBINARY(4096)` or `CHAR(65)`) are shown as children of rows with CDATA encapsulated values.

The PLAIN output type populates the name attribute of the table tags, but all columns are rendered as CDATA encapsulated child elements.

Another option is JSON, which outputs an array of objects named 'table', each of which owns a name value and an array of row objects each of which owns one named value per column. Most ajax libraries support this format.

EASY_XML is the PLAIN type, except that instead of <table name="tableName"> elements, the output is written like <tableName>. This may cause problems if multiple result sets from the same table are output.

EASY_JSON is JSON, except that instead of an array of objects named 'table', the output is an object with each result set as an element named for its table. This may cause problems if multiple result sets from the same table are output.

All result sets are output, as well as a "table" called PROC_OUT, which holds all INOUT and OUT values as well as mvp_session and mvp_template values. If a table is not declared in a select out (eg. SELECT 'whatever' AS myval) the table will be called "status".

Here's an example of MIXED:

```
<results>
    <table name="widgets">
        <row idwidget="1" label="demo" version="1.00">
            <description>A demo for testing</description>
        </row>
    </table>
    <table name="PROC_OUT">
        <row>
            <mvp_session>19e820417390d3bf564261886d70ea64</mvp_session>
            <mvp_template>getWidgets</mvp_template>
        </row>
    </table>
</results>
```

Errors will look like this:

```
<results>
    <table name="status">
        <row>
            <error>Please consult your documentation for correct usage.</error>
        </row>
    </table>
</results>
```

Templates

Templates are typically html pages with tags for the parser to use to plug in values, loop through result sets, conditionally show or not show markup, include other templates. Some simple rules:

- an MVProc tag looks like this: <# the tag is in here #>.
- String literals are between single quotes.
- Inside single quotes you can have any characters you want – the quotes escape everything.
- Escape single quotes within single quotes with a backslash: \'
- The tag names must be either ALL CAPS (ELSEIF) or all lower (elseif).

The template tags are intentionally few in number. It encourages separation of concern.

- Value - <# [table.]field_name[[row_num]] #>
  The default table is PROC_OUT, and the default row_num is 0. Inside a LOOP, the default table becomes the LOOP table and the default row becomes the LOOP's CURRENT_ROW.
  CURRENT_ROW (all caps) is a built-in value of type INT (actually unsigned long in C terms).
  NUM_ROWS (all caps) is a built-in value of type INT accessed like: <# IF tableName.NUM_ROWS > 0 #>
  The '@' table holds user variables for the templates (see SET).
- **IF** - <# IF myvar = 'hello' #>
  Supported comparison operators are: =, ==, !=, !, <>, <, >, <=, and >=
  With no operator, a value equals true if non-zero (int & float) or non-empty (string).
  The not operator (!) equals true if zero (int & float) or empty (string).
  Nesting is supported, as well as AND, and, OR, or, &&, and ||
  AND, and, and && take precedence over OR, or, and ||
  Example: <# IF mytable.what[2] = 'ok' AND (CURRENT_ROW > 4 OR !@.checkit) #>
  Nesting is arbitrarily limited to a depth of 64.
  No math is supported in IF tags.
  Constants are supported: strings are quoted with single quotes and floats must have a '.' (eg 0.0)
- **ELSIF** - <# ELSIF @.val_is_set #> - Identical parsing and evaluation to IF.
- **ELSE** - <# ELSE #> - This functions as anyone would expect.
- **ENDIF** - <# ENDIF #> - Again, like anyone would expect. This tag is REQUIRED with IF usage.
- **LOOP** - <# LOOP mytable #>
  The LOOP tag begins a template segment that will iterate once for each row in a result set.
  Inside the LOOP, the table specified becomes the default table, and CURRENT_ROW will evaluate to the current row (zero indexed).
- **ENDLOOP** - <# ENDLOOP #> - Closes a loop. This tag is REQUIRED for each LOOP started.
- **INCLUDE** - <# INCLUDE another_template #>
  The specified template will be included at the tag's position.
  The included template is referenced from the configured templateDir, so if your template directory is /var/templates and you <# INCLUDE layouts/header #>, the file looked for will be /var/templates/layouts/header.tpl
  Always leave the '.tpl' off the include argument.
  INCLUDE can also accept a value tag like: <# INCLUDE myTable.myValue[4] #>
- **TEMPLATE** - <# TEMPLATE #> - This is essentially an include tag, except that it uses the @mvp_template session variable. Use this convenience tag inside a layout template.
- **SET** - <# SET myvar = 'ok' #>
  One could create a very rich site with lots of functionality without using this tag. I haven't benchmarked the difference between setting all required values in the procedure vs. setting some in the template but I suspect SET might be a bit slower, and it's certainly less efficient with memory. That being said, is it available.
  The SET tag sets a value in the '@' table, which supports only one row.
  <# SET row_class = 'color' + CURRENT_ROW % 2 + 1 #> would be referenced
  <# @.row_class #> (and this is useful for alternating row css style)
  Supported operators are =, +, -, *, /, %, and comma(,)
  String “math” supports only concatenation (+), ints and floats use C-style math.
  *, /, and % take precedence over + and -.
  Modulus (%) evaluates to the remainder for int values and the fraction for floats.
  Constants are supported: strings are quoted with single quotes and floats must have a '.' (eg 0.0)
  The comma is for multiple assignments in a tag, like so: <# SET a = 0, b = 'hi', c = @.a / 1 #>

And that's it.

I think. Any questions, bugs, requests – please post on sourceforge.net

-Jeff Walter
maintainer, MVProc-FCGI