

MVProc is a Model-View-Controller web framework that uses MySQL stored procedures as the controller element. It is implemented as an Apache2 module, mod_mvproc.

There are currently no plans to implement this for any other database, as no other database has the flexibility of MySQL in the context of stored routines.

Installation

The Makefile and #includes section of mvproc.h need to be edited before compiling, specifically the APACHE_HOME variable and location of apr and apreq. I built apache from source and linked against that, but Ubuntu puts include files in slightly different locations.

The debian package is also available.

Configuration

Since mod_mvproc is a module it must be loaded in your apache configuration file. apreq_module is required and must be loaded prior to loading mvproc, like so:

```
LoadModule apreq_module /home/jeff/apache2/modules/mod_apreq2.so
LoadModule mvproc_module /home/jeff/apache2/modules/mod_mvproc.so
```

There are four (4) configuration directives for mod_mvproc:

mvprocSession (Y or N) – sets session behavior. If sessions are enabled (Y), the module 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.

mvprocDbGroup (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
user      = myuser
password  = mypass
database  = webdb
```

mvprocTemplateDir (/your/template/directory) – the location of the templates. This should probably be outside the webroot. More on templates later.

mvprocCache (Y or N) – Y enables caching of procs AND templates. Set to N during development and Y for production use.

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 url is requested without a procedure specified, a procedure named 'landing' will be looked for. If a procedure does not exist with the requested name, a 500 error will be generated.

GET and POST requests are supported, as well as file uploads. A file upload will be written to the system's temp directory 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/87YHF228FA.jpg` into the procedure as the IN or INOUT argument 'myfile' 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 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_further/as_deep_as_you_like/template_name'`
Remember to leave off the '.tpl'
- @mvp_headers – the full request
- @mvp_requesttype – the uri scheme
- @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.) You could even store data in the cookie if it's ok for everyone to see. The only size limit is what a browser will actually send and receive in a cookie.

XML Output

XML is the default output of a request. If no template exists, the module goes with this. Great for ajax calls. Result sets are rendered in xml as a <table> element with child <row> elements. Non-blob columns are shown as attributes of rows, while blobs are shown as children of rows with CDATA encapsulated values. 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”. This is not preferable, as only the first such table will be seen by the parser. But if you're ready to handle multiple tables named “status” and don't plan to use the template parser, feel free.

Here's an example:

```
<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_template="getWidgets"/>
  </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).
The '@' table holds user variables for the templates (see SET).
The tag is `<# IF myvar = 'hello' #>`
- **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 mvprocTemplateDir, 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.
- **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