1   Apache::test - Facilitates testing of Apache::* modules

1.1 Synopsis

```
# In Makefile.PL
use Apache::test;
my %params = Apache::test->get_test_params();
Apache::test->write_httpd_conf(%params, include => $more_directives);
*MY::test = sub { Apache::test->MM_test(%params) };

# In t/*.t script (or test.pl)
use Apache::test qw(skip_test have_httpd);
skip_test unless have_httpd;
(Some more methods of Doug's that I haven't reviewed or documented yet)
```

1.2 Description

This module helps authors of `Apache::*` modules write test suites that can query an actual running Apache server with mod_perl and their modules loaded into it.

Its functionality is generally separated into methods that go in a *Makefile.PL* to configure, start, and stop the server, and methods that go in one of the test scripts to make HTTP queries and manage the results.

1.3 Methods

1.3.1 get_test_params()

This will ask the user a few questions about where the httpd binary is, and what user/group/port should be used when running the server. It will return a hash of the information it discovers. This hash is suitable for passing to the `write_httpd_conf()` method.

1.3.2 write_httpd_conf(%params)

This will write a basic *httpd.conf* file suitable for starting a HTTP server during the `make test` stage. A hash of key/value pairs that affect the written file can be passed as arguments. The following keys are recognized:

- **conf_file**

  The path to the file that will be created. Default is *t/httpd.conf*.

- **port**

  The port that the Apache server will listen on.

- **user**

  The user that the Apache server will run as.

- **group**

  The group that the Apache server will run as.

- **include**

  Any additional text you want added at the end of the config file. Typically you'll have some `PerlModule` and `Perl*Handler` directives to pass control to the module you're testing. The `blib/` directories will be added to the `@INC` path when searching for modules, so that's nice.

1.3.3 MM_test(%params)

This method helps write a Makefile that supports running a web server during the `make test` stage. When you execute `make test`, make will run `make start_httpd`, `make run_tests`, and `make kill_httpd` in sequence. You can also run these commands independently if you want.

Pass the hash of parameters returned by `get_test_params()` as an argument to `MM_test()`.

To patch into the `ExtUtils::MakeMaker` wizardry (voodoo?), typically you'll do the following in your *Makefile.PL*:

```
  *MY::test = sub { Apache::test->MM_test(%params) };
```

1.3.4 fetch

```
  Apache::test->fetch($request);
  Apache::test->fetch($user_agent, $request);
```

Call this method in a test script in order to fetch a page from the running web server. If you pass two arguments, the first should be an `LWP::UserAgent` object, and the second should specify the request to make of the server. If you only pass one argument, it specifies the request to make.

The request can be specified either by a simple string indicating the URI to fetch, or by a hash reference, which gives you more control over the request. The following keys are recognized in the hash:

- **uri**

  The URI to fetch from the server. If the URI does not begin with `http`, we prepend `http://localhost:$PORT` so that we make requests of the test server.

- **method**

  The request method to use. Default is `GET`.

- **content**

  The request content body. Typically used to simulate HTML fill-out form submission for `POST` requests. Default is null.

- **headers**

  A hash of headers you want sent with the request. You might use this to send cookies or provide some application-specific header.

If you don't provide a `headers` parameter and you set the `method` to `POST`, then we assume that you're trying to simulate HTML form submission and we add a `Content-Type` header with a value of `application/x-www-form-urlencoded`.

In a scalar context, `fetch()` returns the content of the web server's response. In a list context, `fetch()` returns the content and the `HTTP::Response` object itself. This can be handy if you need to check the response headers, or the HTTP return code, or whatever.

1.3.5 *static_modules*

```
Example: $mods = Apache::test->static_modules('/path/to/httpd');
```

This method returns a hashref whose keys are all the modules statically compiled into the given httpd binary. The corresponding values are all 1.

1.4 Examples

No good examples yet. Example submissions are welcome. In the meantime, see http://forum.swarthmore.edu/~ken/modules/Apache-AuthCookie/ , which I'm retrofitting to use `Apache::test`.

1.5 To Do

The `MM_test` method doesn't try to be very smart, it just writes the text that seems to work in my configuration. I am morally against using the `make` command for installing Perl modules (though of course I do it anyway), so I haven't looked into this very much. Send bug reports or better (patches).

I've got lots of code in my `Apache::AuthCookie` module (etc.) that assists in actually making the queries of the running server. I plan to add that to this module, but first I need to compare what's already here that does the same stuff.

1.6 Kudos

To Doug MacEachern for writing the first version of this module.

To caelum@debian.org (Rafael Kitover) for contributing the code to parse existing httpd.conf files for `--enable-shared=max` and DSOs.

1.7 Caveats

Except for making sure that the mod_perl distribution itself can run `make test` okay, I haven't tried very hard to keep compatibility with older versions of this module. In particular `MM_test()` has changed and probably isn't usable in the old ways, since some of its assumptions are gone. But none of this was ever documented, and `MM_test()` doesn't seem to actually be used anywhere in the mod_perl disribution, so I don't feel so bad about it.

1.8 Maintainers

Maintainer is the person(s) you should contact with updates, corrections and patches.

- **The documentation mailing list**

1.9 Authors

- **Doug MacEachern**
- **Ken Williams**

Only the major authors are listed above. For contributors see the Changes file.

Table of Contents: