1 Apache::File - advanced functions for manipulating files at the server side

1.1 Synopsis

```
use Apache::File ();

my $fh = Apache::File->new($filename);
print $fh 'Hello';
$fh->close;

my($name, $fh) = Apache::File->tmpfile;

if ((my $rc = $r->discard_request_body) != OK) {
   return $rc;
}

if((my $rc = $r->meets_conditions) != OK) {
   return $rc;
}

my $date_string = localtime $r->mtime;

$r->set_content_length;
$r->set_etag;
$r->update_mtime;
$r->set_last_modified;
```

1.2 Description

`Apache::File` does two things: it provides an object-oriented interface to filehandles similar to Perl's standard `IO::File class`. While the `Apache::File` module does not provide all the functionality of `IO::File`, its methods are approximately twice as fast as the equivalent `IO::File` methods. Secondly, when you use `Apache::File`, it adds several new methods to the `Apache` class which provide support for handling files under the `HTTP/1.1` protocol.

1.3 Apache::File methods

- **new()**

  This method creates a new filehandle, returning the filehandle object on success, undef on failure. If an additional argument is given, it will be passed to the `open()` method automatically.

  ```
  use Apache::File ();
  my $fh = Apache::File->new;

  my $fh = Apache::File->new($filename) or die "Can't open $filename $!";
  ```

- **open()**

  Given an Apache::File object previously created with `new()`, this method opens a file and associates it with the object. The `open()` method accepts the same types of arguments as the standard Perl `open()` function, including support for file modes.

```
$fh->open($filename);
```

```
$fh->open(">$out_file");
```

```
$fh->open("|$program");
```

- **close()**

  The `close()` method is equivalent to the Perl builtin close function, returns true upon success, false upon failure.

  ```
  $fh->close or die "Can’t close $filename $!";
  ```

- **tmpfile()**

  The `tmpfile()` method is responsible for opening up a unique temporary file. It is similar to the `tmpnam()` function in the `POSIX` module, but doesn’t come with all the memory overhead that loading `POSIX` does. It will choose a suitable temporary directory (which must be writable by the Web server process). It then generates a series of filenames using the current process ID and the `$TMPNAM` package global. Once a unique name is found, it is opened for writing, using flags that will cause the file to be created only if it does not already exist. This prevents race conditions in which the function finds what seems to be an unused name, but someone else claims the same name before it can be created.

  As an added bonus, `tmpfile()` calls the `register_cleanup()` method behind the scenes to make sure the file is unlinked after the transaction is finished.

  Called in a list context, `tmpfile()` returns the temporary file name and a filehandle opened for reading and writing. In a scalar context only the filehandle is returned.

  ```
  my($tmpnam, $fh) = Apache::File->tmpfile;
  ```

  ```
  my $fh = Apache::File->tmpfile;
  ```

1.4 Apache Methods added by Apache::File

When a handler pulls in `Apache::File`, the module adds a number of new methods to the Apache request object. These methods are generally of interest to handlers that wish to serve static files from disk or memory using the features of the `HTTP/1.1` protocol that provide increased performance through client-side document caching.

- **$r->discard_request_body()**

  This method tests for the existence of a request body and if present, simply throws away the data. This discarding is especially important when persistent connections are being used, so that the request body will not be attached to the next request. If the request is malformed, an error code will be returned, which the module handler should propagate back to Apache.

```
if ((my $rc = $r->discard_request_body) != OK) {
   return $rc;
}
```

- **$r->meets_conditions()**

  In the interest of HTTP/1.1 compliance, the `meets_conditions()` method is used to implement ‘‘conditional GET’’ rules. These rules include inspection of client headers, including `If-Modified-Since`, `If-Unmodified-Since`, `If-Match` and `If-None-Match`.

  As far as Apache modules are concerned, they need only check the return value of this method before sending a request body. If the return value is anything other than `OK`, the module should return from the handler with that value. A common return value other than `OK` is `HTTP_NOT_MODIFIED`, which is sent when the document is already cached on the client side, and has not changed since it was cached.

  ```
  if((my $rc = $r->meets_conditions) != OK) {
     return $rc;
  }
  #else ... go and send the response body ...
  ```

- **$r->mtime()**

  This method returns the last modified time of the requested file, expressed as seconds since the epoch. The last modified time may also be changed using this method, although `update_mtime()` method is better suited to this purpose.

  ```
  my $date_string = localtime $r->mtime;
  ```

- **$r->set_content_length()**

  This method sets the outgoing `Content-length` header based on its argument, which should be expressed in byte units. If no argument is specified, the method will use the size returned by `$r->filename`. This method is a bit faster and more concise than setting `Content-length` in the headers_out table yourself.

  ```
  $r->set_content_length;
  $r->set_content_length(-s $r->finfo); #same as above
  $r->set_content_length(-s $filename);
  ```

- **$r->set_etag()**

  This method is used to set the outgoing `ETag` header corresponding to the requested file. `ETag` is an opaque string that identifies the currrent version of the file and changes whenever the file is modified. This string is tested by the `meets_conditions()` method if the client provide an `If-Match` or `If-None-Match` header.

  ```
  $r->set_etag;
  ```

- **$r->set_last_modified()**

This method is used to set the outgoing `Last-Modified header` from the value returned by `$r->mtime`. The method checks that the specified time is not in the future. In addition, using `set_last_modified()` is faster and more concise than setting `Last-Modified` in the `headers_out` table yourself.

You may provide an optional time argument, in which case the method will first call the `update_mtime()` to set the file's last modification date. It will then set the outgoing `Last-Modified` header as before.

```
$r->update_mtime((stat $r->finfo)[9]);
$r->set_last_modified;
$r->set_last_modified((stat $r->finfo)[9]); #same as the two lines above
```

- **$r->update_mtime()**

  Rather than setting the request record mtime field directly, you can use the `update_mtime()` method to change the value of this field. It will only be updated if the new time is more recent than the current mtime. If no time argument is present, the default is the last modified time of $r->filename.

  ```
  $r->update_mtime;
  $r->update_mtime((stat $r->finfo)[9]); #same as above
  $r->update_mtime(time);
  ```

1.5 Maintainers

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

- **The documentation mailing list**

1.6 Authors

- **Doug MacEachern**

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

Table of Contents: