APPENDIX B
Available Constants

This appendix lists some constants useful in programming the mod_perl API. Note that the `Apache::Constants` class provides access to nearly all the constants required of the Apache C API, some of which are only required by mod_perl internals, and others of which are rarely used in practice. Because the list is quite large (there are over 90 of them) and generally available via other documentation, only the constants that are genuinely useful to the mod_perl programmer are listed here.

Handler Return Codes

All handlers must return a meaningful status. Sometimes this status is an Apache-specific code, such as `OK`, and sometimes it is an HTTP-specific code such as `REDIRECT`. Most of these constants are defined in `httpd.h` in the Apache sources, with the exception of `DECLINE_CMD`, which is defined with the other configuration constants in `http_config.h` (see Table B.1).

Table B.1 *Apache-Specific Return Codes*

Constant	Description
`OK`	Most handlers will return `OK` to register the success of the handler to Apache.
`DECLINED`	This constant is usually reserved for telling Apache that no action was taken by the handler, that the handler declined to handle the phase of the request that it was configured to process. In some instances, such as with `PerlTransHandlers`, returning `DECLINED` is desirable even after you have inserted some processing into the request, so that your handler does not stomp on the default Apache mechanisms.
`DONE`	This return code signals the end of all processing; Apache will proceed directly to the logging phase.
`DECLINE_CMD`	Directive handlers can return `DECLINE_CMD` if they want to pass that directive back to Apache for handling. This is typically used in cases where directive handlers want to step in only if a corresponding standard Apache module is not found. However, like with `DECLINED`, it can also be used to trick Apache into thinking that your handler has not handled the directive when, in fact, some processing has occurred.

HTTP Return Codes

The HTTP-based constants are the official names for the `HTTP/1.1` status codes. Anything in the `400` or `500` series of responses is considered to be an error response. Although the constants shown in Table B.2 are the ones you are most likely to encounter, the official list is available in section 10 of the `HTTP/1.1` RFC or in `httpd.h` in the Apache sources. Both Apache and mod_perl provide aliases for the more frequently used HTTP codes in order to make code more manageable. Consult `httpd.h` for the complete list of aliases as well.

Table B.2 *HTTP Return Codes*

Constant	HTTP Status Code	Alias	Description
`HTTP_OK`	`200`	`DOCUMENT_FOLLOWS`	Although handlers ought to return `OK` and not `HTTP_OK` to indicate success, this is the constant to check when you want to know the success of an HTTP request, such as the return value of `$sub->run()`.
`HTTP_PARTIAL_CONTENT`	`206`	`PARTIAL_CONTENT`	This is used to indicate that a portion of the requested document follows. Typically, this response code is used when byteserving and is set by Apache internally via the `set_byterange()` method.
`HTTP_MOVED_PERMANENTLY`	`301`	`MOVED`	This return code states that the requested document has moved permanently to a new URI. mod_dir returns this status when it redirects requests to `/sailboat` to `/sailboat/`.
`HTTP_MOVED_TEMPORARILY`	`302`	`REDIRECT`	This return code indicates that the requested resource has moved temporarily to a new URI. `REDIRECT` is the standard return code for redirects using the `Location` header.

Table B.2 *(continued)*

Constant	HTTP Status Code	Alias	Description
HTTP_NOT_MODIFIED	304	USE_LOCAL_COPY	This indicates to the client that the document has not been modified when compared against the incoming conditional GET headers. Typically, whether a 304 is warranted is determined by the meets_conditions() method and is not calculated by handlers directly.
HTTP_UNAUTHORIZED	401	AUTH_REQUIRED	This is returned if the client did not provide proper authorization credentials for the request, typically returned by PerlAuthenHandlers and PerlAuthzHandlers.
HTTP_FORBIDDEN	403	FORBIDDEN	This is returned if the client is not allowed to access the requested document, such as with a PerlAccessHandler.
HTTP_NOT_FOUND	404	NOT_FOUND	This is returned if the requested document does not exist.
HTTP_METHOD_NOT_ALLOWED	405	METHOD_NOT_ALLOWED	This indicates that the request method (for example, GET, POST, or PUT), is not allowed for this request.

Table B.2 *(continued)*

Constant	HTTP Status Code	Alias	Description
HTTP_REQUEST_ENTITY_TOO_LARGE	413	None	This is used to indicate that the message body POSTed by the client is too large to be processed. Usually a handler does not return a value itself but instead passes it on silently from various methods, such as Apache::Request's parse().
HTTP_INTERNAL_SERVER_ERROR	500	SERVER_ERROR	This return code, which is dreaded by Web programmers everywhere, indicates that the server encountered an error in processing the request.

Directive Handler Constants

These constants correspond to the args_how field of the Apache command record as described in Chapter 7, "Creating Handlers." You can find the official definitions in http_config.h in the Apache sources. These constants can be imported into your module either explicitly or by using the :args_how import tag with Apache::Constants. Generally, however, you do not need to actually import these constants, because they are used as string literals in Makefile.PL, and Apache::ExtUtils transparently transforms the literals into the necessary constant values. Table B.3 lists the available constants, together with the respective prototype and an example parameter list.

Table B.3 *Directive Handler Constants*

Constant	Prototype	Parameter List	Description
NO_ARGS	$$	my ($cfg, $parms)	Specifies that the directive is to have no arguments. For example, CacheNegotiatedDocs.
TAKE1	$$$	my ($cfg, $parms, $arg)	Specifies that the directive takes exactly one argument, such as XBitHack.
FLAG	$$$	my ($cfg, $parms, $arg)	Specifies that the directive takes exactly one argument, but that argument must be either On or Off. The argument that is passed back in $arg is either 1 or 0, respectively. An example of this directive is ExtendedStatus.
TAKE2	$$$$	my ($cfg, $parms, $arg1, $arg2)	This specifies that the directive takes exactly two arguments. For example, LoadModule.
TAKE12	$$$;$	my ($cfg, $parms, $arg1, $arg2)	Specifies that the directive can accept either one or two arguments, such as the LogFormat directive.
TAKE3	$$$$$	my ($cfg, $parms, $arg1, $arg2, $arg3)	The directive accepts exactly three arguments. No modules in the standard distribution use this prototype.
TAKE13	*none*	my ($cfg, $parms, $arg1, $arg2, $arg3)	The directive accepts either one or three arguments. No modules in the standard distribution use this prototype. There is also no Perl subroutine prototype specified in the mod_perl sources at this time. In reality, you only need to specify a subroutine prototype if you do not specify a value to the args_how field in your Makefile.PL, so the lack of a Perl prototype does not prohibit you from using TAKE13 as a directive prototype. Apache still does the job of checking your argument list for the proper format.

Table B.3 *(continued)*

Constant	Prototype	Parameter List	Description
TAKE23	$$$$;$	my ($cfg, $parms, $arg1, $arg2, $arg3)	The directive accepts either two or three arguments, such as the CustomLog directive.
TAKE123	$$$;$$	my ($cfg, $parms, $arg1, $arg2, $arg3)	The directive accepts one, two, or three arguments. No modules in the standard distribution use this prototype. This is the default prototype when no prototype is given.
ITERATE	$$@	my ($cfg, $parms, $arg)	The directive can be called with any number of arguments. The directive handler is called once for each argument. AddHandler is an example of this prototype.
ITERATE2	$$@;@	my ($cfg, $parms, $arg1, $arg2)	The directive is called with two or more arguments. The directive handler is called once for each argument save the first, which is passed as the first argument during each iteration. The AddLanguage directive provides an example of this prototype.
RAW_ARGS	$$$;*	my ($cfg, $parms, $args, $fh)	The directive parsing is left completely to the directive handler. $args represents the remainder of the line following the directive, and $fh is an open filehandle on httpd.conf for reading the configuration data directly. In addition to all container directives, RAW_ARGS is also used for the UserDir and several other directives.

The constants in Table B.4 correspond to the req_override field of the Apache command record, as described in Chapter 7, "Creating Handlers." You can find the official definitions in http_config.h in the Apache sources. These constants can be imported into your module either explicitly or by using the :override import tag with Apache::Constants. Table B.2 lists the various places a directive can appear within a configuration. <Directory> really means <Directory>, <Location>, <Files>, and all

their regular expression matching cousins. `.htaccess` can be any file specified with the `AccessFileName` directive. Keep in mind that while these various values can be bitwise `OR`ed together, the only combination that adds any real value is `RSRC_CONF|ACCESS_CONF`, which is why it is a separate entry in the table. Also note that `OR_ALL` is the default.

Table B.4 *Directive Override Constants*

Directive Constant	Can Appear Inside `.htaccess`	Can Appear Inside `<Directory>`, etc.	Can Appear Outside `<Directory>`, etc.
`RSRC_CONF`	No	No	Yes
`ACCESS_CONF`	No	Yes	No
`RSRC_CONF|ACCESS_CONF`	No	Yes	Yes
`OR_ALL`	Yes	Yes	Yes
`OR_AUTHCFG`	Yes, with `AuthConfig` override	Yes	No
`OR_LIMIT`	Yes, with `Limit` override	Yes	No
`OR_FILEINFO`	Yes, with `FileInfo` override	Yes	Yes
`OR_INDEXES`	Yes, with `Indexes` override	Yes	Yes
`OR_OPTIONS`	Yes, with `Options` override	Yes	Yes

Logging Constants

These constants correspond to the various settings of the `LogLevel` directive. Due to the way they are implemented within mod_perl, they are part of the `Apache::Log` class and not the `Apache::Constants` class. They also do not need to be imported into your handler specifically; a simple `use Apache::Log;` is sufficient to be able to use all these constants within your code. Note that in the current implementation, a `LogLevel` of `debug` has the highest numerical value, and `emerg` the lowest, which is the opposite of what you might expect.

```
use Apache::Log;

$r->server->log->info('LogLevel is info or debug...')
  if $r->server->loglevel >= Apache::Log::INFO;
```

The `LogLevel` constants are

- `Apache::Log::EMERG`
- `Apache::Log::ALERT`
- `Apache::Log::CRIT`
- `Apache::Log::ERR`
- `Apache::Log::WARNING`
- `Apache::Log::NOTICE`
- `Apache::Log::INFO`
- `Apache::Log::DEBUG`

Server Constants

Table B.5 shows the two constants that are available from the `Apache::Constants` class that are really Apache API calls underneath, and which are useful for digging out base server information.

Table B.5 `SERVER_BUILT` *and* `SERVER_VERSION`

Constant Name	Description
`SERVER_BUILT`	Returns the date and time the `httpd` binary was compiled.
`SERVER_VERSION`	Returns the Apache version as specified in the `ServerTokens` directive and/or the `$Apache::Server::AddPerlVersion` global.