Mod Snake API

Jon Travis

jtravis@p00p.org

1. Terminology

The following important terms are used throughout this documentation:

Callback

A `Callable` Python object. This object can take the form of an object method, a function, etc.

Directive

A line in an Apache configuration file (`httpd.conf`, `.htaccess`, etc.)

PyMod

A PyMod is a grouping of function callbacks, callback data, and directive information. In general, this cooresponds to a class within a Python module.

2. ModSnakePyMod API

The ModSnakePyMod API is a specialized API available to callbacks within PyMods, called from mod_snake. Occasionally a ModSnakePyMod object will be passed into a PyMod's callback function. This object has special attributes and methods for dictating how a PyMod will interact with the Apache server.

2.1. add_hook

Synopsis

ModSnakePyMod.add_hook(*hookname*, *hookfunc*)

Validity

Valid within the `__init__` method of the PyMod's object class.

Arguments

- *hookname* - (type String) a hook name from the hook table in Table 1
- *hookfunc* - (type Callable) a function to call when the given hookname phase is reached by the server.

Description

This routine registers hooks that Apache can call at designated time. A ModSnakePyMod is passed into the `__init__` method via mod_snake. The *hookname* can be a variety of strings, each cooresponding to a different hook phase within Apache. The *hookfunc* is the callback function to call whenever the phase cooresponding to *hookname* is reached. In the subsequent sections, when referring to any specific hook (i.e. 'create_dir_config') the reference is to the actual function (hookfunc) associated with it by the PyMod.

Table 1. Mod Snake/Apache Hooks

Hook Name	Apache Equivalent
create_dir_config	**The per-dir creation function within the module record**
merge_dir_config	**The per-dir merge function within the module record**

Hook Name	Apache Equivalent
create_svr_config	The per-server creation function within the module record
merge_svr_config	The per-server merge function within the module record
post_config	ap_hook_post_config
open_logs	ap_hook_open_logs
pre_connection	ap_hook_pre_connection
process_connection	ap_hook_process_connection
post_read_request	ap_hook_post_read_request
translate_name	ap_hook_translate_name
header_parser	ap_hook_header_parser
access_checker	ap_hook_access_checker
check_user_id	ap_hook_check_user_id
auth_checker	ap_hook_auth_checker
fixups	ap_hook_fixups
content-handler	The handler_rec structure within the module
log_transaction	ap_hook_log_transaction
insert_filter	ap_hook_insert_filter

2.1.1. create_dir_config

Synopsis

create_dir_config(`path`)

Arguments

- *path* - (type String or `None`) A path given to the PyMod, designating the association between the callback's return data and the path. This pertains to the directory that the directory config is being created for. For instance, when a `.htaccess` in a particular directory is read.

Return Value

The PyMod may return any data it wishes from this callback. mod_snake will pass the return value from this method to a variety of other callbacks, registered by the PyMod.

Description

Apache will call the create_dir_config callback in the PyMod whenever it is required to generate per-directory configuration data. *path* may be `None`, indicating that the path associated with this call is in a server context, and not necessarily associated with any given directory.

2.1.2. merge_dir_config

Synopsis

merge_dir_config(*per_dir1*, *per_dir2*)

Arguments

- *per_dir1* - The base per-directory PyMod data
- *per_dir2* - The sub-directory per-directory PyMod data

Return Value

The PyMod may return any data it wishes from this callback. If this callback is not performed, it is the same as though this callback returned `per_dir2`. mod_snake will pass the return value from this function to a variety of other callbacks, registered by the PyMod.

Description

Apache occasionally gives modules the chance to combine per-directory configuration information into 1 per-dir configuration. Apache gives the base and new directory configuration information back to the module, and the module must combine them in whatever fashion it sees fit. This can be especially useful for doing fine grained configuration throughout sub-directories.

2.1.3. create_svr_config

Synopsis

create_svr_config(*`svr`*)

Arguments

- *`svr`* - The server for which the configuration is being created for

Return Value

The PyMod may return any data it wishes from this callback. mod_snake will pass the return value from this function to a variety of other callbacks, registered by the Python module.

Description

Apache will call the create_svr_config callback in the module whenever the module is required to generate per-server configuration data.

2.1.4. merge_svr_config

Synopsis

merge_svr_config(*per_svr1*, *per_svr2*)

Arguments

- *per_svr1* - The base per-server module data
- *per_svr2* - The new per-server module data

Return Value

The PyMod may return any data it wishes from this callback. If this callback is not performed, it is the same as though this callback returned *per_svr2*. mod_snake will pass the return value from this function to a variety of other callbacks, registered by the Python module.

Description

Apache occasionally gives modules the chance to combine per-server configuration information into 1 per-server configuration. Apache gives the base and new server configuration information back to the module, and the module must combine them in whatever fashion it sees fit.

2.1.5. post_config

Synopsis

post_config(*svr_cfg*)

Arguments

- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks

Return Value

The result of post_config is not checked, nor used.

Description

This callback is called after Apache has read the configuration file. Keep in mind that Apache does read the configuration file more than one time.

2.1.6. open_logs

Synopsis

open_logs(*svr_cfg*, *module*)

Arguments

- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *module* - (type ModSnakePyMod) a reference to the module's external representation. This can be used to obtain the current server record, etc.

Return Value

The result of open_logs is not checked, nor used.

Description

The open_logs callback will be called when Apache has finished reading the configuration and allows modules to open up logs. Module writers may want to register this hook if they intend to write their own logging routines (instead of using built in mod_snake API calls), as it gives them 'special' permission to open logfiles prior to becoming an ordinary (no-permissions) process. mod_snake passes in a ModSnakePyMod module descriptor, which the module can use at its discretion.

2.1.7. pre_connection

Synopsis

pre_connection(`svr_cfg`, `conn_rec`)

Arguments

- `svr_cfg` - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- `conn_rec` - (type conn_rec) Information about the current connection.

Return Value

The result of pre_connection is not checked, nor used.

Compatability

This hook is only available when mod_snake is running on an Apache 2.0 server.

Description

The pre_connection callback is called when Apache receives an incoming connection on a socket. All pre_connection callbacks are run until one returns a value other than `OK` or `DECLINED`.

2.1.8. process_connection

Synopsis

process_connection(*svr_cfg*, *conn_rec*)

Arguments

- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *conn_rec* - (type conn_rec) Information about the current connection.

Return Value

The result of process_connection is used to determine whether further action must be performed. If the return value is `OK`, the connection is dropped.

Compatability

This hook is only available when mod_snake is running on an Apache 2.0 server.

Description

The process_connection callback is made directly after the pre_connection callback. All process_connection callbacks are run until one returns a value other than `DECLINED`.

2.1.9. post_read_request

Synopsis

post_read_request(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of post_read_request is one of the standard Apache hook result codes.

Descriptions

The post_read_request callback is executed directly after the request has been made and before any URI translation is performed. All post_read_request callbacks are executed in all modules until one returns a value other than `OK` or `DECLINED`.

2.1.10. translate_name

Synopsis

translate_name(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of translate_name is one of the standard Apache hook result codes.

Descriptions

The translate_name callback is executed after the post_read_request callback is performed, and before the header_parsing callback is executed. During this phase a module may perform translations on the requested URI. All translate_name callbacks are executed in all modules until one returns a value other than `DECLINED`.

2.1.11. header_parser

Synopsis

header_parser(*`dir_cfg`*, *`svr_cfg`*, *`request_rec`*)

Arguments

- *`dir_cfg`* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *`svr_cfg`* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *`request_rec`* - (type request_rec) Information about the incoming request.

Return Value

The result of header_parser is one of the standard Apache hook result codes.

Descriptions

The header_parser callback is made when Apache wants to give a chance to the module to validate or deny a given request. The module can inspect the values of request_rec.method to determine an appropriate course of action. All header_parser callbacks are executed in all modules until one returns a value other than `OK` or `DECLINED`.

2.1.12. access_checker

Synopsis

access_checker(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of access_checker is one of the standard Apache hook result codes.

Descriptions

The access_checker callback is called after the header_parser stage, and gives the module an opportunity to allow or deny a request based upon whichever criteria it wishes. mod_access uses the access_checker hook in order to allow or deny requests based upon various criteria such as IP address.

2.1.13. check_user_id

Synopsis

check_user_id(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of check_user_id is one of the standard Apache hook result codes.

Descriptions

The check_user_id callback allows modules to perform user authentication before allowing them to proceed. This can be done in a variety of ways, such as a database, configuration directive, or anything the module wishes. This routine should typically return one of 3 values, HTTP_UNAUTHORIZED, `OK`, and `DECLINED`, but need not be limited by these. All check_user_id handlers are run until one returns a value other than `DECLINED`.

2.1.14. auth_checker

Synopsis

auth_checker(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.

- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of auth_checker is one of the standard Apache hook result codes.

Descriptions

The auth_checker callback allows modules to grant or deny access to data based upon their user data. All auth_checker handlers are run until one returns a value other than `DECLINED`.

2.1.15. type_checker

Synopsis

type_checker(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of type_checker is one of the standard Apache hook result codes.

Descriptions

The type_checker callback is used to determine a MIME type for the requested document. The module can use this callback and use whatever server configuration it desires to associate a type with the requested document. This type is later used to determine which handler will be called for the type. All type_checker handlers are run until one returns a value other than `DECLINED`.

2.1.16. fixups

Synopsis

fixups(*`dir_cfg`*, *`svr_cfg`*, *`request_rec`*)

Arguments

- *`dir_cfg`* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *`svr_cfg`* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *`request_rec`* - (type request_rec) Information about the incoming request.

Return Value

The result of fixups is one of the standard Apache hook result codes.

Descriptions

The fixups callback is used as a last ditch effort by modules to tweak any headers, etc. before the request is handed off to the content-handler. mod_env uses this stage to add various environment variables before passing along the request. All fixups handlers are run until one returns something ore than `OK` or `DECLINED`.

2.1.17. content_handler

Synopsis

content_handler(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of content_handler is one of the standard Apache hook result codes.

Descriptions

The content_handler is used to actually generate content to be returned to the client. This can range in anything from reading files on the system, to querying database entries, to formulating data from the system environment. The content_handler phase in mod_snake is slightly different than that in regular Apache, in that if a module registers with the content_handler phase, all requsets for any content will be passed into this routine. If the module does not wish to handle the request for a specific content type, it can return `DECLINED`. All content_handler handlers are run until one returns something other than `DECLINED`.

2.1.18. log_transaction

Synopsis

log_transaction(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of log_transaction is one of the standard Apache hook result codes.

Descriptions

The log_transaction handler enables modules to do any specialized logging after the request has been processed, and the content been sent to the client. All log_transaction handlers are run until one returns a value other than `OK` or `DECLINED`.

2.1.19. insert_filter

Synopsis

insert_filter(*dir_cfg*, *svr_cfg*, *request_rec*)

Arguments

- *dir_cfg* - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.

- *svr_cfg* - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- *request_rec* - (type request_rec) Information about the incoming request.

Return Value

The result of insert_filter is one of the standard Apache hook result codes.

Compatability

This hook is only available when mod_snake is running on an Apache 2.0 server.

Descriptions

The insert_filter hook is called directly after the fixups callback. Within this hook, the PyMod can add a filter to the filter chain (See Section 5.15). All insert_filter hooks are run, regardless of return value.

2.1.20. Standard Hook Return Values

Hooks may return any integer value that they wish, but there are already a myriad of established return values and associated meanings. The values used internally within Apache (and thus available to modules through the mod_snake API, such as mod_snake.HTTP_UNAUTHORIZED) include:

Table 2. Standard Hook Return Values

DONE
DECLINED
OK
HTTP_CONTINUE
HTTP_SWITCHING_PROTOCOLS

HTTP_OK
HTTP_CREATED
HTTP_ACCEPTED
HTTP_NON_AUTHORITATIVE
HTTP_NO_CONTENT
HTTP_RESET_CONTENT
HTTP_PARTIAL_CONTENT
HTTP_MULTI_STATUS
HTTP_MULTIPLE_CHOICES
HTTP_MOVED_PERMANENTLY
HTTP_MOVED_TEMPORARILY
HTTP_SEE_OTHER
HTTP_NOT_MODIFIED
HTTP_USE_PROXY
HTTP_TEMPORARY_REDIRECT
HTTP_BAD_REQUEST
HTTP_UNAUTHORIZED
HTTP_PAYMENT_REQUIRED
HTTP_FORBIDDEN
HTTP_NOT_FOUND
HTTP_METHOD_NOT_ALLOWED
HTTP_NOT_ACCEPTABLE
HTTP_PROXY_AUTHENTICATION_REQUIRED
HTTP_REQUEST_TIME_OUT
HTTP_CONFLICT
HTTP_DONE
HTTP_LENGTH_REQUIRED

HTTP_PRECONDITION_FAILED
HTTP_REQUEST_ENTITY_TOO_LARGE
HTTP_REQUEST_URI_TOO_LARGE
HTTP_UNSUPPORTED_MEDIA_TYPE
HTTP_RANGE_NOT_SATISFIABLE
HTTP_EXPECTATION_FAILED
HTTP_UNPROCESSABLE_ENTITY
HTTP_LOCKED
HTTP_FAILED_DEPENDENCY
HTTP_INTERNAL_SERVER_ERROR
HTTP_NOT_IMPLEMENTED
HTTP_BAD_GATEWAY
HTTP_SERVICE_UNAVAILABLE
HTTP_GATEWAY_TIME_OUT
HTTP_VERSION_NOT_SUPPORTED
HTTP_VARIANT_ALSO_VARIES
HTTP_INSUFFICIENT_STORAGE
HTTP_NOT_EXTENDED

Note that DONE has the special meaning that the module has served the response completely, and no other hooks should be called, `OK` has the special meaning that the handler has handled a given phase, and `DECLINED` means that the handler did not process the stage, and that Apache should continue to look for a handler to process it.

2.2. add_directives

Synopsis

ModSnakePyMod.add_directives(*`directive_dict`*)

Validity

Valid within the __init__ method of the module's object class.

Arguments

- *`directive_dict`* - A dictionary specifying the name, location, type, and callbacks for Python created configuration directives. The dictionary has the form:

  `{ "DirectiveName" : (directive_location, functype, func_ptr), ... }`

Description

Python written modules running under mod_snake have the same ability as C written modules when it comes to creating custom configuration directives. The directive_dict contains a dictionary of directive names, and associated attributes.

- *`DirectiveName`* - specifies the textual name within the Apache configuration file to trigger the modules directive handler (func_ptr). This value is case insensitive.
- *`directive_location`* - Puts a limit on the locations where this directive is valid. The supported values may also be binary ORed togther to combine multiple types of locations. The current possible values are:
  - *`mod_snake.OR_LIMIT`* - The directive is valid in the main configuration file inside <directory> and <Location> blocks, and also within .htaccess files when AllowOverride Limit is set.
  - *`mod_snake.OR_OPTIONS`* - The directive is valid anywhere in the main configuration file, and also within .htaccess files when AllowOverride Options is

set.

  - *mod_snake.OR_FILEINFO* - The directive is valid anywhere in the main configuration file, and also within .htaccess files when AllowOverride FileInfo is set.
  - *mod_snake.OR_AUTHCFG* - The directive is valid in the main configuration file inside <directory> and <Location> blocks, and also within .htaccess files when AllowOverride AuthConfig is set.
  - *mod_snake.OR_INDEXES* - The directive is valid anywhere in the main configuration file, and also within .htaccess files when AllowOverride Indexes is set.
  - *mod_snake.ACCESS_CONF* - The directive is valid in the main configuration file within <directory> and <Location> blocks.
  - *mod_snake.RSRC_CONF* - The directive is valid in the main configuration file, outside of <directory> and <Location> blocks.
  - *mod_snake.OR_ALL* - The directive is valid anywhere.

- *functype* - Designates the format that the arguments to the directive callback function take. This alleviates some of the hassle of having each module parse all of their directives manually. Note that the callback functions associated with the functype can potentially have many different declarations (especially when doing 1 or etc.).

  - *mod_snake.RAW_ARGS* - The arguments to the directive are passed in, unparsed.

    func(*self*, *dir_data*, *argline*)

  - *mod_snake.TAKE1* - The directive takes only one argument

    func(*self*, *dir_data*, *arg1*)

  - *mod_snake.TAKE2* - The directive takes exactly two arguments

    func(*self*, *dir_data*, *arg1*, *arg2*)

  - *mod_snake.ITERATE* - The directive can take multiple arguments, but the associated callback function will be executed once for each argument.

    func(*self*, *dir_data*, *arg1*)

  - *mod_snake.ITERATE2* - The directive can take at least 2 arguments, but the second argument is iterated over, for > 2 arguments.

    func(*self*, *dir_data*, *arg1*, *arg2*)

  - *mod_snake.FLAG* - The directive takes 1 argument, 'on' or 'off'. Within the module this argument is represented as true or false.

    func(*self*, *dir_data*, *on_or_off*)

  - *mod_snake.NO_ARGS* - The directive takes no arguments

    func(*self*, *dir_data*)

  - *mod_snake.TAKE12* - The directive can take either 1 or 2 arguments

    func(*self*, *dir_data*, *arg1*, **arg2*)

  - *mod_snake.TAKE3* - The directive takes exactly 3 arguments

    func(*self*, *dir_data*, *arg1*, *arg2*, *arg3*)

  - *mod_snake.TAKE23* - The directive takes either 2 or 3 arguments

    func(*self*, *dir_data*, *arg1*, **arg23*)

  - *mod_snake.TAKE123* - The directive takes either 1, 2 or 3 arguments

    func(*self*, *dir_data*, **arg123*)

  - *mod_snake.TAKE13* - The directive takes either 1 or 3 arguments

    func(*self*, *dir_data*, **arg1or3*)

- *func_ptr* - A pointer to a function to call when the directive is processed within the configuration file. The function prototype must match that expected by the caller (as given by the functype). In addition, the first argument passed to func_ptr is that of a per-directory data structure, as returned by the modules create_dir_config callback, if it exists.

2.3. profile_obj

Synopsis

This attribute of the ModSnakePyMod object holds the value of the profiling object. If profiling is not enabled, this value will be None. The profiling object is an instantiation of profile.Profile class.

2.4. module

This attribute of the ModSnakePyMod object holds the value of the Python module containing the loaded pymod. For instance, if the pymod loaded into the server via the SnakeModule line was: SnakeModule spam.eggs, the module attribute will contain a reference to the spam module.

2.5. add_version_component

Synopsis

ModSnakePyMod.add_version_component(vers)

Description

The HTTP header, "Server: ", returned by Apache on requests can be set to include additional information for modules loaded. This method can be used to add module data into the server string which all clients see. The string should be of the form, "module/version", such as "mod_snake/1.2.3". This routine should be called in the module's initialization phase.

2.6. get_hooks

Synopsis

ModSnakePyMod.get_hooks()

Return Value

This method returns a dictionary of { hookname : funcref } pairs.

Description

The get_hooks method looks directly within the loaded PyMod to determine which hooks it has hooked into (via add_hook). The returned value is a dictionary containing the hook name and a reference to the function which gets called for that hook.

2.7. register_output_filter

Synopsis

ModSnakePyMod.register_output_filter(*`name`*, *`callback`*, *`type`*)

Arguments

- *`name`* - The name of the filter. This string is left to the PyMod's discretion. It distinguishes it from other filters, and allows it to be added from external means, such as `httpd.conf`
- *`callback`* - The function which should be called when it is given its turn to process outgoing data.
- *`type`* - One of `mod_snake.FTYPE_CONTENT`, `mod_snake.FTYPE_HTTP_HEADER`, `mod_snake.FTYPE_TRANSCODE`, `mod_snake.FTYPE_CONNECTION`, or `mod_snake.FTYPE_NETWORK` indicating when the filter should be run. Content filters are for altering content. Transcode is a special type of filter type for doing transport based encoding (chunking). Connection

based filters are filters which modify content, but are done on a more connection oriented basis. Network are the lowest level filter possible – they read/write data directly to the remote client. The aforementioned filter types are sorted in execution order, with NETWORK being the first run on input filters, and last run on output filters.

Compatability

This function is only available when mod_snake is running on an Apache 2.0 server.

Description

By using output filters, PyMods have the ability to process data written by content generators before it is sent to the remote client. This routine only needs to be called once in each module for each filter. It provides Apache with information needed to associate the textual name of the filter with the callback function.

2.7.1. output_filter

Synopsis

output_filter(`dir_cfg`, `svr_cfg`, `filter`, `brigade`)

Arguments

- `dir_cfg` - Directory configuration, as returned by calls to the create_dir_config and merge_dir_config hooks.
- `svr_cfg` - Server configuration, as returned by calls to the create_svr_config and merge_svr_config hooks.
- `filter` - Information about the current running filter. This object contains many attributes. See Section 8.
- `brigade` - A bucket brigade of data to be filtered. See Section 9

Return Value

The return value should be 0 on success, -1 to indicate failure.

Description

`output_filter` is only a prototype for the `callback` to register_output_filter. In order for this callback to ever be invoked, it must be added to the filter chain via add_output_filter.

3. mod_snake Misc API

The mod_snake Misc API is available as methods within the mod_snake module. These methods don't really fit in anywhere else.. ;-)

3.1. get_version

Synopsis

mod_snake.get_version()

Return Value

This method returns version information about mod_snake, in the same fashion as the Apache server-string. The result is of the form: "mod_snake/version", where version is generally something like "0.1.0", etc.

3.2. get_modules

Synopsis

mod_snake.get_modules()

Return Value

This method returns a list of all the modules loaded into the server. The list contains objects of type ModSnakePyMod

4. mod_snake Apache API

The mod_snake Apache API is available as methods within the mod_snake module. The mod_snake module can be imported into any Python module, and the routines accessed directly. The Apache API contains routines which should be accessable by regular modules, CGIs, and embedded Python.

4.1. ap_server_root_relative

Synopsis

mod_snake.ap_server_root_relative(*`path`*)

Arguments

- *`path`* - (type string) A path (relative or absolute) to convert, relative to the server's root directory.

Return Value

The return value is a new string, containing the absolute path on the filesystem. For example, if the server's root directory was /home/jtravis/apache, then ap_server_root_relative('logs/mylog') would return /home/jtravis/apache/logs/mylog.

4.2. ap_log_error

Synopsis

mod_snake.ap_log_error(`level`, `server`, `err_string`)

Arguments

- `level` - A level, indicating the severity of the error. Possible values are: APLOG_EMERG, APLOG_ALERT, APLOG_CRIT, APLOG_ERR, APLOG_WARNING, APLOG_NOTICE, APLOG_INFO, APLOG_DEBUG, all of which are available as attributes within the mod_snake module.
- `server` - A server_rec object where the error occurred, or None
- `err_string` - A string (with no terminating '\n') description of the error to be logged.

Return Value

No values are returned from this function.

4.3. ap_getwords_conf

Synopsis

mod_snake.ap_getwords_conf(`str`)

Arguments

- `str` - a string to parse as arguments to a configuration directive.

Return Value

A list containing elements of the split-up str is returned.

Description

Apache uses the ap_getword_conf routine to obtain a new argument to a configuration directive. Arguments may be quoted strings or plain strings. The ap_getwords_conf routine returns a list of arguments as parsed from the str argument.

4.4. ap_get_server_version

Synopsis

mod_snake.ap_get_server_version()

Return Value

Returns a string containing the Apache server version

Description

The server version as returned by the function is the same that is sent to clients via the server string. The common form of the server-version is “Apache/version”. e.g.: “Apache/2.0a3”

4.5. ap_validate_password

Synopsis

mod_snake.ap_validate_password(pword, hashed)

Return Value

Returns a true value of the hashed argument is a hashed representation of the password, else false.

Description

This routine compares a plaintext password with a hashed password (which is often created with the htpasswd program, for instance). Hashed passwords are often stored in a database or file as opposed to their plaintext counterparts. This provides safety against theft of the password file.

5. request_rec API

The request record is what Apache passes to most of the phase handlers.

5.1. send_http_header

Synopsis

request_rec.send_http_header()

Description

Before content data is sent to the client, headers must be sent. Modules which choose to handle the entire content_handler phase by themselves, must call this routine prior to sending content. The headers sent are setup within the request_rec's headers_in and headers_out dictionaries. Other fields such as content_type and status are also used when send_http_header() generates its response.

5.2. add_cgi_vars

Synopsis

request_rec.add_cgi_vars()

Description

This routine is equivalent to the Apache ap_add_cgi_vars(), which adds commonly used CGI environment variables to the request's subprocess_env dictionary.

5.3. add_common_vars

Synopsis

request_rec.add_common_vars()

Description

This routine is equivalent to the Apache ap_add_common_vars(), which adds common CGI environment variables to the request's subprocess_env dictionary.

5.4. get_basic_auth_pw

Synopsis

request_rec.get_basic_auth_pw()

Description

This routine returns a tuple (rescode, pword), where rescode is one of mod_snake.HTTP_INTERNAL_SERVER_ERROR, mod_snake.HTTP_UNAUTHORIZED, mod_snake.OK, or mod_snake.DECLINED. If rescode is mod_snake.OK, pword will be a string password passed by the client, else it will be None.

5.5. requires

Synopsis

request_rec.requires()

Return Value

This routine returns a tuple containing a number of require_line objects. Each require_line object contains requirement and method_mask attributes.

Description

When Apache processes ’Require’ directives, the values are added to an internal array. PyMods may access this array via the requires() function. The objects returned contain a requirement attribute, designating the value passed to the ’Require’ directive, and a method_mask indicating the limit imposed on the requirement as designated by the <Limit> directives.

5.6. note_auth_failure

Synopsis

request_rec.note_auth_failure()

Description

This routine is a wrapper around the request_rec.note_basic_auth_failure(), and request_rec.note_digest_auth_failure() functions. Either of them, or none may be called depending on the authentication type of the request.

5.7. note_basic_failure

Synopsis

request_rec.note_basic_auth_failure()

Description

This routine makes modifications to the outgoing client error headers. It will add the WWW-Authenticate header, or the Proxy-Authenticate header, depending on the type of request.

5.8. note_digest_failure

Synopsis

request_rec.note_digest_auth_failure()

Description

This routine makes modifications to the outgoing client error headers. It will add the WWW-Authenticate header, or the Proxy-Authenticate header, depending on the type of request.

5.9. send_http_trace

Synopsis

request_rec.send_http_trace()

Return Value

This routine returns mod_snake.OK on success, and a mod_snake.HTTP_* error on failure.

Description

This routine sends a copy of the clients request back to the client, including header and body information.

5.10. ap_rwrite

Synopsis

request_rec.rwrite(*str*)

Arguments

- `str` - data to write to the remote client.

Return Value

Returns an integer designating how many bytes were written.

Description

The rwrite function is one of the main ways that PyMods can communicate information back to the remote client. This routine sends all data passed in through all of Apache's internal routines for buffering and filtering. Under Apache 2.0, this method is still available, but buffers data internally before converting it to a bucket in a brigade.

> **Warning**
>
> Modules should never use both rwrite calls and bucket brigades to write to a client. Since rwrite calls are buffered internally, data may be sent out of order if they are mixed.

5.11. setup_client_block

Synopsis

request_rec.setup_client_block(*`policy`*)

Arguments

- *`policy`* - One of the pre-defined policies

Return Value

The return value is one of the standard hook return values in Table 2.

Description

The read policy argument is one of:

Table 3. Read Policies

Argument	Description
REQUEST_NO_BODY	**Send 413 error if message has any body**
REQUEST_CHUNKED_ERROR	**Send 411 error if body without Content-Length**
REQUEST_CHUNKED_DECHUNK	**If chunked, remove the chunks for the caller**

This routine sets up the policy for how subsequent reads via should_client_block, and get_client_block are performed.

5.12. should_client_block

Synopsis

request_rec.should_client_block()

Return Value

This method will return 0 if there is no data to be read, otherwise it will return 1.

Description

This method is an intermediate step between the setup of the client block, and getting the client blocks. It ensures that 100 Continue responses are sent appropriately. This function should never be called more than once per request.

5.13. get_client_block

Synopsis

request_rec.get_client_block(`len`)

Arguments

- `len` - Number of bytes to read.

Return Value

This method returns a tuple containing a (data, datalen) pair – the data read, and the length of the data. On error, 'data' will be Py_None.

Description

This method gets client sent information. It takes care of chunking, as per the setup_client_block.

5.14. discard_request_body

Synopsis

request_rec.discard_request_body()

Return Value

Returns one of the standard hook return values in Table 2.

Description

In the case where a handler does not want to deal with a request body, this routine can be called to safely discard it. This is especially useful, as to not disrupt persistant connections.

5.15. add_output_filter

Synopsis

request_rec.add_output_filter(`filtername`, `callback_data`)

Arguments

- `filtername` - Name of filter to add.
- `callback_data` - Callback data to send to the filter function.

Compatability

This method is only available when mod_snake is running on an Apache 2.0 server.

Description

This method is used to add a filter to the filter chain. The filter must first be registered via register_output_filter (see Section 2.7). The callback data is available within the filter function as filter.ctx.

5.16. brigade_create

Synopsis

request_rec.brigade_create()

Compatability

This method is only available when mod_snake is running on an Apache 2.0 server.

Description

This method creates a new brigade (see Section 9).

5.17. output_filters

Synopsis

request_rec.output_filters

Compatability

This hook is only available when mod_snake is running on an Apache 2.0 server.

Description

This attribute contains a reference to the chain of filters that are scheduled to process data. Content generation phases can call the pass_brigade method of this object to send data to the client.

5.18. connection

Synopsis

request_rec.connection

Description

This attribute contains a reference to the connection associated with the incoming request.

5.19. server

Synopsis

request_rec.server

Description

This attribute contains a reference to the server that the incoming request came in on.

5.20. next

Synopsis

request_rec.next

Description

If the request ends up getting redirected, this attribute points to the request that it was directed to.

5.21. prev

Synopsis

request_rec.prev

Description

If the current request is an internal redirect, this attribute points to the request it came from.

5.22. main

Synopsis

request_rec.main

Description

If this request is a sub request, this attribute points back to the main request.

5.23. the_request

Synopsis

request_rec.the_request

Description

This attribute contains the first line of the request. Example: 'GET /foo/bar.html HTTP/1.0'

5.24. header_only

Synopsis

request_rec.header_only

Description

This attribute is a true/false indicating whether or not it is a HEAD request

5.25. protocol

Synopsis

request_rec.protocol

Description

This attribute is a string containing the protocol sent by the client, otherwise HTTP/0.9

5.26. proto_num

Synopsis

request_rec.proto_num

Description

This attribute is an integer representing the protocol number, as sent by the client. Example: HTTP/0.9 == 9, HTTP/1.1 == 1001

5.27. hostname

Synopsis

request_rec.hostname

Description

This attribute contains the hostname, as set by the Host: header, or the full URI. If not given, it is 'None'

5.28. status_line

Synopsis

request_rec.status_line

Description

This string attribute may be set by the module to return a specialized status-line back to the client.

5.29. status

Synopsis

request_rec.status

Description

This integer attribute may be set byt he module to return a specific status number back to the client.

5.30. method

Synopsis

request_rec.method

Description

This string attribute contains the method as set by the client. Example: GET, HEAD, FOO, etc.

5.31. method_number

Synopsis

request_rec.method_number

Description

This attribute holds an integer value representing the method as sent by the client. The following values are the ones recognized by the server, and are properties of the mod_snake module: M_GET, M_PUT, M_POST, M_DELETE, M_CONNECT, M_OPTIONS, M_TRACE, M_PATCH, M_PROPFIND, M_PROPPATCH,

M_MKCOL, M_COPY, M_MOVE, M_LOCK, M_UNLOCK, M_INVALID. Note that M_GET is valid for both GET and HEAD methods.

5.32. headers_in

Synopsis

request_rec.headers_in

Description

This dictionary attribute contains the MIME headers as specified by the client.

5.33. headers_out

Synopsis

request_rec.headers_out

Description

This dictionary attribute contains the MIME headers that will be sent out to the client when the send_http_header method is called.

5.34. notes

Synopsis

request_rec.notes

Description

This dictionary attribute provides storage for modules on a *per request* basis. Modules can store any data they wish in this dictionary, however it will be unavailable for subsequent requests. This storage can be useful when a module needs to keep per-request information in between different handlers (such as the fixups and content-handlers.)

5.35. content_type

Synopsis

request_rec.content_type

Description

This string attribute contains the content type as established by the Apache type-checker phase, or None before the type-checker phase is called. Example: "text/html"

5.36. content_encoding

Synopsis

request_rec.content_encoding

Description

This string attribute contains the encoding of the content.

5.37. handler

Synopsis

request_rec.handler

Description

This string attribute contains the handler to be dispatched on for the content-handler, else None.

5.38. user

Synopsis

request_rec.user

Description

This string attribute contains the user name if authentication was made, else None.

5.39. ap_auth_type

Synopsis

request_rec.ap_auth_type

Description

This string attribute contains the authentication type if authentication was made, else None. Example: 'basic', 'digest', etc.

5.40. unparsed_uri

Synopsis

request_rec.unparsed_uri

Description

This string attribute contains the unparsed URI.

5.41. uri

Synopsis

request_rec.uri

Description

This string attribute contains the path portion of the URI.

5.42. filename

Synopsis

request_rec.filename

Description

This string attribute contains the filename portion of the requested URI.

5.43. path_info

Synopsis

request_rec.path_info

Description

This string attribute contains the path information of the requested URI, with no appended filename. If there is no path, this attribute is 'None'

5.44. args

Synopsis

request_rec.args

Description

This string attribute contains the query arguments from the URI if given, else None. This string can be passed to cgi.py's parse_qs() function to turn it into a dictionary of values.

cgi.py: cgi.py is distributed with all Python distributions.

5.45. finfo

Synopsis

request_rec.finfo

Description

If the requested document is a real file, this attribute contains a tuple from the stat(). The tuple is the same as returned by the os.stat function, and contains the following elements: (mode, inode, device, number_links, uid, guid, size, atime, mtime, ctime)

6. server_rec API

The server records that Apache uses contain a ton of useful information. While modules do not generally use all of these attributes, some of them are fairly useful and important.

6.1. next

Synopsis

server_rec.next

Description

Within the Apache webserver, many virtual-hosts can be run simultaneously. The servers are stored in a linked list within the server, and the next attribute returns the next server in the list.

6.2. def_name

Synopsis

server_rec.def_name

Description

This string attribute contains the definition name of the server, as from the configuration file. In the case of the main server, this value is None

6.3. defn_line_number

Synopsis

server_rec.defn_line_number

Description

This integer attribute contains the line number of the VirtualHost directive for which the server was created.

6.4. server_admin

Synopsis

server_rec.server_admin

Description

This attribute contains the server admin string, as specified within the configuration file by the ServerAdmin directive.

6.5. server_hostname

Synopsis

server_rec.server_hostname

Description

This attribute contains the server hostname string, as specified within the configuration file by the ServerName directive.

6.6. port

Synopsis

server_rec.port

Description

This attribute contains the port number that the server is running on.

6.7. is_virtual

Synopsis

server_rec.is_virtual

Description

Attribute specifying whether or not a given server is a virtual server or not (true or false)

6.8. path

Synopsis

server_rec.path

Description

Attribute specifying the pathname, as given by the ServerPath directive within the configuration file.

6.9. names

Synopsis

server_rec.names

Description

Returns a tuple containing the names as denoted by ServerAlias

6.10. wild_names

Synopsis

server_rec.wild_names

Description

Returns a tuple containing the wildcarded names as denoted by ServerAlias.

6.11. server_uid

Synopsis

server_rec.server_uid

Description

Returns the effective user id when calling the exec wrapper

6.12. server_gid

Synopsis

server_rec.server_gid

Description

Returns the effective group id when calling the exec wrapper.

7. conn_rec API

Every incoming request to Apache has an associated conn_rec object which contains information about the connection. This object can also be used to directly read and write to the remote client.

7.1. base_server

Synopsis

conn_rec.base_server

Description

This attribute is a reference to a server_rec object on which the connection came in.

7.2. local_addr

Synopsis

conn_rec.local_addr

Description

This string attribute contains the local address that the request came in on. Example: 127.0.0.1

7.3. remote_addr

Synopsis

conn_rec.remote_addr

Description

This string attribute contains the remote address of the incoming connection. Example: 69.0.0.1

7.4. write

Synopsis

conn_rec.write(*the_skinny*)

Arguments

- *the_skinny* - string data to write to the remote client

Return Value

Returns a tuple (status, nbytes), where status is one of the APR error codes in the mod_snake module, and nbytes are the number of bytes successfully written.

Description

Occasionally a module writer may wish to write directly to the remote connection, such as when using a protocol other than HTTP. This routine allows the module to communicate directly with the remote client.

7.5. read

Synopsis

conn_rec.read(*nbytes*)

Arguments

- *nbytes* - The number of bytes to read

Return Value

Returns a tuple (status, nbytes, data) where status is one of the APR error codes in the mod_snake module, nbytes is the number of bytes read, and data is the actual data read from the client.

Description

This method allows modules to read data from the remote client.

7.6. flush

Synopsis

conn_rec.flush

Description

This method flushes the buffered output, as sent from conn_rec.write()

7.7. remote_ip

Synopsis

conn_rec.remote_ip

Description

This string attribute contains the remote IP address of the incoming connection. Example: 127.0.0.1

7.8. keepalive

Synopsis

conn_rec.keepalive

Description

This integer attribute indicates whether or not HTTP keep-alive is in effect for the current connection. -1 indiciates a fatal error, 0 undecided, and 1 yes.

7.9. keptalive

Synopsis

conn_rec.keptalive

Description

This true/false attribute indicates whether or not HTTP keep-alive was used during the request.

7.10. double_reverse

Synopsis

conn_rec.double_reverse

Description

This integer attribute indicates whether or not double-reverse DNS has been performed. -1 = failure, 0 = not yet, 1 = yes

7.11. keepalives

Synopsis

conn_rec.keepalives

Description

This integer attribute indicates how many times this connection has been used for requests.

7.12. local_ip

Synopsis

conn_rec.local_ip

Description

This string attribute contains the IP of the server that the connection came in on.

7.13. local_host

Synopsis

conn_rec.local_host

Description

This string attribute contains the hostname of the server that the connection came in on.

7.14. id

Synopsis

conn_rec.id

Description

This integer attribute contains a unique ID at any given time.

8. filter API

Under Apache 2.0, modules have the ability to filter content before sending it to the remote client. This incorporates several different APIs. First, the module must register the filter via the register_output_filter method (see Section 2.7). Secondly, the PyMod must add the hook to the filter chain somewhere within the request phase. For output filters, this is commonly done at either post read request time, or in the insert_filters hook. Lastly, when the filter is called, it must utilitize both the filter API, and the bucket brigade API to manipulate content before sending it to the client or next filter.

8.1. name

Synopsis

filter.name

Description

This string attribute contains the name of the filter, as specified when calling register_output_filter. (See Section 2.7)

8.2. pass_brigade

Synopsis

filter.pass_brigade(`brigade`)

Arguments

- `brigade` - The bucket brigade to pass down the filter chain.

Description

When a filter is to be given a new chunk of data to process, whether from a content generator, or another filter, this routine can be called to start the processing. Subsequent filters should pass the data down the chain until the data is either written to the client, discarded, or buffered.

8.3. next

Synopsis

filter.next

Description

The `next` attribute points to the next filter in the filter chain. This is commonly used from within a filter, to pass the brigade on to the next filter.

8.4. ctx

Synopsis

filter.ctx

Description

The `ctx` parameter contains callback data, as given by the PyMod, when the filter was added. (See Section 5.15)

8.5. r

Synopsis

filter.r

Description

The current request_rec object that the filter is processing.

9. brigade API

Under Apache 2.0, modules can generate, filter, and discard content. This content comes in the form of bucket brigades. A bucket brigade is simply a list of several 'buckets.' Each bucket contains data. The reason that the data is dispersed into these packets is that a module may make a specific optimization with respect to that data - Such as, that it may be mmap()ed, or static data. The brigade API allows modules to manipulate the brigades, and the data stored within. Brigades last the lifetime of the object that creates them. Buckets may or may not have a different lifetime. Brigade creation for request output filters is done in the request_rec (See Section 5.16).

9.1. get_head

Synopsis

brigade.get_head()

Description

This routine returns the first bucket in the bucket brigade. If the brigade is empty, this returns `None`.

9.2. get_tail

Synopsis

brigade.get_tail()

Description

This routine returns the last bucket in the bucket brigade. If the brigade is empty, this returns `None`.

9.3. get_next

Synopsis

brigade.get_next(`bucket`)

Arguments

- `bucket` - A bucket to get the next one after.

Description

This routine returns the next bucket after the passed in bucket argument. If there is no bucket after the argument, the return value is `None`

9.4. get_prev

Synopsis

brigade.get_prev(`bucket`)

Arguments

- `bucket` - A bucket to get the previous one of.

Description

This routine returns the previous bucket before the passed in bucket argument. If there is no bucket before the argument, the return value is `None`

9.5. move_head

Synopsis

brigade.move_head(*bucket*)

Arguments

- *bucket* - A bucket to move to the head of the brigade.

Description

This routine can either move a bucket from a totally seperate brigade into the head of the called brigade, or move a bucket currently in the brigade to the head.

9.6. move_tail

Synopsis

brigade.move_tail(*bucket*)

Arguments

- *bucket* - A bucket to move to the tail of the brigade.

Description

This routine can either move a bucket from a totally seperate brigade into the tail of the called brigade, or move a bucket currently in the brigade to the tail.

9.7. move_before

Synopsis

brigade.move_before(*before*, *bucket*)

Arguments

- *before* - The sentinel bucket. The moved bucket will be moved before this one.
- *bucket* - The bucket to move.

Description

This routine moves the *bucket* to a position prior to '*before*'. This method can also move buckets between different brigades, if *bucket*, and *before* are in different brigades.

9.8. move_after

Synopsis

brigade.move_after(*after*, *bucket*)

Arguments

- *before* - The sentinel bucket. The moved bucket will be moved after this one.
- *bucket* - The bucket to move.

Description

This routine moves the *bucket* to a position after '*after*'. This method can also move buckets between different brigades, if *bucket*, and *after* are in different brigades.

9.9. remove

Synopsis

brigade.move_after(*bucket*)

Arguments

- *bucket* - The bucket to remove.

Description

This routine removes a bucket from the brigade. The removed bucket is no longer accessable from the brigade, but may continue to exist internally, if other brigades are accessing it.

9.10. bucket_create_simple

Synopsis

brigade.bucket_create_simple(*data*)

Arguments

- *data* - The data value of the new bucket.

Description

This is the main routine for creation of new buckets. The *data* passed in will be available to other filters or other modules accessing the brigade. In Apache 2.0, this is one of the primary methods of generating data for the client.

9.11. bucket_create_eos

Synopsis

brigade.bucket_create_eos()

Description

This routine creates a special `eos` bucket. This bucket indicates that it is the end of the stream. This triggers subsequent filters to assume that no more data will be sent after the `eos` bucket is sent.

10. bucket API

One of the fundamental blocks for moving data to the client comes in the form of the bucket brigade (See Section 9). The brigade is simply an ordered list of buckets.

10.1. is_eos

Synopsis

bucket.is_eos()

Description

This integer attribute indicates whether or not the bucket is an `EOS` (end of stream) bucket. If *`is_eos`* is true, the bucket is `EOS`, otherwise it is a regular bucket.

10.2. name

Synopsis

bucket.name()

Description

This string attribute gives the internally defined name of the bucket type.

10.3. length

Synopsis

bucket.length()

Description

This integer attribute contains the length of the data stored in the bucket. It can equal `-1`, indicating that the length of the data cannot be determined until it is read. Values other than `-1` indicate the length of the data that will be read. If *`length`* is `-1`, bucket data must be read until `None` is returned.

10.4. read

Synopsis

bucket.read(*`block`*)

Arguments

- *`block`* - Integer value indicating whether or not this routine should block when trying to read data.

Description

This is the main routine for transferring data from the bucket to the caller. If the *`block`* parameter is given, this routine will block until the data has been read. If the bucket's length is `-1` this routine will return `None` when there is no more data to be read.