extensions

Release 1.0

Mark Anacker

Jan 09, 2024

CONTENTS

CHAPTER

ONE

EXTENSIONS PACKAGE

1.1 Submodules

1.1.1 extensions.fileext module

fileext - file handling extension.

This extension provides some simple text file handling. It limits the file locations to a pre-set root directory.

Make these functions available to a script by adding:

> loadExtension_('fileext')

to it. Functions exported by this extension:

readLines_()
: read a text file line-by-line or all-at-once

writeLines_()
: write a line (or lines) to a text file, creating the file if need be. *

appendLines_()
: write a line (or lines) to the end of a text file, extending it. *

listFiles_()
: return a list of files in the given path *

- may be disabled by option settings

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class FileExt(*api*, *options={}*)
: Bases: `object`

  This class manages commands to read and write files.

__init__ (*api*, *options={}*)
Constructs an instance of the FileExt class.

This instance will manage all named queues. There will be only one of these instances at a time.

Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine
- **options** – a dict of option settings passed down to the extension

Options

'file_root' - a path prepended to all filenames, restricting access to files below this point.

'read_only' = If True, the writeLines_ and appendLines_ functions are not installed.

'list_files' - If True, allows the **list_files_** call

appendLines_ (*filepath*, *data*, *handler=None*, *maxlines=0*)
Write lines of data to a text file, appending to any previous file

Append lines to a text file all-at-once, or line by line.

Args:

filepath [filename (and optional path) to read from.] Absolute (/) and parent (..) paths are not allowed. Path will be under "file_root" if configured. Required.

data [A string to write to the file as a line of text,] or a list[] of strings that will be written as multiple lines. Required.

handler [The name of a script function that wll be] called to obtain a line of text to be written to the file. If the handler returns False, file writing is stopped. Optional.

maxlines [The number of lines to write before stopping. If] omitted or 0, write until the data is exhausted. Optional.

Returns:

None : If there was an error writing to the file

count [int The number of lines sent to the] handler, or the number of bytes sent if no handler.

listFiles_ (*filepath=''*)
Return a list of files on a path.

Returns a list of files in the specified directory, or the configured basepath if not specified. The path is always relative to basepath.

Args:

filepath : The directory to list

Returns:

A list[] with the contents of the directory.

readLines_ (*filepath*, *handler=None*, *maxlines=0*)
Read data from a file in text mode.

Read a text file all-at-once, or line by line.

Args:

filepath [filename (and optional path) to read from.] Absolute (/) and parent (..) paths are not allowed. Path will be under "file_root" if configured. Required.

handler [The name of a script function that wll be] called with every line of text. The function will be called with a single str parameter containing the text line. If the handler returns False, file reading is stopped. Optional.

maxlines [The number of lines to read before stopping. If] omitted or 0, read until the end of the file. Optional.

Returns:

None : If there was an error opening the file

data [str The lines from the file if no handler] was specified

count : int The number of lines sent to the handler

register()

Make this extension's functions available to scripts

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- readLines_() : read a text file line-by-line or all-at-once
- **writeLines_()** [write a line (or lines) to a text file,] creating the file if need be.
- **appendLines_()** [write a line (or lines) to the end of a text] file, extending it.
- listFiles_() : return a list of files in the given path

Parameters None

Returns

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

shutdown()

Perform a graceful shutdown

unregister()

Remove this extension's commands

writeLines_(*filepath*, *data*, *handler=None*, *maxlines=0*)

Write lines of data to a text file, overwriting any previous file

Write to a text file all-at-once, or line by line.

Args:

filepath [filename (and optional path) to read from.] Absolute (/) and parent (..) paths are not allowed. Path will be under "file_root" if configured. Required.

data [A string to write to the file as a line of text,] or a list[] of strings that will be written as multiple lines. Required.

handler [The name of a script function that wll be] called to obtain a line of text to be written to the file. If the handler returns False, file writing is stopped. Optional.

maxlines [The number of lines to write before stopping. If] omitted or 0, write until the data is exhausted. Optional.

Returns:

None : If there was an error opening the file

count [int The number of lines sent to the] handler, or the number of bytes sent if no handler.

lineWriter(*api*, *fp*, *data*, *handler=None*, *maxlines=0*, *mode='w'*)

Write lines to a file in either overwite or append mode

Parameters

- **fp** – Filepath to write to
- **data** – Data to write if no handler
- **handler**
- **maxlines** – Max number of lines to write
- **mode** – "w" - overwrite. "a" - append

Returns:

typeWrite(*f*, *data*)

1.1.2 extensions.flagext module

flagext - Named multi-thread Event flags

This extension provides a means for threads to synchronize with each other. It allows the script to create named "flags" that script elements running in different threads (such as handlers) can manipulate. A flag may be raised or lowered, and threads can wait for a flag to be raised. Flags may also be checked for state without waiting.

If multiple threads are waiting on a flag, they will all be released when it's raised.

This is really a wrapper around a set of Threading.Event() objects. The "flag" metaphore is easier to visualize, and the Extension handles all of the fussy bits of managing a bunch of Event objects.

Make the functions available to a script by adding:

loadExtension_('flagext')

to it. Functions exported by this extension:

flag_add_()

Create a named flag (state == lowered)

flag_del_()

Delete a named flag and release any waits

flag_raise_()

Raise the flag and release any flag_wait_()s

flag_lower_()

Reset the flag to lowered state

flag_israised_()

Check the flag state

flag_wait_()
Wait for a flag to be raised, or timeout

flag_list_()
Return a list[] of flag names

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class FlagExt(*api*, *options={}*)
Bases: object

This class manages commands to set and clear sync flags.

__init__(*api*, *options={}*)
Constructs an instance of the FlagExt class.

This instance will manage all named flags. There will be only one of these instances at a time.

Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine
- **options** – a dict of option settings passed down to the extension

flag_add_(*name*)
Handles the flag_add_() function.

Creates a new flag and adds it's name to the table.

Args:

name : The flag name to create. Must not be in use.

Returns:

True if the flag was created.

False if an error occurred.

Options:

None

flag_del_(*name*)
Handles the flag_del_() function.

Momentarily raises the flag (to release any flag_wait_() listeners), then deletes the flag from the table.

Args:

name : The name of the flag to delete.

Returns:

The return value. True for success, False otherwise.

flag_israised_(*name*)
Handles the flag_israised_() function.

Checks the state of the flag without having to wait()

Args:

name : The name of the flag to check.

Returns:

The flag state. True if raised, False if lowered

flag_list_()
Handles the flag_list_() function.

Returns a list with the names of all named flags.

Args:

None

Returns:

A list[] of flag names. The list may be empty if there are no flags.

None if there was an error.

flag_lower_(*name*)
Handles the flag_lower_() function.

Lowers the flag, priming it for flag_wait_() listeners.

Args:

name : The name of the flag to lower.

Returns:

The return value. True for success, False otherwise.

flag_raise_(*name*, *toggle=False*)
Handles the flag_raise_() function.

Raises the flag (releasing any flag_wait_() listeners).

Args:

name : The name of the flag to raise.

toggle : If True, lower the flag after a tiny pause

Returns:

The return value. True for success, False otherwise.

flag_wait_(*name*, *timeout=1.0*, ***kwargs*)
Handles the flag_wait_() function.

Waits for a flag to be raised, or a timeout to expire.

Args:

name : The name of the flag to wait on.

timeout : The time to wait in seconds

**kwargs : A dict with supported options.

Returns:

The return value. True for success, False on error or timeout.

Options:

prelower : If True, lower the flag BEFORE waiting

postlower : If True, lower the flag AFTER waiting

register()
Make this extension's functions available to scripts

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- flag_add_() : Create a named flag (state == lowered)
- flag_del_() : Delete a named flag and release any waits
- flag_raise_() : Raise the flag and release any flag_wait_()s
- flag_lower_() : Reset the flag to lowered state
- flag_israised_() : Check the flag state
- flag_wait_() : Wait for a flag to be raised, or timeout
- flag_list_() : Return a list[] of flag names

Args:

None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

---

shutdown()
Perform a graceful shutdown.

This gets called by the extension manager just before the extension is unloaded.

unregister()
Remove this extension's functions from the engine.

1.1.3 extensions.httpext module

httpext - HTTP(S) client extension

This extension provides HTTP(s) client functions to the scripts. It can perform GET and PUT requests, using one of two external modules (requests or http.client). If you are using the requests mode (1), you can also issue any HTTP request type.

Make these functions available to a script by adding:

loadExtension_('httpext')

to it. Functions exported by this extension:

http_modes_()
Return a list of available client modes

http_get_()
Do a GET to a URL

http_put_()
Do a PUT to a URL

http_requests_()
Calls requests() directly (if available)

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class HttpExt(*api*, *options={}*)
Bases: object

This class provides an HTTP/HTTPS client.

__init__(*api*, *options={}*)
This instance contains the HTTP client functions. Some flags are set based on the results of the imports above.

Args:

api : an instance of ExtensionAPI connecting us to the engine.

options : a dict of option settings passed down to the extension.

Returns:

Nothing.

http_clientModes_()
Returns the list of available modes.

Returns a tuple consisting of the list[] of available client modes, and the default mode selected by the extension. There are currently two client modes: 1 uses the requests module, and 2 uses the http.client module. This extension tries to use mode 1, but will use mode 2 if the requests module fails to import.

Each request may override the default and choose which mode it wants, if that mode is available.

Args:

None

Returns:

A tuple: (modelist, default)

Example:

([1,2], 1)

http_get_(*url*, ***kwargs*)
Performs a GET from a URL.

Sends an HTTP or HTTPS GET to the specified URL, using one of the client modes. The mode is either the default set in __init__() above, or specified by the 'client=' option.

Args:

url : The URL to send the GET to.

**kwargs : A dict with supported options.

Returns:

If simple is True, return just the data from the body of the response. In the case of an error, an empty string will be returned.

If simple is False, return a tuple with the HTTP status code, the contents of the response, and a dict with detailed response data. The entries in this dict will vary depending on the client mode.

Options:

client : The client mode (1 or 2) to use.

headers : A dict of HTTP headers to use.

simple : Returns only the retrieved text.

timeout : Request timeout in seconds.

cert : TLS certificate - either a String with just the client cert, or a tuple with the cert and keyfile (cert, keyfile).

verify : Verify the server hostname.

Other options are passed to the requests or http.client modules.

http_put_(*url*, ***kwargs*)
Performs a PUT to a URL.

Sends an HTTP or HTTPS PUT to the specified URL, using one of the client modes. The mode is either the default set in __init__() above, or specified by the 'client=' option.

The data to send in the body of the PUT is set by the 'data=' option.

Args:

url : The URL to send the GET to.

**kwargs : A dict with supported options.

Returns:

Returns a tuple with the HTTP status code, the contents of the response, and a dict with detailed response data. The entries in this dict will vary depending on the client mode.

Options:

client : The client mode (1 or 2) to use.

data : The data to send to the server

headers : A dict of HTTP headers to use.

timeout : Request timeout in seconds.

cert : TLS certificate - either a String with just the client cert, or a tuple with the cert and keyfile (cert, keyfile).

verify : Verify the server hostname.

Other options are passed to the requests or http.client modules.

http_request_ (*method*, *url*, ***kwargs*)
Sends a request to a URL.

This is a thin wrapper over the requests method of the requests module. As such, it's only available if the requests import succeeded.

Args:

method : The method to use (String). Must be: GET, OPTIONS, HEAD, POST, PUT, PATCH, or DELETE.

url : The URL to use (String).

**kwargs : A dict with supported options.

Returns:

Returns a tuple with the HTTP status code, the contents of the response, and a dict with detailed response data.

Options:

Options are passed to the requests module. See: https://requests.readthedocs.io/en/latest/

register ()
Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- http_modes_() : Returns the list of available modes
- http_get_() : Performs a GET to a URL
- http_put_() : Performs a PUT to a URL
- http_request_() : Calls requests() directly (if available)

Args:

None

Returns

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

---

shutdown ()
Perform a graceful shutdown

unregister ()
Remove this extension's commands

debug(**args*)

1.1.4 extensions.mqttext module

mqttext -mqtt client commands

This extension implements a client for the MQTT pub/sub protocol. It can handle multiple connections to MQTT brokers, publish messages to topics, and subscribe to topics. Incoming messages on subscribed topics may be delivered by polling, or by callbacks.

The link to a broker is represented by a connection name. Each connection is separate from the others, and each may have multiple topics subscribed to it.

Make the functions available to a script by adding:

loadExtension_('mqttext')

to it. Functions exported by this extension:

mqtt_connect_()
Create a named connection to a broker

mqtt_disconnect_()
Close a named connection

mqtt_list_()
List all currently-active connections

mqtt_subscribe_()
Subscribe a connection to a topic

mqtt_unsubscribe_()
Remove a subscription from a connection

mqtt_isrunning_()
True is the connection is attached to a broker

mqtt_waiting_()
The number of messages waiting to be read

mqtt_readmsg_()
Return the first availabel message on the connection

mqtt_sendmsg_()
Send a message to a given topic

Note: Required Python modules:

paho.mqtt.client

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class MqttConnection(*name*, *api*)
Bases: object

This class represents a connection to a broker. There can be several connections active simultaneously, linked to different brokers. Or all to the same broker, but that would be inefficient.

connect_(***kwargs*)

connectclient()

disconnect_()

isrunning_()

on_connect(*client*, *userdata*, *flags*, *rc*)

on_disconnect(*client*, *userdata*, *rc*)

on_message(*client*, *userdata*, *message*)

on_topic_message(*client*, *userdata*, *message*)

on_wildcard_message(*client*, *userdata*, *message*)

readmsg_(*blocking=True*, *timeout=1*)

resuball()

sendmsg_(*dest*, *data*, *qos=0*, *retain=False*)

subtopic_(*topic*, *handler=None*, *qos=0*)

unsubtopic_(*topic*)

waiting_()

class MqttExt(*api*, *options={}*)
Bases: object

This class manages commands to send/receive mqtt messages

__init__(*api*, *options={}*)
Constructs an instance of the MqttExt class.

This instance will manage all connections to mqtt brokers. There will be only once of these instances at a time.

Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine.
- **options** – a dict of option settings passed down to the extension.

Returns None

__api
An instance of ExtensionAPI passed by the host, used to call back into the engine. Copied from api.

__options
A dict of options from the host that may or may not apply to this extension. Copied from options.

__cmddict
Dispatch table of our script command names and their functions.

__conns
The table of active connections, indexed by name.

__locktimeout
Timeout in seconds to wait for a mutex.

__lock
Thread-locking mutex.

connect_(*cname*, ***kwargs*)
Handles the mqtt_connect_() function.

Create a connection object and establish a connection to a broker.

Args:

cname : The name of the connection

**kwargs : A dict with all of the required parameters

Returns:

The return value. True for success, False otherwise.

disconnect_(*cname*)
Handles the mqtt_disconnect_() function.

Closes an open connection to a broker, and removes the connection from the table.

Args:

cname: The name of the connection to remove

Returns:

The return value. True for success, False otherwise.

isrunning_(*cname*)

listConns_()
Return a list[] of open connections.

Args:

None

Returns:

A list[] of active connections. The list may be empty if there are no connections.

None if there was an error.

readmsg_(*cname*, *blocking=True*, *timeout=1*)

register()
Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- mqtt_connect_() : Create a named connection to a broker
- mqtt_disconnect_() : Close a named connection
- mqtt_list_() : List all currently-active connections
- mqtt_subscribe_() : Subscribe a connection to a topic
- mqtt_unsubscribe_() : Remove a subscription from a connection
- mqtt_isrunning_() : True is the connection is attached to a broker
- mqtt_waiting_() : The number of messages waiting to be read
- mqtt_readmsg_() : Return the first availabel message on the connection
- mqtt_sendmsg_() : Send a message to a given topic

Args: None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

sendmsg_(*cname*, *dest*, *data*, *qos=0*, *retain=False*)

shutdown()
Perform a graceful shutdown.

Close all of the active MQTT connections. This gets called by the extension manager just before the extension is unloaded.

subtopic_(*cname*, *topic*, *handler=None*, *qos=0*)

unregister()
Remove this extension's functions from the engine.

unsubtopic_(*cname*, *topic*)

waiting_(*cname*)

debug(**args*)

topicmatch(*subsc*, *topic*)

1.1.5 extensions.queueext module

queueext - Named multi-thread queues.

This extension provides thread-safe queues for communicating between threads (and callback handlers). It has all the queue management functions anyone should need.

Queues are referenced by name (set in the queue_open_() function), and may be either a classic First-In-First-Out queue, or a Last-In-First-Out stack. These queues are an excellent way to pass data from an event-driven handler function to a main processing loop.

Make the functions available to a script by adding:

loadExtension_('queueext')

to it. Functions exported by this extension:

queue_open_()
Create a named queue

queue_close_()
Flush and destroy an existing queue

queue_put_()
Put data into a queue

queue_get_()
Get the next item from a queue

queue_clear_()
Clear all items from a queue

queue_len_()
Return the number of items in a queue

queue_isempty_()
Return True if the queue is empty

queue_list_()
Return a list[] of the current queue names

Credits

- version: 1.0.1
- last update: 2024-Jan-08
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class QueueExt(*api*, *options={}*)
Bases: object

This class provides thread-safe queues.

__init__(*api*, *options={}*)
Constructs an instance of the QueueExt class.

This instance will manage all named queues. There will be only one of these instances at a time.

Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine
- **options** – a dict of option settings passed down to the extension

queue_clear_(*name*)
Handles the queue_clear_() function.

Removes all items in a queue.

Args:

name : The name of the queue to get from.

Returns:

The return value. True for success, False otherwise.

queue_close_(*name*)
Handles the queue_close_() function.

Close an open queue and clear any data currently in it.

Args:

name : The name of the queue to close.

Returns:

The return value. True for success, False otherwise.

queue_get_(*name*, ***kwargs*)
Handles the queue_get_() function.

Gets an item from the queue. A 'fifo' queue returns the item at the HEAD of the queue. A 'lifo' type will return the item at the END of the queue.

The get operation may be either blocking or non-blocking. If the block option in kwargs is False, the operation will return immediately. If the queue was empty, the return will be None.

If block is True (default) and the queue is empty, the get operation will be retried at 1-second intervals until either: an item is added to the queue, the timeout count is reached, or the queue is closed. A timeout of 0 will retry until either success or close.

Args:

name : The name of the queue to get from.

**kwargs : Options to pass down to the Queue.

Options:

block : If True, block until timeout if the queue is empty.

timeout : Seconds to wait when blocking. 0 means wait forever.

Returns:

The next value from the queue, or None.

queue_isempty_(*name*)
Handles the queue_isempty_() function.

Returns True if the queue is empty.

Args:

name : The name of the queue to check.

Returns:

True if the named queue is empty, or there was an error.

queue_len_(*name*)
Handles the queue_len_() function.

Returns the number of entries in a queue.

Args:

name : The name of the queue to check.

Returns:

The number of items in the named queue. Any error returns 0.

queue_list_()
Handles the queue_list_() function.

Returns a list with the names of all open queues.

Args:

None

Returns:

A list[] of open queues. The list may be empty if there are no queues open.

None if there was an error.

queue_open_(*name*, ***kwargs*)
Handles the queue_open_() function.

Creates a new queue and adds it's name to the table.

Args:

cname : The queue name to use. Must not be in use.

**kwargs : Options to pass down to the Queue.

Returns:

True if the queue was created.

False if an error occurred.

Options:

type : 'fifo' or 'lifo' - queue or stack

queue_put_(*name*, *value*, ***kwargs*)
Handles the queue_put_() function.

Adds an item to the queue. Items are always added to the back of the queue.

The put operation may be either blocking or non-blocking. If the block option in kwargs is False, the item is simply sent to the queue. If the queue is full, the item will be silently discarded. If block is True (default), the put operation will be retried at 1-second intervals until either: the item is added to the queue, the timeout count is reached, or the queue is closed. A timeout of 0 will retry until either success or close.

Args:

name : The name of the queue to add to.

value : The data item to add to the queue

**kwargs : Options to pass down to the Queue.

Options:

block : If True, block until timeout if the queue is full.

timeout : Seconds to wait when blocking. 0 means wait forever.

Returns:

The return value. True for success, False otherwise.

register()
Make this extension's functions available to scripts

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- queue_open_() : Create a named queue
- queue_close_() : Flush and destroy an existing queue
- queue_put_() : Put data on a qurur
- queue_get_() : Get the next item from a queue
- queue_clear_() : Clear all items from a queue
- queue_len_() : Return the number of items in a queue
- queue_isempty_() : Return True if the queue is empty
- queue_list_() : Return a list[] of the current queue names

Args:

None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

---

shutdown()
Perform a graceful shutdown.

This gets called by the extension manager just before the extension is unloaded.

unregister()
Remove this extension's functions from the engine.

class ThreadQueue_(*name*, *api*, *qtype*)
Bases: `object`

This class implements the actual thread-safe queue.

There is one of these for every named queue in the table.

tqueue_clear_()

tqueue_close_()

tqueue_get_(**kwargs*)

tqueue_isempty_()

tqueue_len_()

tqueue_put_(*value*, ***kwargs*)

clearq__(*q*)

1.1.6 extensions.redisext module

redisext - basic interface to the Redis cache.

This extension provides a client connection to a redis in-memory cache. Not all redis functions are available (there are a *lot* of them), but a lot of the useful ones are included.

Familiarity with redis is helpful. See: https://redis.io/commands/

Make the functions available to a script by adding:

> loadExtension_('redisext')

to it. Functions exported by this extension:

redis_connect_()
: Connect to a redis server

redis_disconnect_()
: Disconnect from the server

redis_list_()
: Return a list[] of active connections

redis_cmd_()
: Pass a command string to redis *

  these map directly to redis_cli commands:

redis_set_()
: Set a value in the cache

redis_get_()
: Get a value from the cache

redis_del_()
: Remove a keyed entry from the cache

redis_incr_()
: Increments the number stored at key by one

redis_decr_()
: Decrements the number stored at key by one

redis_hset_()
: Sets the value associated in a hash stored at key.

redis_hmset_()
: Sets multiple values in a hash

redis_hget_()
: Returns the value associated in a hash stored at key.

redis_hdel_()
: Removes the specified fields from the hash stored at key

redis_hkeys_()
: Returns all field names in the hash stored at key.

redis_hvals_()
: Returns all values in the hash stored at key.

- may be disabled by option settings

Options

allow_redis_cmds If True, installs the "redis_cmd_" function to pass complete redis command strings to the server. This is a potential security risk.

Note: Required Python modules:

redis

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class RedisConnection(*name*, *api*)
Bases: object

This class represents a connection to a Redis server. There can be several connections active simultaneously, open to different servers (or possibly the same server).

cmds_(**args*)
Send a command directly to the server.

connect_(*host='127.0.0.1'*, *port=6379*, ***kwargs*)
Connect to the Redis server.

disconnect_()
Disconnect from the Redis server.

hmset_(**args*)
Load a whole dict into a redis hashmap.

class RedisExt(*api*, *options={}*)
Bases: object

This class manages connections to a Redis cache server.

__init__(*api*, *options={}*)
Constructs an instance of the RedisExt class.

This instance will manage all connections to one or mode redis servers. There will be only once of these instances at a time.

Args:

api : an instance of ExtensionAPI connecting us to the engine.

options : a dict of option settings passed down to the extension.

Returns:

None

Attributes:

__api [An instance of ExtensionAPI passed by the host, used] to call back into the engine. Copied from api.

__options [A dict of options from the host that may or may not] apply to this extension. Copied from options.

__cmddict [Dispatch table of our script command names and their] functions.

__conns : The table of active connections, indexed by name.

__cmdsflag [If True, install the "redis_cmd_()" function. Set] in the options dict passed in.

__locktimeout : Timeout in seconds to wait for a mutex.

__lock : Thread-locking mutex.

cmd_(*cname*, **args*)
Handles the redis_cmd_() function.

Sends a raw command directly to the redis server. This is potentially dangerous, and is only available if specifically enabled by the host application via the options.

Args:

cname: The name of the connection to use.

*args: The arguments to pass to the execute_command function.

Returns:

The results of the command.

None if there was an error

connect_(*cname*, *host='127.0.0.1'*, *port=6379*, ***kwargs*)
Handles the redis_connect_() function.

This function establishes a named connection to a redis server. Successful completion is required before any other functions may be used (except **redis_list_**).

Args:

cname: The name of the connection

host: The hostname or ip address of the server

port: The port to connect to the server on

**kwargs: Optional arguments

Returns:

The return value. True for success, False otherwise.

decr_(*cname*, *key*, *val=1*)
Handles the redis_decr_() function.

del_(*cname*, *key*)
Handles the redis_del_() function.

disconnect_(*cname*)
Handles the redis_disconnect_() function.

Closes an open connection to a redis server and removes the connection from the table.

Args:

cname: The name of the connection to remove

Returns:

The return value. True for success, False otherwise.

get_(*cname*, *key*)
Handles the redis_get_() function.

hdel_(*cname*, *hashname*, *key*)
Handles the redis_hdel_() function.

hget_(*cname*, *hashname*, *key*)
Handles the redis_hget_() function.

hkeys_(*cname*, *hashname*)
Handles the redis_hkeys_() function.

hmset_(*cname*, *hashname*, *map*)
Handles the redis_hmset_() function.

hset_(*cname*, *hashname*, *key*, *value=''*)
Handles the redis_hset_() function.

hvals_(*cname*, *hashname*)
Handles the redis_hvals_() function.

incr_(*cname*, *key*, *val=1*)
Handles the redis_incr_() function.

list_()
Handles the redis_list_() function.

This method returns a list[] of active Redis connections.

Args:

None

Returns:

A list[] of active connections. The list may be empty if there are no connections.

None if there was an error.

register()
Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- redis_connect_() : Connect to a redis server
- redis_disconnect_() : Disconnect from the server
- redis_list_() : Return a list[] of active connections
- redis_cmd_() : Pass a command string to redis [OPT]

  these map directly to redis_cli commands:
- redis_set_() : Set a value in the cache

- redis_get_() : Get a value from the cache
- redis_del_() : Remove a keyed entry from the cache
- redis_incr_() : Increments the number stored at key by one
- redis_decr_() : Decrements the number stored at key by one
- redis_hset_() : Sets the value associated in a hash stored at key.
- redis_hmset_() : Sets multiple values in a hash
- redis_hget_() : Returns the value associated in a hash stored at key.
- redis_hdel_() : Removes the specified fields from the hash stored at key
- redis_hkeys_() : Returns all field names in the hash stored at key.
- redis_hvals_() : Returns all values in the hash stored at key.

[OPT] may be disabled by option settings

Args:

None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

---

set_(*cname*, *key*, *value*)
Handles the redis_set_() function.

shutdown()
Perform a graceful shutdown.

Close all of the active Redis connections. This gets called by the extension manager just before the extension is unloaded.

unregister()
Remove this extension's functions from the engine.

1.1.7 extensions.sqliteext module

sqliteext - SQlite3 database functions.

This extension allows a script to store and retrieve data from a sqlite3 database. The database files are stored under a directory passed in via the "sql_root" option.

Some basic knowledge of SqLite3 and SQL is helpful when using this extension.

All functions are mutex-protected, and can be used from multiple threads.

Make the functions available to a script by adding:

loadExtension_('sqliteext')

to it. Functions exported by this extension:

sql_open_()
Open a sqlite3 database, creating it if it doesn't exist.

sql_close_()
Close an open connection.

sql_execute_()
Execute a SQL command in the open database.

sql_commit_()
Commit any pending transactions (in autocommit == False)

sql_rollback_()
Roll back any pending transactions

sql_changes_()
Return the number of rows recently altered

sql_cursor_fetch_()
Return rows from a cursor after an execute_()

sql_list_()
Return a list[] of the active connection names

Note: Required Python modules:

sqlite3

Credits

- version: 1.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class SqLiteExt(*api*, *options={}*)
Bases: object

This class manages commands to use a sqlite3 database.

__init__(*api*, *options={}*)
Constructs an instance of the SqLiteExt class.

This instance will manage all connections to sqlite3 databases. There will be only one of these instances at a time.

Args:

api : an instance of ExtensionAPI connecting us to the engine.

options : a dict of option settings passed down to the extension.

Returns:

Nothing.

__api
An instance of ExtensionAPI passed by the host, used to call back into the engine. Copied from api.

__options
A dict of options from the host that may or may not apply to this extension. Copied from options.

__cmddict
Dispatch table of our script command names and their functions.

__conns
The table of active connections, indexed by name.

__cmdsflag
If True, install the "redis_cmd_()" function. Set in the options dict passed in.

sqlroot
The path to prepend to database file names.

sqlext
The file extension to append to database file names.

__locktimeout
Timeout in seconds to wait for a mutex.

__lock
Thread-locking mutex.

Options

'sql_root' [a path prepended to all database names,] restricting access to db files below this point.

'sql_ext' [filename extension to use for database files.] Defaults to '.db'

register()
Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- sql_open_() : Open a sqlite3 database, creating it if it doesn't exist.
- sql_close_() : Close an open connection.
- sql_execute_() : Execute a SQL command in the open database.
- sql_commit_() : Commit any pending transactions (if autocommit is False)
- sql_rollback_() : Roll back any pending transactions
- sql_changes_() : Return the number of rows recently altered
- sql_cursor_fetch_() : Return rows from a cursor after an execute_()
- sql_list_() : Return a list[] of the active connection names

---

Parameters None

Returns

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

shutdown()
Perform a graceful shutdown.

Close all of the active database connections. This gets called by the extension manager just before the extension is unloaded.

sql_changes_(*cname*)
Handles the sql_cursor_fetch_() function.

Returns the number of changes to the database since it was last opened.

Args:

cname : The name of the connection to use.

Returns:

The number of updates in this session.

-1 if there was an error.

sql_close_(*cname*)
Handles the sql_close_() function.

Close an open db connection and remove the connection from the table.

Args:

cname : The name of the connection to remove.

Returns:

The return value. True for success, False otherwise.

sql_commit_(*cname*)
Handles the sql_commit_() function.

Commit pending updates to the database. If autocommit is True, this function has no effect.

Args:

cname : The name of the connection to commit.

Returns:

True for success, False otherwise.

sql_cursor_fetch_(*cname*, *cursor*, *count=0*)
Handles the sql_cursor_fetch_() function.

Return one or many rows from the supplied cursor (returned by sql_execute_()).

Args:

cname : The name of the connection to use.

cursor : The cursor reference returned by sql_execute_().

count : The number of rows to return.

- -1 = get the next row as a tuple.
- 0 = get all the rows as a list of tuples.
- >0 = gets all rows <n> at a time as a list of tuples.

Returns:

The requested number of rows, None if an error occurred.

sql_execute_(*cname*, *sql*, **args*, ***kwargs*)
Handles the sql_execute_() function.

Execute a SQL statement in an open db.

Args:

cname : The connection name to use.

sql : The SQL statement to execute.

*args : Values to substitute in the above SQL (if any).

**kwargs : Options to pass down to sqlite3.

Returns:

A cursor reference if successful.

None if there was an error.

sql_list_()
Handles the sql_list_() function.

Returns a list with the names of all current connections.

Args:

None

Returns:

A list[] of active connections. The list may be empty if there are no connections.

None if there was an error.

sql_open_(*cname*, *dbname*, ***kwargs*)
Handles the sql_open_() function.

Open a connection to a sqllite3 database file, creating the files if it doesn't exist. The connection name and the database filename do not have to be the same.

A check is made to make sure the database filename isn't already in use by another connection. It's probably not a good idea to have the same file accessed by multiple connections simultaneously. That's not the same as calling this extension from multiple threads (which is supported).

Args:

cname : The connection name to use. Must not be in use.

dbname : The Sqlite3 database file to use.

**kwargs : Options to pass down to sqlite3.

Returns:

True if the database was opened.

False if an error occurred.

Options:

autocommit : If True, write the changes to the file after every change. Default=True.

check_same_thread : If True, restrict callers to a single thread. Default=False.

sql_rollback_(*cname*)
Handles the sql_rollback_() function.

Rollback pending changes.

If autocommit is False, updates to the database are held in a buffer until sql_commit_() is called. This function clears that buffer, effectively reverting changes back to the last sql_commit_(). If autocommit is True, this function has no effect.

Args:

cname : The name of the connection to use.

Returns:

The return value. True for success, False otherwise.

unregister()
Remove this extension's functions from the engine.

class SqlLiteConnection(*name*, *dbname*, *api*)
Bases: object

This class represents a connection to a sqlite3 database. There can be several connections active simultaneously, open to different database files.

__init__(*name*, *dbname*, *api*)
Create a SqlLiteConnection object.

Sets up the info for a single connection to a db file.

Args: name : The name of this connection.

dbname : The base name of the database file.

api : A reference back to the engine API. Used for error messages.

Returns: None

sql_changes_()
Return the number of recent updates.

Returns the number of changes to the database since it was last opened.

Args: None

Returns: The number of updates in this session. -1 if there was an error.

sql_close_()
Close the connection to database and clean-up.

sql_commit_()
Handles the sql_commit_() function.

Commit pending updates to the database. If autocommit is True, this function has no effect.

Args: None

Returns: The return value. True for success, False otherwise.

sql_cursor_fetch_(*cursor*, *count=0*)
Get rows from a cursor (returned by sql_execute_()).

Return one or many rows from the supplied cursor.

Args: cursor : The cursor reference returned by sql_execute_(). count : The number of rows to return.

- -1 = get the next row as a tuple.
- 0 = get all the rows as a list of tuples.
- >0 = gets all rows <n> at a time as a list of tuples.

Returns: The requested number of rows. None if an error occurred.

sql_execute_(*sql*, **args*, ***kwargs*)
Execute a sql statement.

Submit a SQL statement to the database.

Args: sql : The SQL statement to execute.

*args : Values to substitute in the above SQL (if any).

**kwargs : Options to pass down to sqlite3.

Returns: A cursor reference if successful.

None if there was an error.

Options: args - values to replace ? in the sql statement.

autocommit=True - Commits changes to the database after the execute_() call. Default=False.

sql_get_dbname_()
Return this connection's db file name .

sql_open_(*dbpath*, ***kwargs*)
Open a database for use.

This method tries to connect to a Sqlite3 database file.

Args: dbpath : The full pathname of the Sqlite3 db file.

**kwargs : Options to pass down to sqlite3.

Returns: True if the database was opened.

False if there was an error.

Options:

- autocommit=True : Commits changes to the database after sql_execute_() call. Default=False
- check_same_thread=False : Allow updates from multiple threads.

sql_rollback_()
Rollback pending changes.

If autocommit is False, updates to the database are held in a buffer until sql_commit_() is called. This function clears that buffer, effectively reverting changes back to the last sql_commit_(). If autocommit is True, this function has no effect.

Args: None

Returns: The return value. True for success, False otherwise.

1.1.8 extensions.tasksext module

tasksext - Thread-safe task runner.

This extension manages script functions running in multiple threads.

Make the functions available to a script by adding:

    loadExtension_('tasksext')

to it. Functions exported by this extension:

tasks_open_()
create a named task

tasks_close_()
delete an existing task

tasks_start_()
Starts a task running in a thread

tasks_stop_()
Stop a running task

tasks_status_()
Return state of a task

tasks_list_()
Return list[] of tasks

Credits

- version: 1.0.1
- last update: 2024-Jan-08
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class TasksExt(*api*, *options={}*)
Bases: object

This class provides scheduled tasks.

__init__(*api*, *options={}*)
Constructs an instance of the TasksExt class.

This instance will manage all thread tasks. There will be only one of these instances at a time.

Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine
- **options** – a dict of option settings passed down to the extension

register()
Make this extension's commands available to scripts

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- tasks_open_() : create a named task
- tasks_close_() : delete an existing task
- tasks_start_() : Starts a task running in a thread
- tasks_stop_() : Stop a running task
- tasks_status_() : Return state of a task
- tasks_list_() : Return list[] of tasks

Args:

None

Returns:

True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

shutdown()
Perform a graceful shutdown.

This gets called by the extension manager just before the extension is unloaded.

tasks_close_(*tname*)
Handles the tasks_close_() function..

Stops an existing task (if running), and releases it's resources.

Args:

tname : The name of the task to close.

Returns:

True if the task was closed.

False if an error occurred.

tasks_cmd(*cmd*, *tname*, *flag=None*)

tasks_list_()
Handles the tasks_list_() function.

Returns a list with the names of all open tasks.

Args:

None

Returns:

A list[] of open tasks. The list may be empty if there are no tasks open.

None if there was an error.

tasks_open_(*tname*, *handler*, *data=None*, *delay=1.0*)
Handles the tasks_open_() function..

Creates a new task thread and adds it's name to the table. The task does not start running until tasks_start_() is called.

Args:

tname : The name to assign to this task.

handler : The name of the script function to run.

data : A parameter to pass to the handler.

delay : The delay between calls to handler. Pass 0 to call it once only. Value is in seconds.

Returns:

True if the task was created.

False if an error occurred.

tasks_pause_(*tname*, *flag*)

Handles the tasks_pause_() function.

Pauses or resumes a running task. Resuming a paused task causes it to execute immediately, then start a new delay period.

Args:

tname : The name of the task to pause/resume.

flag : if True, pause the task. If False, resume it.

Returns:

True if the task was paused/resumed.

False if an error occurred.

tasks_start_(*tname*)

Handles the tasks_start_() function..

Begins calling the handler specified in tasks_open_(). If the delay value was 0, the handler will be called only once. If > 0, the handler will be called repeatedly with 'delay' seconds between iterations.

Args:

tname : The name of the task to start.

Returns:

True if the task was started.

False if an error occurred.

tasks_status_(*tname*)

Handles the tasks_status_() function.

Returns True if the specified task is running.

Args:

tname : The name of the task to check.

Returns:

True if the task is running.

False if the task is stopped, or an error occurred.

tasks_stop_(*tname*)
Handles the tasks_stop_() function.

Stops a running task. Do not reuse this task after stopping it.

Args:

tname : The name of the task to stop.

Returns:

True if the task was stopped.

False if an error occurred.

unregister()
Remove this extension's functions from the engine.

class ThreadTask_(*name*, *api*, *handler*, *data=None*, *delay=1*)
Bases: `threading.Thread`

close_()

is_running_()

pause_(*flag*)

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

stop_()

1.1.9 extensions.tdictext module

tdictext - Thread-safe Dict

This extension provides thread-safe, named dictionaries (key,value). Python's built-in dict objects are *mostly* atomic (and therefore thread-safe), but not *always*. And not every operation is guaranteed to be safe. Scripts in apyshell may make extensive use of threads, so an assured safe dict is needed.

Each named tdict has it's own lock, so they are relatively independant once created.

See also the companion tlist extension.

Make these functions available to a script by adding:

loadExtension_('tdictext')

to it. Functions exported by this extension:

tdict_open_()
create a named dict

tdict_close_()
delete an existing dict

tdict_put_()
add an item to the named dict

tdict_update_()
add a dict to the named dict

tdict_get_()
get an item by key

tdict_pop_()
get an item by key, then remove it

tdict_del_()
remove an item from the dict by key

tdict_clear_()
remove all items from the dict

tdict_keys_()
return a list[] of keys

tdict_items_()
return a list[] of items

tdict_len_()
return the number of items in the dict

tdict_copy_()
return a shallow copy of the dict

tdict_list_()
return a list[] of tdicts

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class TDictExt(*api*, *options={}*)
Bases: object

This class provides a thread-safe key-value store.

__init__(*api*, *options={}*)
Constructs an instance of the TDictExt class.

This instance will manage all thread-safe dictionaries. There will be only one of these instances at a time.

Args:

api : an instance of ExtensionAPI connecting us to the engine.

options : a dict of option settings passed down to the extension.

Returns:

Nothing.

register()
Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- tdict_open_() : create a named dict
- tdict_close_() : delete an existing dict
- tdict_put_() : add an item to the named dict
- tdict_update_() : add a dict to the named dict
- tdict_get_() : get an item by key
- tdict_pop_() : get an item by key, then remove it
- tdict_del_() : remove an item from the dict by key
- tdict_clear_() : remove all items from the dict
- tdict_keys_() : return a list[] of keys
- tdict_items_() : return a list[] of items
- tdict_len_() : return the number of items in the dict
- tdict_copy_() : return a shallow copy of the dict
- tdict_list_() : return a list[] of tdicts

Parameters None

Returns True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

shutdown()
Perform a graceful shutdown.

Clear then close all of the active dictionaries. This gets called by the extension manager just before the extension is unloaded.

tdict_clear_(*cname*)
Handles the tdict_clear_() function.

Removes all items from the named dict.

Args: cname : The name of the dict to use.

Returns: True for success, False otherwise.

tdict_close_(*cname*)
Handles the tdict_close_() function.

Clears out the data from an existing tdict, then removes it.

Args: cname : The name of the dict to remove.

Returns: True for success, False otherwise.

tdict_cmd(*cmd*, *cname*, *key=None*, *value=None*)
Internal command dispatcher.

Performs the operation specified by the "cmd" argument on the dict object associated with "cname", and returns the result.

Args: cmd : The operation to perform

cname : The name of the dict to use.

key : A key string (if needed)

value : A value argument (if needed)

Returns: Depends on the operation. See the functions below for the specific returns.

tdict_copy_(*cname*)
Handles the tdict_copy_() function.

Returns a standard Python dict object with a shallow copy of the contents of the specified tdict.

Args: cname : The name of the dict to use.

Returns: A Python dict with the contents of this tdict.

None if there was an error.

tdict_del_(*cname*, *key*)
Handles the tdict_del_() function.

Removes an item indexed by the key from the dict.

Args: cname : The name of the dict to use.

key : A key string to remove.

Returns: True for success, False otherwise.

tdict_get_(*cname*, *key*, *value=None*)
Handles the tdict_get_() function.

Retrieves an item indexed by the key from the dict, or returns a default value if the key isn't found.

Args: cname : The name of the dict to use.

key : A key string to look for.

value : A default to return if key isn't in the dict.

Returns: The retrieved item, or the supplied value.

tdict_items_(*cname*)
Handles the tdict_items_() function.

Returns a list of every item in the dict, in the form of a tuple with (key,value) for each..

Args: cname : The name of the dict to use.

Returns: A list with items in the dict, or an empty list if there are none.

tdict_keys_(*cname*)
Handles the tdict_keys_() function.

Returns a list with the keys in the named dict.

Args: cname : The name of the dict to use.

Returns: A list[] of keys in the dict. The list may be empty if there are none.

None if there was an error.

tdict_len_(*cname*)
Handles the tdict_len_() function.

Returns the number of items currently in the tdict.

Args: cname : The name of the dict to use.

Returns: An int with the number of items.

tdict_list_()
Handles the tdict_list_() function.

Returns a list with the names of all current dictionaries.

Args: None

Returns: A list[] of active dictionaries. The list may be empty if there are none.

None if there was an error.

tdict_open_(*cname*)
Handles the tdict_open_() function.

Creates a new tdict object attached to the given name.

Args: cname : The name of the dict to create.

Returns: True for success, False otherwise.

tdict_pop_(*cname*, *key*, *value*)
Handles the tdict_pop_() function.

Retrieves an item indexed by the key from the dict, or returns a default value if the key isn't found. Removes the item from the dict if the key was found.

Args: cname : The name of the dict to use.

key : A key string to look for.

value : A default to return if key isn't in the dict.

Returns: The retrieved item, or the supplied value.

tdict_put_(*cname*, *key*, *value*)
Handles the tdict_put_() function.

Adds a new value indexed by the key to the dict, or replaces an existing value with the new one.

Args: cname : The name of the dict to use.

key : A key string.

value : A value argument to add or replace.

Returns: True for success, False otherwise.

tdict_update_(*cname*, *value*)
Handles the tdict_update_() function.

Merges the entire contents of an existing dict object into the named tdict.

Args: cname : The name of the dict to use.

value : The dict object to merge in.

Returns: True for success, False otherwise.

unregister()
Remove this extension's functions from the engine.

class ThreadDict_(*name*, *api*, *timeout=20*)
Bases: object

close_()

cmd_(*cmd*, *key=None*, *value=None*)

1.1.10 extensions.utilext module

utilext - utility functions.

This extension adds a few useful functions callable by scripts. Some of these functions may be disabled by options passed in from apyshell.

Make these functions available to a script by adding:

loadExtension_('utilext')

to it. Functions exported by this extension:

input_()
read characters from the terminal, return as a string.

system_()
execute a string in the user's default shell. *

getenv_()
return the value of an environment variable. *

- may be disabled by option settings

Credits

- version: 1.0.0
- last update: 2024-Jan-05
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023,2024 by Mark Anacker

class UtilExt(*api*, *options={}*)
Bases: object

This class provides utility commands.

__init__(*api*, *options={}*)
Constructs an instance of the UtilExt class.

This instance supplies some useful functions.

Parameters

- **api** – an instance of ExtensionAPI connecting us to the engine.
- **options** – a dict of option settings passed down to the extension.

Returns None

Options

'allow_system' [install the system_() command allowing] the script to run commands in the system shell.

'allow_getenv' [install the getenv_() command to allow] the script to read environment variables.

getenv_(*name*)
Return an environment variable.

This function returns the value of an item in the host application's environment. Since this might be unsafe, it has to be specifically enabled by the host application.

Args:

name : The environment variable to return

Returns:

The contents of the env variable, or None if it isn't valid.

input_(*prompt=None*, *default=None*, ***kwargs*)
Get console input.

This function will (optionally) print a prompt on the console, then wait for (and return) whatever the user types in. A default value may be set to be returned if nothing was entered. A timeout can also be set, either returning a default value or raising an exception if the timeout expires without a console entry.

The timeout default value may be different from the input default.

Args:

prompt : The prompt to display on the console

default : A value to return if there is no input

**kwargs : Various options:

Options:

- timeout : input timeout in seconds.
- todef : default value to return if the timeout expires.
- toraise : If True, raise an Exception when the timeout expires.

Returns: A string with the input (minus the trailing newline), or a sepcified default value.

Raises: Exception('Timed out')

register()
Make this extension's functions available to scripts.

This method installs our script API methods as functions in the engine symbol table, making them available to scripts.

This is called by the ExtensionMgr during loading.

Note:

Functions installed:

- input_() : read characters from the terminal, return as a string.

- system_() : execute a string in the user's default shell.
- getenv_() : return the value of an environment variable.

Parameters None

Returns True : Commands are installed and the extension is ready to use.

False : Commands are NOT installed, and the extension is inactive.

shutdown()
Perform a graceful shutdown.

This gets called by the extension manager just before the extension is unloaded.

system_(*cmd*)
Run a shell command - possibly dangerous. (definitely dangerous!).

This function simply runs whatever string is passed to it in the host environment. Useful for debugging or ad-hoc scripting, it has to be deliberately enabled by the host application.

Args:

cmd : A command to run.

Returns:

The output of the command.

unregister()
Remove this extension's functions from the engine.

1.2 Module contents

Extensions - Add additional functionality to scripts.

This package contains modules that add special fuctions to apyshell. Scripts can use the loadExtension_() command to add any of these modules that are in the extensionsdir directory, and make their commands available to the scripts. Extensions may be unloaded as well.33

Credits

- version: 1.0.0
- last update: 2023-Nov-17
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023 by Mark Anacker

PYTHON MODULE INDEX

e

INDEX

Symbols

A

C

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

W