extensions

Release 1.0

Mark Anacker

Nov 20, 2023

CONTENTS

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

- last update: 2023-Nov-13

- License: MIT

- Author: Mark Anacker <closecrowd@pm.me>

- Copyright (c) 2023 by Mark Anacker

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

> This class manages commands to read and write files.

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

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.

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**
>>> - **data**
>>> - **handler**
>>> - **maxlines**
>>> - **mode**

typeWrite (*f*, *data*)

1.1.2 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 seperate 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

- last update: 2023-Nov-13

- License: MIT

- Author: Mark Anacker <closecrowd@pm.me>

- Copyright (c) 2023 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

 connect_(*cname='mqttconn'*, ***kwargs*)
  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='mqttconn'*)
  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='mqttconn'*)

 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='mqttconn'*, *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.

> **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

> **Parameters** 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='mqttconn'*, *dest=None*, *data=None*)

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='mqttconn'*, *topic=None*, *handler=None*, *qos=0*)

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

unsubtopic_(*cname='mqttconn'*, *topic=None*)

waiting_(*cname='mqttconn'*)

debug(**args*)

topicmatch(*subsc*, *topic*)

1.1.3 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
- last update: 2023-Nov-13
- License: MIT
- Author: Mark Anacker <closecrowd@pm.me>
- Copyright (c) 2023 by Mark Anacker

class QueueExt(*api*, *options={}*)
> Bases: `object`
>
> This class provides thread-safe queues.
>
> **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 add 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.

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

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.

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*)

1.1.4 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: 2023-Nov-17

- License: MIT

- Author: Mark Anacker <closecrowd@pm.me>

- Copyright (c) 2023 by Mark Anacker

class SqLiteExt(*api*, *options={}*)
>   Bases: `object`
>
>   This class manages commands to use a sqlite3 database.
>
>   **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.
>   >
>   >   ---
>   >
>   >   **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:** The return value. 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.
>
> **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.

1.1.5 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

- last update: 2023-Nov-17

- License: MIT

- Author: Mark Anacker <closecrowd@pm.me>

- Copyright (c) 2023 by Mark Anacker

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

> This class provides utility commands.

> **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.

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

- last update: 2023-Nov-17

- License: MIT

- Author: Mark Anacker <closecrowd@pm.me>

- Copyright (c) 2023 by Mark Anacker

PYTHON MODULE INDEX

e