apyshell
Release 1.0

Mark Anacker

Nov 21, 2023

CONTENTS

APYSHELL PACKAGE

1.1 Subpackages

1.2 Submodules

1.2.1 apyshell.apyshell module

apyshell - Python Embedded apy script runnner

This script creates a framework for running lightweight scripts under the apyengine interpreter. It demonstrates embedding and controlling the engine. It can be run either stand-alone, or itself embedded into an application.

Credits

- version: 1.0

- last update: 2023-Nov-17

- License: MIT

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

- Copyright (c) 2023 by Mark Anacker

apyshell(*script*, *basedir='/opt/apyshell/scripts'*, *extensiondir='/opt/apyshell/extensions'*, *extension_opts=None*, *args=None*, *initscript=None*, *globals=False*)
Execute a script file under anyengine.

This is the main entry point.

> **Args:** script : The .apy script to execute.

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

savepid(*pfile*)
Save our current PID if we can.

> **Note:** this doesn't do any checking of the file path. Take care if running under root.

---

usage()
Help message.

1.2.2 apyshell.extensionapi module

extensionapi - The interface between extensions and the engine.

An instance of this class is passed to the extensions at load time to give them an API back into the engine via the ExtensionMgr

Credits

- version: 1.0

- last update: 2023-Nov-20

- License: MIT

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

- Copyright (c) 2023 by Mark Anacker

class ExtensionAPI(*eng*, *parent*, *internal=True*)

Bases: `object`

getSysvar_(*name*, *default=None*)
Return the value of a system variable or a default

getvar_(*vname*, *default=None*)
Return the value of a script variable or a default

handleEvents(*name*, *data*)
Handle callback events.

If an extension allows for handlers to receive events, each callback comes through here. This function passes the name and data object through to the ExtensionMgr to run in the engine.

Args:

name : The name of the handler func().

data : The object to pass as the **only** argument to the handler.

Returns:

ret : A string with the return from the handler()

None : Something failed

isDef_(*name*)
Return True is the name is defined

listDefs_(*exception_list=None*)
Return a list of script def procs

list_Modules_()
Return a list of install_()-ed modules

loadScript_(*filename*, *persist=False*)
Load and execute a script file

logError(*modname=''*, **args*)

regcmd(*name*, *func=None*)
Add a new command callable by scripts

registerCmds(*mdict*)
Make an extension's functions available to scripts

Extensions call here through the api reference to install their commands into the engine.

Args:

mdict : A dict with the commands and function refs.

Returns

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

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

setSysvar_(*name*, *val*)
Set a system variable

setvar_(*vname*, *val*)
Set a script variable

unregisterCmds(*mdict*)
Remove an extension's commands from the engine.

Extensions call here through the api reference to remove their commands from the engine.

Args:

mdict : A dict with the commands and function refs.

Returns

True : Commands have been removed.

False : Something failed.

debug(**args*)

1.2.3 apyshell.extensionmgr module

extensionmgr - Handles extension load/unload for apyshell.

This module supports the extension handling commands for apyshell.

Credits

- version: 1.0

- last update: 2023-Nov-20

- License: MIT

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

- Copyright (c) 2023 by Mark Anacker

class ExtensionMgr(*eng*, *epath*, *options=None*)
Bases: `object`

handleEvents(*name*, *data*)
Handle callback events.

If an extension allows for handlers to receive events, each callback comes through here. This function makes sure the handler (a def func() in the script) is defined. It then executes the handler func **in the context of the thread that calls this method.** This may not be the thread that the majority of the script is running on.

> Args:
>
> > name : The name of the handler func().
> >
> > data : The object to pass as the **only** argument to the handler.
>
> Returns:
>
> > ret : A string with the return from the handler()
>
> > None : Something failed

isExtLoaded_(*ename*)
Handles the isExtLoaded_() function.

This function is used to test whether an extension is currently loaded or not.

> Args:
>
> > ename : The name of the extension to check.
>
> Returns:
>
> > True : The extension is loaded.
>
> > False : It's not loaded.

listExtensions_()
Handles the listExtensions_() function.

Return a list[] of currently-loaded extensions - those that a script has used loadExtension_() to install.

> Args:
>
> > None
>
> Returns:
>
> > A list[] of the loaded extension names.
>
> > None if there are no loaded extensions.

loadExtension_(*ename*)
Handles the loadExtension_() function.

This function loads an extension by name (no paths allowed), if it's in the __availExtensions list. If the list is empty, the extensions directories are scanned.

If the extension is available, it's loaded into the engine's symbol table and the register() function is called to add it's functions.

> Args:
>
> > ename : The name of the extension to load.
>
> Returns:
>
> > True : The extension loaded correctly.
>
> > False : It didn't load.

---

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:

>   - scanExtensions_() : Look through the extensions dir and return a list of available extensions.

>   - listExtensions_() : Return a list of loaded extensions.

>   - isExtLoaded_() : Return True if the extension is loaded.

>   - loadExtension_() : Load an extension into the engine.

>   - unloadExtension_() : Remove an extension from the engine.

Args:

>   None

Returns

>   **True** [Commands are installed and the extension is] ready to use.

>   **False** [Commands are NOT installed, and the extension] is inactive.

---

scanExtensions_()
>   Handles the scanExtensions_() function.

>   This function builds a list of the Python files in the extension director(ies) stored in self.__expath. This can be either a single directory, or a list of them. All .py files in these directories are presumed to be extensions, although there is a signature test performed at extension load time.

>   You can limit which extensions are available simply by limiting which files are placed into these directories.

>>   Args:

>>>   None

>>   Returns:

>>>   A list[] of the available extension names.

>>>   None if there are no extensions.

scanForExtensions(*dir*, *iflag*)

shutdown()
>   Perform a graceful shutdown.

>   Calls the shutdown() method of each loaded extension, making sure they go cleanly.

>   This gets called by apyshell just before is exits.

unloadExtension_(*ename*)
>   Handles the unloadExtension_() function.

>   This function removes a currently-loaded extension from the symbol table, making it unavailable to scripts. It calls the shutdown() and unregister() methods in the extension before removing it.

Args:

ename : The name of the extension to unload.

Returns:

True : The extension unloaded correctly.

False : There was an error.

quoteSpecial(*orig*)

1.2.4 apyshell.support module

support - Various stand-alone support functions for apyshell.

This file contains various utility functions used by apyshell.

Credits

- version: 1.0

- last update: 2023-Nov-13

- License: MIT

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

- Copyright (c) 2023 by Mark Anacker

checkFileName(*fname*)
    Check a file name

    Checks the given filename for characters in the set of a-z, A-Z, _ or -

        Args: fname : The name to check

        Returns: True if the name is entirely in that set False if there were invalid char(s)

debugMsg(*source=''*, **args*)

enableDebug(*arg*)

errorMsg(*source=''*, **args*)

get_queue(*q*, *defvalue=None*, *timeout=0*)

getparam(*table*, *key*, *default*, *remove=False*)

retError(*api*, *module*, *msg*, *ret=False*)

sanitizePath(*path*)
    Clean a path.

    Remove dangerous characters from a path string.

        Args: path : The strin with the path to clean

        Returns: The cleaned path or None if there was a problem

setparam(*table*, *key*, *value*, *replace=False*)

swapparam(*table*, *oldkey*, *newkey*, *value=None*)

unlock__(*lock*)

1.3 Module contents

ApyShell - Python Embedded apy script runnner

This package creates a framework for running lightweight scripts under the apyengine interpreter. It demonstrates embedding and controlling the engine. It can be run either stand-alone, or itself embedded into an application.

The companion project "apyengine" has documentation and examples of how to use the apyengine outside of this package. The entire apyengine package is included in this package for convenience. <https://github.com/closecrowd/apyengine>

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

a