mylinux Documentation

Release 0.0.3

Uros Jarc

October 06, 2015

Contents

Contents:

CHAPTER 1

mylinux

Helper to install and managing your custom linux system

- Free software: BSD license
- Documentation: https://mylinux.readthedocs.org.

1.1 Features

- Text base user interfaces.
- Advance user mistakes error catching system.
- Logs reporting
- Bash integration
- Testing installed applications
- Custom scripts execution
- Group packages by class and module
- And many many more...

1.2 Todo

- Add better support for bash... ”..” & ”.”

1.3 Future

- Example of good app: yeoman
- Create application logo (hands holding baby tux)
- **Create github.io page for representation**
  - Small and sweet

- **Add short and advance README.rst**
  - Create youtube demo video.
  - Database of mylinux.data
- **Documentation**
  - Add dia graphs
- Register application to the apt-get etc...
- Ustvari checksum za aplikacijo

CHAPTER 2

Installation

At the command line:

```
$ easy_install mylinux
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv mylinux
$ pip install mylinux
```

CHAPTER 3

Usage

To use mylinux is really simple!

1. Open terminal.
2. Write mylinux command for more info.

Sample:

```
mylinux
```

This is joust simple example, for more info see the mylinux section.

CHAPTER 4

Documentation

Warning: In documentation you will see `Other Parameters` list, which will hold custom information obout the source code.
You will see `Other Parameters` in:

- `modules`, to create info for `module variables`.
- `methods`, to explain what the method is doing!

Note:

For types in documentation you will see:

- int : Integer
- str : String
- cls : Class
- mod : Module
- boo : Boolean
- arr-<type> : Array of <type>

The basic rule is shorting name on 3 characters.

4.1 mylinux package

4.1.1 Subpackages

mylinux.constants package

Submodules

mylinux.constants.error module

Here are contained all error messages that are used for error report, raising exception, etc...

`mylinux.constants.error.`**`newErrorDiscoveryMsg`** **= 'UKNOWN ERROR: Inform mylinux developers obout this er**

`mylinux.constants.error.`**`singletone`** **= 'Singletone has been already inited'**

class mylinux.constants.error.**Format**
Bases: object

Class for filing missing msg. strings.

static keyMissingIn(*key*, *where*)

Parameters

- **key** (*str*) – What key is missing.
- **where** (*str*) – Where is key missing.

Returns Logical msg. string.

static keyNotValidFor(*key*, *what*)

Parameters

- **key** (*str*) – What key is not valid.
- **what** (*str*) – Why is not valid.

Returns Logical msg. string.

static alreadyExistIn(*where*)

Parameters **where** (*str*) – Where object already exist.

Returns Logical msg. string.

static notInstalled()

Returns Logical msg. string.

static notExistIn(*where*)

Parameters **where** (*str*) – Where object not exist.

Returns Logical msg. string.

static notFoundIn(*what*, *where*)

Parameters

- **what** (*str*) – What is not found.
- **where** (*str*) – Where is not found.

Returns Logical msg. string.

mylinux.constants.runTime module

mylinux.constants.runTime.**username = None**
True user name.

mylinux.constants.runTime.**homePath = '/home/root'**
Home path for the user.

mylinux.constants.runTime.**filesPath = '/home/root/.mylinux'**
Path to the folder where mylinux reads application data/ settings.

mylinux.constants.setup module

All variables that is setup.py using for self configuration/installation etc...

Note: Here are listed only few variables.

`mylinux.constants.setup.`**`version`** **= '0.0.3'**

`mylinux.constants.setup.`**`author`** **= 'Uros Jarc'**

`mylinux.constants.setup.`**`author_email`** **= 'jar.fmf@gmail.com'**

`mylinux.constants.setup.`**`url`** **= 'https://github.com/urosjarc/mylinux'**

`mylinux.constants.setup.`**`entry_points`** **= {'console_scripts': ['mylinux = mylinux.__main__:main']}**

Module contents

Other Parameters

- **setup (mod)**
- **error (mod)**
- **runTime (mod)**

mylinux.libs package

Submodules

mylinux.libs.AppErr module

Here are defined all possible application type errors. They are defined to show more info to the user/developer, and to catch errors that hides from developers.

exception `mylinux.libs.AppErr.`**`model`**(*message*)
Bases: `exceptions.Exception`

Exception type for model error.

Parameters **`message`** (*str*) – Message for error info.

Variables

- **`message`** (*str*) – Message for error info.
- **`exitCode`** (*int*) – Exit code.

`__str__`()

Returns Error message.

exception `mylinux.libs.AppErr.`**`user`**(*message*)
Bases: `exceptions.Exception`

Exception type for user error.

Parameters **`message`** (*str*) – Message for error info.

Variables

- **message** (*str*) – Message for error info.
- **exitCode** (*int*) – Exit code.

__str__()

Returns Error message.

exception mylinux.libs.AppErr.**developer**(*message*)
Bases: exceptions.Exception

Exception type for developer error.

Parameters **message** (*str*) – Message for error info.

Variables

- **message** (*str*) – Message for error info.
- **exitCode** (*int*) – Exit code.

__str__()

Returns Error message.

mylinux.libs.Assert module

Warning: Every unit test module should have Assert.coverage test in setUp class!

class mylinux.libs.Assert.**Assert**
Bases: unittest.case.TestCase

Hellper class for adding functionality to unittesting.

Variables **maxDiff** (*boo*) – You want to see max diff of error?

classmethod getErrorMessage(*err*, *replaceChar=""'*, *newChar=''*)
Getting error message base on python version.

Parameters

- **err** (*cls*) – Error class.
- **replaceChar** (*str*) – Replace which string in error msg.
- **newChar** (*str*) – Put new string in error msg.

Returns Error message string.

coverage(*testClass*, *testClassSkiped*, *typeRemove*, *typeElements*, *typeSkiped*)
Custom coverage testing for testing class.

Parameters

- **testClass** (*cls*) – Test coverage for testing class.
- **testClassSkiped** (*arr-str*) – What methods don't have testing logic?
- **typeElements** (*arr-cls*) – What class/instance is testClass testing?
- **typeRemove** (*arr-str*) – Remove strings from typeElements example: '_Controller'.
- **typeSkiped** (*arr-str*) – What methods testClass don't want to test?

Raises

- `ValueError` – If typeElements is empty.
- `ValueError` – If testClass don't containes **test_** string.
- `ValueError` – If test are missing in testClass.

sameDirStructures(*dir1*, *dir2*)
Check if dirs have the same structure

Parameters

- **dir1** (*str*) – Path to first dir structure.
- **dir2** (*str*) – Path to second -//-.

Raises `ValueError` – If dirs are diferent.

termCommand(*strCommand*)
Catching terminal stdout/stderr

Parameters **strCommand** (*str*) – String command to be executed.

Returns [True or False, stdout, stderr] array will return True if no error is catched, else False.

mylinux.libs.decorator module

Here are defined all custom decorators.

mylinux.libs.decorator.**overrides**(*interface_class*)
Decorator to override class method

Parameters **interface_class** (*cls*) – Which class is being overrided?

Raises `ValueError` – If method is not located in interface_class.

Returns Method that is overriding interface_class.

mylinux.libs.decorator.**override**(*function*)
To visualy see that you are overriding some class method.

Parameters **function** (*fun*) – You can joust do @override and it will pass function in args.

Returns Method that is overriding interface_class.

class mylinux.libs.decorator.**ClassProperty**
Bases: `property`

Make class property (not instance property).

Parameters **property** (*fun*) – You can joust do @ClassProperty and fun will be passed in args.

Returns Value from property function.

mylinux.libs.e2eData module

class mylinux.libs.e2eData.**E2E_data**
Hellper class for integration tests.

This class will simulate mylinux data structure for the executing tests.

Variables

- ***filesPath*** (*str*) – Path where temporary tests will be located.
- ***packagesPath*** (*str*) – Relative path in fielsPath to package location.

- ***scriptsPath*** (*str*) – Relative path -//-
- ***installedPath*** (*str*) – Relative path -//-

Example of usage:

```
E2E_data.setUp(
        installed={
                u'package': {
                        u'configState': u'error',
                        u'created': u'%d-%d-%d %d:%d:%d.%d',
                        ...
                },
                ...
        },
        packages={
                'package': {
                        'info': {'class': 'class', ... },
                        'scripts': {
                                'config-purge': 'printf CP...',
                                'config-install': 'printf CI...',
                                ...
                        }
                },
                ...
        }
)

E2E-test-data.tearDown()
```

filesPath = 'E2E-test-data'

packagesPath = 'E2E-test-data/packages'

scriptsPath = 'E2E-test-data/scripts'

installedPath = 'E2E-test-data/installed.json'

classmethod tearDown()
Remove and clean testing file structure.

classmethod setUp(*installed={}*, *scripts={}*, *packages={}*)
Setup testing file structure.

Parameters

- **installed** (*dic*) – Dictionary if installed data.
- **scripts** (*dic*) – All main scripts names and data.
- **packages** (*dic*) – List package names and his structures.

Raises ValueError – If testing structure already exist.

classmethod getData()
Get all data info from filesPath.

Returns

Dictionary of all files in files structure.

Example of returning data:

```
                {
                        '<ROOT>/installed.json': {
                                u'package': {
                                        u'configState': u'None',
                                        u'created': u'%d-%d-%d %d:%d:%d.%d',
                                        u'lastChange': u'%d-%d-%d %d:%d:%d.%d',
                                        u'packageState': u'None'},
                                }
                        },
                        '<ROOT>/packages/package/info.json': {
                                u'class': u'class',
                                u'info': u'info',
                                u'module': u'module'
                        },
                        '<ROOT>/packages/package/scripts/config-install': ['printf CI...'],
                        '<ROOT>/packages/package/scripts/config-purge': ['printf CP...'],
                        '<ROOT>/packages/package/scripts/package-install': ['printf PI...'],
                        '<ROOT>/packages/package/scripts/package-purge': ['printf PP...'],
                        '<ROOT>/packages/package/scripts/test.py': ['printf T...']
                }
```

Module contents

Other Parameters

- **module_Assert (mod)**
- **module_e2eData (mod)**
- **decorator (all-mix)**
- **AppErr (all-cls)**
- **Assert (cla)**
- **E2E_data (cla)**

mylinux.model package

Subpackages

mylinux.model.config package

Submodules

mylinux.model.config.Info module

class `mylinux.model.config.Info.`**`Info`**(*path*)

Class for project info.json file

Parameters **`path`** (*str*) – Path to the info.json file.

Variables

- **`__path`** (*str*) – See args...
- **`__data`** (*dic*) – Data from info file on __path.

- **__requiredDataKeys** (*arr-str*) – What dic keys info file should have.

Other Parameters **Update __data with info data.**

requiredDataKeys = ['info', 'class', 'module']

__call__()

Returns __data

__getitem__(*item*)

Parameters **(str)** (*item*) –

Returns __data[item]

path

Returns __path

update()

Checking and updating __data with new data

Other Parameters

- **Check if file on __path exist.**
- **Check if json on __path is correct.**
- **If pre check fails show error report.**

_Info__requiredDataKeys = ['info', 'class', 'module']

mylinux.model.config.Installed module

class mylinux.model.config.Installed.**Installed**(*path*)

Class for handeling installed file.

Parameters **path** (*str*) – Absolute path to installed file.

Variables

- **__inited** (*boo*) – Was class allready inited?
- **__selectTableRule** (*str*) – Define CSV like decision table rule for selecting packages.
- **__executeTableRule** (*str*) – Define CSV like decision table rule for executing packages.
- **__selectTable** (*cla*) – DecisionTable class activated with select table rule.
- **__executeTable** (*cla*) – DecisionTable class activated with execute table rule.
- **__requirdPackageKeys** (*arr-str*) – Required packages keys in every package which is installed.
- **__path** (*str*) – See args info.
- **__data** (*dic*) – All installed data.

Raises AppErr.developer – If class was already inited.

Other Parameters **Set data to __data.**

Note: Installed class for every package executing/selecting action is using decisionTable package which makes logic visual more nice than raw if/else/elif code...

For more info visit: https://decisiontable.readthedocs.org

__call__()

Returns __data

__getitem__(*item*)

Parameters **(str)** (*item*) –

Returns __data[item]

path

Returns __path

selectTableRule

Returns __selectTableRule

executeTableRule

Returns __executeTableRule

requiredPackageKeys

Returns __requiredPackageKeys

getNames()
Get packages names from installed

Returns Array of str. of installed packages.

packageAfterExecution(*packageName*, *err_package*, *err_config*, *stdout=''*, *stderr=''*)
Set package in installed new set of data

Parameters

- **packageName** (*str*) – Package which states will be updated.
- **err_package** (*str*) – Does have package after execution error?
- **err_config** (*str*) – Does config have execution error?
- **stdout** (*str*) – Stout from execution.
- **stderr** (*str*) – Possible error message from terminal.

Other Parameters

- **Get package and config state base on execution information.**
- **Update package in installed with new set of data.**

decidePackageExecution(*packageName*)
Decide what action should be executed on package.

Parameters **packageName** (*str*) – Package name

Returns [package state executing decision, config -//- decision]

getStates(*key*, *packageNames*, *filler*)
Hellper method for TUI select class.

Get infos data value from all installed package key. If package is not installed replace states with filler.

Parameters

- **key** (*str*) – Package key from installed package data.
- **packageNames** (*arr-str*) – All package names so that key value is extracted.

- **filler** (*str*) – If package is not in installed replace value with this.

Returns If not packageNames return arr-str of all key values from installed packages data. If packageNames return arr-str of all selected package key values from installed packages data.

createOrUpdateSelected(*config*, *action*, *packageNames*)
Method for creation or updating installed packages data.

If package in installed exist it will update its data. Else the package will be created.

Parameters

- **config** (*boo*) – Is action want to be executed on configs too?
- **action** (*str*) – What action will be made on packageNames?
- **packageNames** (*arr-str*) – What packages names should be executed with an action?

Other Parameters

- **If package do not exist in installed create one with None package state and None config state.**
- **If package do not exist but don't have right action show error report.**
- **Make select decision for package base on action and config boolean value.**
- **If right decision was not finded show hinting report.**
- **If decision was finded change package values.**

override()
Update or override __data variable

Other Parameters

- **Check if file on __path exist.**
- **Check if json format is ok.**
- **If pre checks fail show error report.**

_Installed__changePackageValues(*packageName*, ***args*)
Handler for updating packages data.

Parameters

- **packageName** (*str*) – Package name whose values are going to change.
- ****args** – Keys and values for package data.

Other Parameters

- **Check if key in args is permited to be changed.**
- **If check fails show error report.**
- **Set key data to new value from args.**
- **Update last change value in package.**

_Installed__createPackage(*packageName*)
Create new installed package.

Parameters **packageName** (*str*) – Name for new package.

Other Parameters

- **Check if packageName don't exist in installed.**

- **If pre check fails show error report.**
- **Create new package in installed.**

_Installed__executeTable = <decisionTable.DecisionTable.DecisionTable object>

_Installed__executeTableRule = '\n\t\tpackageState configState execute err_package err_config new_packageSta

_Installed__inited = False

_Installed__requiredPackageKeys = ['created', 'configState', 'stdout', 'lastChange', 'stderr', 'packageState']

_Installed__selectTable = <decisionTable.DecisionTable.DecisionTable object>

_Installed__selectTableRule = '\n\t\tpackageState configState config action new_packageState new_configState

_Installed__setData()

Seting installed data.

Other Parameters

- **Check if file on __path exist.**
- **Check if json format is correct.**
- **If pre checks fails show error report.**
- **Set __data with data from __path.**

mylinux.model.config.Package module

class mylinux.model.config.Package.**Package**(*name*, *path*)

Class for handeling all package data.

Parameters

- **name** (*str*) – Package name.
- **path** (*str*) – Absolute path to the package folder.

Variables

- **__requiredScriptNames** (*str*) – What scripts must package scripts folder have?
- **__relScriptsPath** (*str*) – Relative path to the scripts folder.
- **__relInfoPath** (*str*) – Relative path to the info package file.
- **__name** (*str*) – Name of the package.
- **__path** (*str*) – See args info.
- **__info** (*cls*) – Info class instance.
- **__scripts** (*arr-cls*) – Every package script instances.

requiredScriptNames = ['config-install', 'config-purge', 'package-install', 'package-purge', 'test.py']

relScriptsPath = '/scripts'

relInfoPath = '/info.json'

scriptsPath

Returns Absolute scripts path

infoPath

Returns Absolute info path

name

Returns __name

path

Returns __path

info

Returns __info

scripts

Returns __scripts

getScript(*name*)

Parameters **name** (*str*) – Script name

Returns Script instance

Other Parameters

- **Check if script name match.**
- **If check fails show error report.**

_Package__relInfoPath = '/info.json'

_Package__relScriptsPath = '/scripts'

_Package__requiredScriptNames = ['config-install', 'config-purge', 'package-install', 'package-purge', 'test.py']

_Package__setInfo()
Set __info variable with Info class instance.

_Package__setScripts()
Set __scripts variable with all Scripts class instance.

Other Parameters

- **Get all scripts file names in scripts path.**
- **Check if every script name is inrequired Script variable.**
- **If pre check fails show error report.**
- **Fill __scripts variable with Script instances.**

mylinux.model.config.Script module

class mylinux.model.config.Script.**Script**(*name*, *path*)
Class for handeling scripts data etc...

Parameters

- **name** (*str*) – Name of the script.
- **path** (*str*) – Absolute path to the scirpt.

Variables

- **__data** (*str*) – All data from file on path.
- **__name** (*str*) – Name of the script.
- **__path** (*str*) – Absolute path to the script.
- **__cmdArrays** (*arr-arr-str*) – Script commands.

Other Parameters **Update __data and __cmdArrays variables.**

__call__()

Returns __data

name

Returns __name

path

Returns __path

cmdArrays

Returns __cmdArrays

update()

Update __data and __cmdArrays

Other Parameters

- **Check if file exist on __path.**
- **If check fails show error report.**
- **Set __data variable.**
- **Split file on lines.**
- **Split every line on words.**
- **Set __cmdArrays variable.**

Module contents

Other Parameters

- **module_Info (mod)**
- **module_Installed (mod)**
- **module_Package (mod)**
- **module_Script (mod)**
- **Script (cla)**
- **Package (cla)**
- **Installed (cla)**
- **Info (cla)**

mylinux.model.linux package

Submodules

mylinux.model.linux.Executor module

class mylinux.model.linux.Executor.**Executor**

Main model terminal executor.

This class handles shell bash executing, returning stdout/stderr etc...

Variables

- **__errorsCount** (*int*) – Count errors.
- **__executeCount** (*int*) – Count how many command was executed.
- **__status** (*int*) – Holding status variables for user visualization of the progress.

_Executor__display(*line*)

Print display line of executing info.

Parameters **line** (*str*) – What string should be showed to the user when there is active subprocess.

_Executor__getStatusLine()

Format and return status line to be showed.

Returns Pretty status line

_Executor__supportedCmdArray(*cmdArray*)

errorsCount

Returns __errorsCount

execute(*cmdArray*)

Main executing method for shell executing.

Parameters **cmdArray** (*arr-str*) – Commands to be executed in shell.

Returns [Pass (boo), stdout, stderr]

Other Parameters

- **Add 1 to the __executeCount variable.**
- **Update status with cmd string.**
- **Make new processs for cmdArray and pipe stdout/stderr.**
- **Check for user interupts (KeyboardInterrupt,SystemExit).**
- **Check if error has been caught, if tru raise __errorsCount variable**
- **Display stdout/stderr line.**
- **Check for exit code**
- **Close process.**

executeArrays(*cmdArrays*)

Wraper around execute method.

Parameters **cmdArrays** (*arr-arr-str*) – Command arrays that must be executed.

Returns [Pass (boo),stdoutAll,stderrAll]

Other Parameters

- **Execute array**
- **Fill stdout/stderr with data**

executeCount

Returns __executeCount

exit

status

Returns __status

Module contents

Other Parameters

- **module_Executor (mod)**
- **Executor (cls)**

Submodules

mylinux.model.Files module

class mylinux.model.Files.**Files**(*mainPath*)

This is main Files class.

This class is responsible for all general side effects on mylinux filesystem. For example multiple package searching, creation of new package etc...

Parameters **mainPath** (*str*) – Path to folder where is located mylinux file structure.

Variables

- **__inited** (*boo*) – Was class allready inited?
- **__chmodOfNewPackage** (*str*) – chmod code of created package data.
- **__chmodOfInitFolderStructure** (*str*) – Set this chmod code to new init folder.
- **__mainPath** (*str*) – Save mainPath from args.
- **__packages** (*arr-ins*) – Package class instances array.
- **__scripts** (*arr-ins*) – All script instances.
- **__installed** (*ins*) – Instance of installed class.

Raises AppErr.developer – If class was allready inited.

Other Parameters

- **Update __installed attribute with data.**
- **Update __packages att. with data.**
- **Update __scripts att. with data**

chmodOfNewPackage = '744'

scriptsPath

Returns Absolute path to scripts folder.

packagesPath

Returns Absolute path to packages folder.

installedPath

Returns Absolute path to installed file.

mainPath

Returns __mainPath

installed

Returns __installed

packages

Returns __packages

createPackage(*name='newPackageName'*, *info={}*, *scripts={}*)
Create new package.

Parameters

- **name** (*str*) – Name of new package.
- **info** (*dic*) – Info data of new package.
- **scripts** (*dic*) – Scripts folder structure and data of new package.

Other Parameters

- **Check required data in info.**
- **Check if name is accepted.**
- **If pre checks fail show error report.**
- **Make new project structure.**

getScript(*name*)

getPackages(*names*)
Get packages instances

Parameters **names** (*arr-str*) – Required package names.

Returns Array of packages instances

getModules()
Get sorted by name module values from package info data.

Returns Array of sorted and unique of all package module names.

getClasses()
Get sorted by name class values from package info data.

Returns Array of sorted and unique of all package class names.

getPackageNames()
Get all package names.

Returns Sorted array of all packages names.

getInfoValues()
Get all info values from info data.

Returns Array of all package info values.

_Files__chmodOfNewPackage = '744'

_Files__inited = False

_Files__relInstalledPath = '/installed.json'

_Files__relPackagesPath = '/packages'

_Files__relScriptsPath = '/scripts'

_Files__updateInstalled()
Update __installed instance.

_Files__updatePackages()
Update __packages variable.

_Files__updateScripts()
Update __scripts variable.

Module contents

Other Parameters

- **module_Files** (*mod*) – Module of Files class.
- **Files** (*cls*) – Files class
- **config** (*mod*) – Config module.
- **linux** (*mod*) – Linux module.

mylinux.view package

Subpackages

mylinux.view.tuiElement package

Submodules

mylinux.view.tuiElement.CreateApp module

class mylinux.view.tuiElement.CreateApp.**CreateApp**
Bases: npyscreen.apNPSApplicationManaged.NPSAppManaged

requiredModelValues = ['hints', 'packageName', 'package-install', 'package-purge', 'config-install', 'config-purge', '

onStart()

onCleanExit()

checkModelValues()

setModelValues()

exit(***returned*)

mylinux.view.tuiElement.CreateForm module

class mylinux.view.tuiElement.CreateForm.**CreateForm**(*name=None*, *parentApp=None*, *framed=None*, *help=None*, *color='FORMDEFAULT'*, *widget_list=None*, *cycle_widgets=False*, **args*, ***keywords*)

Bases: npyscreen.fmActionForm.ActionForm

on_ok()

on_cancel()

create()

afterEditing()

init()

setNextForm(*name=None*)

hintingEvent(*widget*, *hints*, *key*)

classNameEvents(*key=None*)

moduleNameEvents(*key=None*)

setHandlers()

mylinux.view.tuiElement.LogsApp module

class mylinux.view.tuiElement.LogsApp.**LogsApp**

Bases: npyscreen.apNPSApplicationManaged.NPSAppManaged

requiredModelValues = ['packageName', 'stdout', 'stderr']

onStart()

onCleanExit()

checkModelValues()

setModelValues()

exit(***returned*)

mylinux.view.tuiElement.LogsForm module

class mylinux.view.tuiElement.LogsForm.**LogsForm**(**args*, ***keywords*)

Bases: npyscreen.fmActionFormV2.ActionFormMinimal

on_ok()

on_cancel()

create()

afterEditing()

init()

setNextForm(*name=None*)

mylinux.view.tuiElement.SelectApp module

class mylinux.view.tuiElement.SelectApp.**SelectApp**

Bases: npyscreen.apNPSApplicationManaged.NPSAppManaged

requiredModelValues = ['action', 'packages', 'modules', 'classes', 'packageNames', 'packageInfos', 'packageStates',

onStart()

onCleanExit()

checkModelValues()

setModelValues()

exit(*packageNames*)

mylinux.view.tuiElement.SelectForm module

class mylinux.view.tuiElement.SelectForm.**SelectForm**(*name=None*, *parentApp=None*, *framed=None*, *help=None*, *color='FORMDEFAULT'*, *widget_list=None*, *cycle_widgets=False*, **args*, ***keywords*)

Bases: npyscreen.fmActionForm.ActionForm

on_ok()

on_cancel()

create()

afterEditing()

init()

setNextForm(*name=None*)

ModulesEvents(*key=None*)

ClassesEvents(*key=None*)

setHandlers()

Module contents

Submodules

mylinux.view.Tli module

class mylinux.view.Tli.**Tli**

Bases: object

errorHeaders = [u'Status', u'Error', u'Message']

reportHeaders = [u'Status', u'Package', u'Message']

classmethod raiseError(*errorType*, *title*, *array*, *headers=None*)

classmethod printReport(*title*, *array*, *headers=None*)

_Tli__allStructure = u'\n>>> {}{}'

classmethod **_Tli__colorSpecialStrings** (*array*)

_Tli__errorHeaders = [u'Status', u'Error', u'Message']

_Tli__errorTitle = u'\x1b[31mPROBLEM : {}\x1b[39m\n'

classmethod **_Tli__getErrorString** (*title*, *array*, *headers=None*)

classmethod **_Tli__getFormatedString** (*title*, *array*, *headers*)

_Tli__greenStrings = [u'OK', u'SUCCESS', u'PASS', u'ok']

_Tli__redStrings = [u'FAIL', u'ERROR', u'err']

_Tli__reportHeaders = [u'Status', u'Package', u'Message']

_Tli__reportTitle = u'REPORT : {}\n'

_Tli__yellowStrings = [u'WARNING', u'None', u'none']

mylinux.view.Tui module

class mylinux.view.Tui.**Tui**

fork = False

classmethod **runSelectApp** (*modelValues*)

classmethod **runCreateApp** (*modelValues*)

classmethod **runLogsApp** (*modelValues*)

Module contents

4.1.2 Submodules

mylinux.Controller module

class mylinux.Controller.**Controller** (*filesPath*)

Bases: object

Main MVC controller of the package.

Parameters **filesPath** (*str*) – Path to the file where main data is located.

Variables

- ***_Controller__inited*** (*boo*) – If False class was not inited yet.
- ***Files*** (*ins*) – Files instance.

Raises AppErr.developer – If singletone was allready inited.

Note: This class is singletone which checks __inited, if instance was allready created.

---

execute (*ACTION*, *CONFIG=False*, *PACKAGE_NAMES=None*)

Executing actions on packages.

Parameters

- **ACTION** (*str*) – Execute action on packages.

- **CONFIG** (*boo*) – If action will be made on package config too.
- **PACKAGE_NAMES** (*arr-str*) – Name of package folder.

Other Parameters

- **Start TUI if you want to select packages.**
- **Create checks if action is allowed for all selected packages.**
- **Execute action on packages.**
- **Make report of executing.**

create (*PACKAGE_NAME*)
Guide to create new package.

Parameters **PACKAGE_NAME** (*str*) – Package to be created.

Other Parameters

- **Get existing packages.**
- **Check if candidate already exist.**
- **Start TUI for input package info.**
- **Make file structure for package.**

logs (*PACKAGE_NAME*)
Show logs

Parameters **PACKAGE_NAME** (*str*) – Package name

Other Parameters **Start TUI for logs**

script (*SCRIPT_NAME*)
Execute main mylinux scripts

Parameters **SCRIPT_NAME** (*str*) – Name of the script that is located in base scripts folder

Other Parameters

- **Get info from the script.**
- **Execute script line by line.**
- **Print report**

test ()
Make tests on already installed packages.

Other Parameters

- **Get installed packages.**
- **Execute test script for each package.**
- **Make report.**

check ()
Check for errors for installed packages.

Other Parameters

- **Check config/package state for every installed package.**
- **Make report.**

_Controller__executeSelected(*PACKAGE_NAMES*)
Execute selected packages

Parameters **PACKAGE_NAMES** (*arg-str*) – Execute selected package names.

Other Parameters

- **Make execute decisions for packages.**
- **Execute package scripts with executor.**
- **Update package info data in installed.**
- **Override installed data.**
- **If user want to exit wait for data to be writen.**
- **If user wants really to exit raise user error.**

_Controller__inited = False

_Controller__postExecuteReport(*PACKAGE_NAMES*)
Post execute report for packages

Parameters **PACKAGE_NAMES** (*arg-str*) – Package names to be subject of reporting.

Other Parameters

- **Get packages states.**
- **Report all packages states and if they pass**

_Controller__postSelectCheck(*ACTION*, *CONFIG*, *PACKAGE_NAMES*)
Post check after selection.

Parameters

- **ACTION** (*str*) – Action that were executed.
- **CONFIG** (*boo*) – Were action done ono config too?
- **PACKAGE_NAMES** (*arr-str*) – Action were done on these packages.

Other Parameters

- **Check existance of packages in package folder.**
- **Check existance in installed and permissions of executing action.**
- **If checks fails show user error report.**
- **Override selected package installed flags for them to be executed.**

_Controller__tuiCreate(*PACKAGE_NAME*)
Start TUI for package creation.

Parameters **PACKAGE_NAME** (*str*) – Package to be created.

Returns Dic object similar to argument `modelValues`. For more info see source.

Other Parameters

- **Start TUI for creation.**
- **Return all info obout the new package.**

_Controller__tuiLogs(*PACKAGE_NAME*)
Start TUI for package log.

Parameters **PACKAGE_NAME** (*str*) – Name of the installed package.

Other Parameters

- **Get ‘‘stderr‘‘ and ‘‘stdout‘‘ info from package in installed.**
- **Start logs TUI.**

_Controller__tuiSelect (*ACTION*, *CONFIG*)
Start TUI for selecting packages.

Parameters

- **ACTION** (*str*) – Joust for user visualization in TUI.
- **CONFIG** (*str*) – Joust for user vis. in TUI

Returns Arr-str of selected packages names.

Other Parameters

- **Get packages base on action and config.**
- **Start TUI for selecting packages.**

4.1.3 Module contents

Module will not import any other packages/modules only what is needed for setup.py and docs. configuration.

Other Parameters

- **SETUP (mod)**
- **__author__** (*str*) – Project author.
- **__email__** (*str*) – Author email
- **__description__** (*str*) – Project description.
- **__version__** (*str*) – Curent version.

mylinux.**initAppData** (*appData*, *filesPath*, *username*)

Entry points

Main text user interface for application. Main file for all entry points for this project.

mylinux.__main__.**main** (*ARGV=None*, *TESTING=False*)
Main mylinux entry point method.

Parameters

- **ARGV** (*arr*) – User defined sys.argv array.
- **TESTING** (*boo*) – If true reset all singletones for possible rerun.

Raises

- `AppErr.model` – If controller produce error obout mylinux models.
- `AppErr.user` – If user make error.
- `AppErr.developer` – If developer makes mistake.
- `Exception` – If unknown error is produced by controller.

Other Parameters

- **If testing reset singletones.**
- **Init main controller.**
- **Call controller**
- **Send exit codes to shell.**

Note: Method will produce at the end of the executing `exit code` that will be passed to the shell.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/urosjarc/mylinux/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

5.1.4 Write Documentation

My linux could always use more documentation, whether as part of the official My linux docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/urosjarc/mylinux/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *mylinux* for local development.

1. Fork the *mylinux* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mylinux.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mylinux
$ cd mylinux/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 mylinux tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/urosjarc/mylinux/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_mylinux
```

CHAPTER 6

Credits

6.1 Development Lead

- Uros Jarc <jar.fmf@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

Version history

7.1 0.0.1 (2015-10-01)

- First release on PyPI.

7.2 0.0.2 (2015-10-04)

- Adding interuption handeling on soft spots.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

m

Symbols

A

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V