system builder

April 7, 2025

Jay Convertino

Contents

1 Usage

1.1 Introduction

System build is a library that will parse a yaml project file, and a yaml command file to generate a list of commands to execute in threads. These can be concurrent or singular in fashion.

1.2 Dependencies

The following are the dependencies of the cores.

- python 3.X
- python progressbar
- python git (creator)

1.3 Example

The file creator.py is an example piece of code for building a project.

```
import system.build

print(f'FUTURE CODE HERE')
```

2 Code Documentation

Natural docs is used to generate documentation for this project. The next lists the following sections.

- **init** python init code
- **builder** system builder library.
- **creator** system builder example code.

__init__.py

AUTHORS

JAY CONVERTINO

DATES

2025/03/08

INFORMATION

Brief

builder define for packages

License MIT

Copyright 2025 Jay Convertino

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

commandCompiler

commandCompiler

Parse yaml files into commands for command executor

FUNCTIONS

__init__

```
def __init__(
self,
yaml_cmds
 =
None,
yaml_projects
 =
None,
target
 =
None
)
```

Setup class

listCmds

```
def listCmds(
self
)
```

Print a list of all the commands available

listProjects

```
def listProjects(
self
)
```

Print a list of all the

clear

```
def clear(
self
)
```

Clear results of create and current yaml file pointer.

setProjects

```
def setProjects(
self,
yaml_projects
)
```

Set a yaml file for project commands

setCmds

```
def setCmds(
self,
yaml_cmds
)
```

Set a yaml file for commands available

create

```
def create(
self,
yaml_cmds
 =
None,
yaml_projects
 =
None,
target
 =
None
)
```

Pass a yaml file to use for processing into format for commandExecutor.

getResult

```
def getResult(
self
)
```

Return dict of dicts that contains lists with lists of lists to execute with subprocess {'project': { 'concurrent': [[["make", "def_config"], ["make"]], [["fusesoc", "run", "--build", "--target", "zed_blinky", "::blinky:1.0.0"]]], 'sequential': [[]]}}

_checkTarget

```
def _checkTarget(
self
)
```

_process

```
def _process(
self
)
```

Create dict of dicts that contains lists with lists of lists to execute with subprocess {'project': { 'concurrent': [[["make", "def_config"], ["make"]], [["fusesoc", "run", "--build", "--target", "zed_blinky", "::blinky:1.0.0"]]], 'sequential': [[]]}}

_load_yaml

```
def _load_yaml(
self,
yaml_file
)
```

Internal function that will load the yaml file that contains project commands or commands.

commandExecutor

commandExecutor

Execute commands create from commandCompileris not a valid selection

FUNCTIONS

clear

```
def clear(
self
)
```

Clear state of commandExecutor to init with no values passed

completed

```
def completed(
self
)
```

return a bool if all items have been completed.

failed

```
def failed(
self
)
```

return a bool indicating failure

stop

```
def stop(
self
)
```

Kill and current processes in build threads

setDryrun

```
def setDryrun(
dryrun
 =
False
)
```

Setting to True sets the project run everything up to the subprocess.Popen call, which is skipped.

setProjects

```
def setProjects(
self,
projects
)
```

Set the projects for the executor to build

runProject

```
def runProject(
self,
projects
 =
None
)
```

Call the internal execute method to start calling up each of the projects to build.

_execute

```
def _execute(
self
)
```

Call subprocess as a thread an_gen_build_cmdsd add it to a list of threads for wait to check on. iterate over projects avaiable and execute commands per project

_subprocess

```
def _subprocess(
self,
list_of_commands
```

```
)
```

Responsible for taking a list of commands and launching threads concurrently or singurely.

_project_cmd_count

```
def _project_cmd_count(
self,
run_types
)
```

Number of commands in a project.

_thread_exception

```
def _thread_exception(
self,
args
)
```

Used to kill all threads once one has failed.

_bar_thread

```
def _bar_thread(
self
)
```

Creates progress bar display in terminal for end user display.

FUNCTIONS

main

```
def main()
```

main execution function

list_deps

```
def list_deps(
deps_file
)
```

open deps text file and print all executable names.

deps_check

```
def deps_check(
deps_file
)
```

Check each line of txt file programs

submodule_init

```
def submodule_init(
repo
)
```

Make sure submodules have been pulled. If not, pull them.

clean

```
def clean()
```

Clean up folders used for output (output and log)

parse_args

```
def parse_args(
argv
)
```

Parse args for tuning build

logger_setup

```
def logger_setup(
debug
)
```

Setup logger for log file