

# builder.py

---

## AUTHORS

---

JAY CONVERTINO

---

## DATES

---

2025/03/08

---

## INFORMATION

---

### Brief

---

parse yaml file to execute build tools

@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_projects
    =
    None,
```

```
yaml_commands
=
None,
target
=
None
)
```

Setup class

## listCommands

---

```
def listCommands(
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

## setCommands

---

```
def setCommands(
self,
yaml_commands
)
```

Set a yaml file for commands available

## create

---

```
def create(
    self,
    yaml_projects
    =
    None,
    yaml_commands
    =
    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 available 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.