system builder

March 9, 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

bob

bob

bob is the builder class for generating systems based on build scripting

FUNCTIONS

stop

```
def stop(
self
)
```

Stop all current builds

run

```
def run(
self
)
```

run the steps to build parts of targets

list

```
def list(
self
)
```

Print a list of all the parsed build commands.

_gen_build_cmds

```
def _gen_build_cmds(
self
)
```

Internal function that will load the yaml file that contains build commands.

_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': [[]]}}

_execute

```
def _execute(
self
)
```

Call subprocess as a thread and 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.

list_projects

```
def list_projects(
yaml,
file_name
)
```

List projects from the yaml file.

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