Build Tool

Release 0.1

Onkar HArsh / Appana Durga Kedareswara Rao

Nov 24, 2022

CONTENTS:

CHAPTER

ONE

NEW_FORMAT_TRY

1.1 build_app module

This module builds an already created app. It doesn't contain any members other than main().

1.2 build_bsp module

This module builds archive files (.a) for the created bsp. These archive files include os, driver and library related archives.

`class build_bsp.BSP(args)`

Bases: `object`

This class contains attributes and functions to build the created bsp. It takes the domain path as input, reads the domain configuration file present in the path to get the required inputs, calls make command inside all the cmake build area and builds the archive (.a) files for baremetal.

`build_lib()`

Compiles all the library source files added in the bsp and generates corresponding .a file.

`gen_libxil()`

Compiles all the driver source files and generates libxil.a

`gen_xilfreertos()`

Compiles the files of freertos bsp folder and generates xilfreertos.a

`gen_xilstandalone()`

Compiles the files of standalone bsp folder and generates xilstandalone.a

`build_bsp.generate_bsp(args)`

Function to compile the created bsp for the user input domain path.

1.3 config_bsp module

This module configures the bsp as per the passed library and os related parameters.

`class config_bsp.`**`Bsp_config`**`(`*`args`*`)`

Bases: *`BSP`*, *`Library`*

This class contains attributes and functions that help in configuring the created bsp. This makes use of BSP and Library class attirbutes and functions to fetch the bsp confiration data and the supporting lib funcs.

`config_bsp.`**`configure_bsp`**`(`*`args`*`)`

This function uses Bsp_config class and configures the bsp based on the user input arguments.

Args:
args (dict): User inputs in a dictionary format

1.4 create_app module

This module creates the template application using the domain information provided to it. It generates the directory structure and the metadata required to build a particular template application.

`class create_app.`**`App`**`(`*`args`*`)`

Bases: *`BSP`*, *`Repo`*

This class helps in creating a template application. It contains attributes and functions that take domain path, and template name as an input, create the directory structure and the metadata(if needed) for the app.

`create_app.`**`create_app`**`(`*`args`*`)`

Function that uses the above App class to create the template application.

Args:
args (dict): Takes all the user inputs in a dictionary.

1.5 create_bsp module

This module cretaes a domain and a bsp for the passed processor, os and system device tree combination.

`class create_bsp.`**`Domain`**`(`*`args`*`)`

Bases: *`Repo`*

This class helps in creating a software domain. This contains functions to create the domain's directory structure, validate the user inputs for the domain on demand and manipulate the cmake toolchain file as per the user inputs.

`apps_cflags_update``(`*`toolchain_file`*`, `*`app_name`*`, `*`proc`*`)`

This function acts as a helper for toolchain_intr_mapping. This adds template application specific compiler entries in the cmake toolchain file of the domain.

Args:

toolchain_file (str): The toolchain file that needs to be updated
app_name (str): Specific app name that needs new entries
proc (str): Proc specific data pertaining to the app.

Returns:
compiler_flags (str): returns the new compiler flags that were set.

`build_dir_struct()`

Creates the include, lib and libsrc folder inside bsp directory.

`toolchain_intr_mapping()`

We have reference toolchain files in embeddedsw which contains default compiler related cmake inputs. This function copies the toolchain file according to user os and processor input in the domain directory. Once copied, it manipulates few entries in the file needed for specific proc /os/app scenario.

In addition, this function also processes the sdt directory to create a a single system dts file that has interrupts correctly mapped as per the input processor.

Returns:

sdt (str):
Processed system device tree file that would be used across the created domain for further processing.

toolchain_file (str):
Toolchain file for cmake infra that would be used across the created domain for builds.

`create_bsp.create_domain(`*args*`)`

Function that uses the above Domain class to create the baremetal domain. Args:

args (dict): Takes all the user inputs in a dictionary.

1.6 library_utils module

This module acts as a supporting module to get/set library related information inside the bsp. It helps in validating the library input, generating the library paramters database and adding/modifying the library when all the criteria are met. It doesnt have any main() function and running this module independently is not intended.

`class library_utils.Library(`*domain_path*, *proc*, *bsp_os*, *sdt*, *cmake_paths_append*, *libsrc_folder*`)`

Bases: *Repo*

This class contains attributes and functions that help in validating library related inputs and adding a library to the created bsp.

`add_lib(`*comp_name*, *is_app=False*`)`

Adds library to the bsp. Creates metadata if needed for the library, runs cmake configure to prepare the build area for library compilation and creates the library configuration of the bsp.

Args:

comp_name (str):
component name (either template or lib). If template depends on certain libs, it fetches them otherwise it adds the passed library.

is_app (bool): To distinguish between lib and template

`copy_lib_src(`*lib*`)`

Copies the src directory of the passed library from the respective path of embeddedsw to the libsrc folder of bsp.

Args:
lib (str): library whose source code needs to be copied

Returns:

libdir (str): Library path inside embeddedsw

srcdir (str): Path of src folder of library inside embeddedsw

dstdir (str): Path of src folder inside libsrc folder of bsp

`get_default_lib_params`(*build_lib_dir*, *lib_list*)

Creates a library configuration data that contains all the available parameters and their values of each library added in the bsp.

Args:

build_lib_dir (str):
Cmake directory where the libraries are configured and compiled

lib_list (str): List of libraries added in the bsp.

Returns:

default_lib_config (dict):
A dictionary that contains all the available parameters and their values of each library added in the bsp.

`validate_lib_in_bsp`(*lib*)

Checks if the passed library name from the user exists in the bsp. This is a helper function to support remove library and set property usecases.

Args:
lib (str): Library name that needs to be validated

`validate_lib_name`(*lib*)

Checks if the passed library name from the user is valid for the sdt proc and os combination. Exits with valid assertion if the user input is wrong.

Args:
lib (str): Library name that needs to be validated

`validate_lib_param`(*lib*, *lib_param*)

Checks if the passed library parameter that needs to be set in library configuration is valid. Exits with a valid assertion if parameter name is wrong. This acts as a helper in set property usecase.

Args:

lib (str): Library name whose config needs to be changed

lib_param (str): Library parameter that needs to be changed

1.7 repo module

This module acts as a supporting module for all the other modules. It helps in validating the embeddedsw repo set in the environment and sets up the correct path for different components within embeddedsw. It doesnt have any main() function and running this module independently is not intended.

`class repo.Repo`

Bases: `object`

This is the base class to get the embeddedsw repo path. This checks if the embeddedsw path is set correctly. If set, this helps in retrieving the corresponding directory path of the component inside embeddedsw.

get_comp_dir(*comp_name*, *is_app=False*)

Returns the absolute path of components in embeddedsw repo.

Args:

comp_name (str): component name whose path is to be retrieved

is_app (bool): if the component passed is a template application.

Returns:

comp_dir (str): Absolute path of the component in embeddedsw

validate_repo(*repo*)

Returns the set absolute path of embeddedsw repo.

Args:

repo (str): The user input for the embeddedsw repo path

Returns:

repo (str):

If user entry is correct, returns the absolute path of that entry

1.8 utils module

This module acts as a supporting module for all the other modules. It contains APIs for small use cases to avoid rewriting of code for those generic requirements. It doesnt have any main() function and running this module independently is not intended.

utils.**add_newline**(*File: str*, *newline: str*) → None

Add a new line at the end of the file.

Args:

File: file path which needs to be modified.

newline: new line that needs to be added.

utils.**copy_directory**(*src: str*, *dst: str*, *symlinks: bool = False*, *ignore=None*) → None

copies the directory from source to destination.

Args:

src: source directory path

dest: destination directory path

symlinks: maintain the symlink while copying

ignore: provide list to ignore files/sub-dirs if any

utils.**copy_file**(*src: str*, *dest: str*, *follow_symlinks: bool = False*, *silent_discard: bool = True*) → None

copies the file from source to destination.

Args:

src: source file path

dest: destination file path

follow_symlinks: maintain the symlink while copying silent_discard: Dont raise exception if the source file doesnt exist

`utils.`**`delete_keys_from_dict`**(*dictionary: dict*, *keys: str*) → dict

Delete keys from the dict. It can detect the key even inside the hierarchical dict.

Args:
dictionary : The dictionary to be processed keys : The key name that needs to be searched and popped out

Returns:
modified_dict : The new dict modified after popping the key

`utils.`**`fetch_yaml_data`**(*config_file: str*, *dir_type: str*) → Optional[dict]

Reads the data from a yaml configuration file, raises assertion if file doesn't exist.

Args:

config_file: The yaml configuration file path
dir_type: Being used for raising a meaningful assertion.

Returns:
data: The read data from yaml file

`utils.`**`find_file`**(*search_file: str*, *search_path: str*)

This api find the file in sub-directories and returns absolute path of file, if file exists

Args:

search_file: The regex pattern to be searched in file names
search_path: The directory that needs to be searched

Returns:
string: Path of the first file that matches the pattern

`utils.`**`find_files`**(*search_pattern*, *search_path*)

This api find the files matching regex directories and returns absolute path of files, if file exists

Args:

search_pattern: The regex pattern to be searched in file names
search_path: The directory that needs to be searched

Returns:
string: All the file paths that matches the pattern in the searched path.

`utils.`**`get_abs_path`**(*fpath*)

This api takes file path and returns it's absolute path

Args:
fpath: Path to get the absolute path from.

Returns:
string: Absolute location of the passed path

`utils.`**`get_base_name`**(*fpath*)

This api takes rel path or full path and returns base name

Args:
fpath: Path to get the base name from.

Returns:
string: Base name of the path

`utils.`**`get_dir_path`**`(`*fpath*`)`

This api takes file path and returns it's directory path

Args:
fpath: Path to get the directory path from.

Returns:
string: Full Directory path of the passed path

`utils.`**`get_original_path`**`(`*fpath*`)`

This api takes file path and returns it's original path. It is equivalent to readlink

Args:
fpath: Path to get the original path from.

Returns:
string: original location of the passed path (after resolving softlink if any)

`utils.`**`is_dir`**`(`*dirpath: str*, *silent_discard: bool = True*`)` → bool

Checks if directory exists.

Args:
dirpath: Directory Path.

Raises:
ValueError (Exception): Raises exception if directory not found.

Returns:
bool: True, if directory is found Or False, if directory is not found.

`utils.`**`is_file`**`(`*filepath: str*, *silent_discard: bool = True*`)` → bool

Return True if the file exists Else returns False and raises Not Found Error Message.

Args:
filepath: File Path.

Raises:
FileNotFoundError: Raises exception if file not found.

Returns:
bool: True, if file is found Or False, if file is not found.

`utils.`**`load_yaml`**`(`*filepath: str*`)` → Optional[dict]

Read yaml file data and returns data in a dict format.

Args:
filepath: Path of the yaml file.

Returns:
dict: Return Python dict if the file reading is successful.

`utils.`**`mkdir`**`(`*folderpath: str*, *silent_discard: bool = True*`)` → None

Create the folder structure, raises Error Message on demand.

Args:
folderpath: Path of the folder structure.

`utils.`**`remove`**`(`*path: str*, *silent_discard: bool = True*`)` → None

Removes any file or folder recursively, if it exists else reports error message based on user demand.

Args:

path: Directory or file path.

`utils.`**`remove_line`**`(`*File: str*, *match_string: str*`)` → None

Remove the lines that match the passed pattern

Args:

File: file path which needs to be modified.

match_string: the string that needs to be searched in the line.

`utils.`**`replace_line`**`(`*File: str*, *search_string: str*, *add_line: str*`)` → None

Replace an existing line that matches the passed string with a new line

Args:

File: file path which needs to be modified.

search_string: the string that needs to be searched in the line.

add_line: New line that needs to be put in.

`utils.`**`reset`**`(`*path: str*`)` → None

Delete the passed path and then recreate it.

Args:

path: Path that needs to be reset

`utils.`**`runcmd`**`(`*cmd*, *logfile=None*`)` → bool

Run the shell commands.

Args:

cmd: shell command that needs to be called

logfile: file to save the command output if required

`utils.`**`update_yaml`**`(`*filepath: str*, *dir_type: str*, *key: str*, *data: Optional[dict]*, *action: str = 'add'*`)`

Update the already created yaml file. Supports the add and remove option to add any new data or remove the existing data in the yaml. Raises assertion if the yaml doesnt exist.

Args:

filepath: the yaml path

dir_type: to raise a meaningful assertion

key: Key that needs to be manipulated.

data: The new data that needs to be updated in yaml in 'add' use case.

action: 'add'/'remove'

`utils.`**`validate_if_exist`**`(`*config_file: str*, *dir_type: str*, *dir_name: str*`)` → None

Raise valid assertion when a file already exists

Args:

config_file : File Path that needs to be checked dir_type, dir_name: Being used for raising a meaningful assertion.

`utils.`**`validate_if_not_exist`**`(`*config_file: str*, *dir_type: str*, *dir_name: str*`)` → None

Raise valid assertion when a file doesnt exist

Args:

config_file : File Path that needs to be checked

dir_type, dir_name: Being used for raising a meaningful assertion.

`utils.`**`write_yaml`**`(`*filepath: str*, *data*`)`

Write the data into a yaml file format

Args:

filepath: the yaml file path

data: the data

1.9 validate_bsp module

This module facilitates the validation of a created BSP with respect to a template application.

`class` `validate_bsp.`**`Validation`**`(`*args*`)`

Bases: *BSP*, *Repo*

This class contains attributes and functions to validate the given bsp w.r.t. the user input template application. This inherits BSP class to inherit all the domain specific data and Repo class to get the embeddedsw related paths.

`get_valid_template_list``()`

This function provides the list of templates that can built using the passed bsp.

`validate_template_for_bsp``()`

This fucntion validates the library dependency of the passed template within the bsp. If the required libs are not available in the bsp, it throws the suitable assertion.

`static validate_template_name``(`*domaindir*, *proc_data*, *bsp_os*, *app*`)`

This function verifies the template name passed by the user. It checks if the name is valid for the given os and proc combination. If not, it raises the suitable assertion. This is being used during domain creation as well (when domain needs to be created for a particular template). Hence, a static function in nature.

Args:

domaindir (str): domain path that contains all the domain specific data

proc_data (dict): App specific data read during initialization

bsp_os (str): os used during domain creation

app (str): template name to be validated

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX