Buildnis

Release 0.2.0

Release-Candidate

19.03.2021

LINKS

Buildnis is a distributed, platform independent build system that can handle C++20 and Fortran modules and is flexible enough to build any language and handle (almost ;) any build step imaginable. It is written in Python and needs a Python interpreter, at least version 3.9.

The main goal is to have a build system with only the minimum needed configuration and specially user interaction to build software on all supported OSes (Linux, Mac OS X and Windows - alphabetically). The list of currently automatically supported compilers and build tools can be found at the chapter *Supported Configurations*. How to customize Buildnis for your needs can be found at *Customizing Buildnis*, how to extend it, that means adding other OSes or CPU architectures or stuff that need changes in the Python code of Buildnis itself, can be found in the chapter *Extending Buildnis*.

To get you started, short how-tos to get Buildnis up and running:

CHAPTER

ONE

INSTALLATION AND USAGE

1.1 Prerequisites

You need Python, at least version 3.9, you can get the latest version at python.org

1.2 Installation

1.2.1 Using a Virtual Environment

I *highly* recommend that you use a virtual environment like Virtualenv or Pipenv to check out Buildnis.

Virtualenv Setup

First install the package `virtualenv` using `pip`:

```
python -m pip install virtualenv
```

then set up a directory to use as the path for your virtual Python environment:

```
python -m virtualenv PATH_TO_YOUR_VENV
```

with `PATH_TO_YOUR_VENV` the directory, in which you want to install the virtual environment. This should generate a script to source (on Linux, OS X and other unixish platforms) or execute (Windows). So, call or source the script:

Windows:

```
PATH_TO_YOUR_VENV\Scripts\activate.bat
```

Unix:

```
source PATH_TO_YOUR_VENV/bin/activate
```

If you now install the packages, they're installed in this virtual environment and can't break your 'real' Python installation.

When you want to leave the virtual environment, call the script `deactivate`. More detailed documentation of Virtualenv you find at the Virtualenv User Guide.

Pipenv Setup

First install the package `pipenv` using `pip`:

```
python -m pip install pipenv
```

Now to activate the virtualenv (`pipenv` uses `virtualenv`), call `pipenv` with the argument `shell`

```
pipenv shell
```

To leave the environment, call `exit`:

```
exit
```

1.2.2 Installation of Buildnis

You can install Buildnis using pip (also in a virtualenv, see *Virtualenv Setup*) or pipenv (see *Pipenv Setup*).

Installation using Pip

The buildnis package package at PyPI can be installed using pip:

```
python -m pip install buildnis
```

To upgrade your installed version use

```
python -m pip install --upgrade buildnis
```

Installation using Pipenv

Using a virtual Python environment with `pipenv`:

```
pipenv install buildnis
```

To upgrade your installed version use

```
pipenv install --upgrade buildnis
```

1.2.3 No Package Installation

You can also use Buildnis without installing a package, by just copying the directory containing the Python source to into your project - that way you can also distribute Buildnis as part of your project.

To do that, copy the directory `buildnis` from Github and call the package from the parent directory of `buildnis`:

```
python -m buildnis
```

If you don't need it or don't want to redistribute it, you can delete the documentation directory `buildnis/doc`, it isn't needed to run the program.

1.3 Usage

The best way to test Buildnis is to check out the test project from Github: Test Project on Github and run Buildnis from this directory:

```
python -m buildnis --generated-conf-dir conf_out test_project/project_config.json
```

This reads the project's configuration `test_project/project_config.json` and stores all generated configurations to the directory `conf_out`.

If you want to delete the generated configuration files, call the program with the option `--distclean`:

```
python -m buildnis --distclean
```

To get an overview of all supported command-line options and arguments call Buildnis with the argument `-h` or `--help`:

```
python -m buildnis --help
```

CHAPTER

TWO

COMMAND LINE ARGUMENTS

Normally (depending on the project's configuration) you do not need to use any argument or target to configure and build software using Buildnis, it does run everything that is needed to be able to build the default targets, but never the `--install` - which would install the generated files - or `--clean`/`--distclean` stages, which would delete the generated files.

So with a default project JSON configuration named `project_config.json` in the current working directory, you only need to call Buildnis like this:

```
python -m buildnis
```

After the build finishes successfully, you can install the generated files using the `--install` argument.

```
python -m buildnis --install
```

Depending where the project is configured to install the files to, you need to run this as user root or as administrator.

For unixish OSes (that use `sudo`):

```
sudo python -m buildnis --install
```

For any other OS you have to start a shell (command interpreter) as an administrator, and install from that shell:

```
python -m buildnis --install
```

Note: With long arguments you only need to input that part of the argument, that makes it unique. E.g. instead of `--distclean` you could also write `--di`

2.1 Show Help Text

The arguments `-h` or `--help` shows the help text.

```
python -m buildnis --help
```

2.2 Show Version

The argument `--version` shows the help text.

```
python -m buildnis --version
```

prints

```
Buildnis 0.2.6
```

Note: `-v` adds verbosity, it is not short for `--version`!

2.3 Main Argument - The Project Configuration

The main argument of the command line is the path to the project configuration JSON file. If none is given, the default of `project_config.json` in the current directory is used.

Example, to build the project with the main configuration file in the directory `test_project`:

```
python -m buildnis ./test_project/project_config.json
```

Note: You can use normal slashes `/` for paths on Windows too, no need for windows-like backslashes (`\`) as path arguments to Python.

2.4 Build Stages

- `--configure`
- `--build`
- `--install`
- `--clean`
- `--distclean`

Example, to configure the project with the main configuration file in the directory `test_project`:

```
python -m buildnis --configure ./test_project/project_config.json
```

Example, to build the default targets of the with the main configuration file in the directory `test_project`:

```
python -m buildnis --build ./test_project/project_config.json
```

Example, to build the targets `documentation` and `fortran_static` of the project with the main configuration file in the directory `test_project`.

Warning: You need to be careful to let the command line parser know the end of the target list. Either use a double dash `--` or add the targets after the project config file.

```
python -m buildnis --build documentation fortran_static -- ./test_project/project_
↪config.json
python -m buildnis ./test_project/project_config.json --build documentation fortran_
↪static
```

Note: You can use normal slashes / for paths on Windows too, no need for windows-like backslashes (\) as path arguments to Python.

2.5 Output and Script Paths

- `--generated-conf-dir DIR_PATH`
- `--conf-script-dir DIR_PATH`

2.6 Logging Options

- `-q` or `--quiet`
- `-v` or `--verbose`
- `--debug` or `-vv` or `--verbose --verbose`
- `--log-file LOG_FILE`

CHAPTER

THREE

CONFIGURATION

CHAPTER

FOUR

BUILD STAGES

CHAPTER

FIVE

LICENSE

Buildnis is licensed under the MIT license:

```
MIT License

Copyright (c) 2021 Release-Candidate

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.
```

CHAPTER

SIX

SUPPORTED CONFIGURATIONS

Supported means that Buildnis can automatically find these executables, if they're installed in a default location or in a directory that is in the PATH.

If the compiler or tool needed for your build isn't in this list, you have to customize and add some scripts in Buildnis or your project, see *Customizing Buildnis*.

If your OS or CPU architecture isn't supported, you have to change Buildnis itself, but that's not much work to do in the Python source code, see *Extending Buildnis*.

6.1 Supported OSes

In alphabetical order:

- Linux (tested with Fedora Rawhide, RedHat, Suse Tumbleweed and Ubuntu)
- Mac OS X (tested with Big Sur and Catalina)
- Windows (tested using Windows 10 and WIndows Server 2019)

6.2 Supported C++ Compilers

All compilers in this list are either free (open source) or free as in free beer for at open source projects or non-commercial use. See the linked websites for details!

6.2.1 Linux

- GCC - the default compiler for Linux. https://gcc.gnu.org/
- Clang https://clang.llvm.org/
- AMD AOCC - uses LLVM https://developer.amd.com/amd-aocc/
- Nvidia HPC Compilers - ex Portland compilers https://developer.nvidia.com/hpc-compilers
- Intel OneAPI C++ and DPC++ https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit.html

6.2.2 Mac OS X

- Clang - the default compiler for OS X, part of XCode package https://developer.apple.com/xcode/
- Intel OneAPI C++ and DPC++ Intel CPU only https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit.html

6.2.3 Windows

- MS Visual Studio cl - the default compiler for Windows https://visualstudio.microsoft.com/vs/
- Clang included in Visual Studio https://visualstudio.microsoft.com/vs/
- Intel oneAPI C++ and DPC++ https://software.intel.com/content/www/us/en/develop/tools/oneapi/base-toolkit.html

6.3 Supported Fortran Compilers

All compilers in this list are either free (open source) or free as in free beer for at open source projects or non-commercial use. See the linked websites for details!

6.3.1 Linux

- GFortran (GCC) - default Fortran compiler for Linux https://gcc.gnu.org/wiki/GFortran
- FLang (LLVM) - Fedora Rawhide has it in its repository https://releases.llvm.org/11.0.0/tools/flang/docs/ReleaseNotes.html
- AMD AOCC - it's called 'optimizing C and C++, but includes a Fortran compiler, FLang' https://developer.amd.com/amd-aocc/
- Nvidia HPC Compilers - ex Portland compilers https://developer.nvidia.com/hpc-compilers
- Intel OneAPI IFort Fortran Classic and IFX, Fortran is included in the HPC kit https://software.intel.com/content/www/us/en/develop/tools/oneapi/hpc-toolkit.html

6.3.2 Mac OS X

- Intel OneAPI IFort Fortran Classic and IFX, Fortran is included in the HPC kit https://software.intel.com/content/www/us/en/develop/tools/oneapi/hpc-toolkit.html

6.3.3 Windows

- Intel oneAPI IFort Fortran Classic and IFX , Fortran is included in the HPC kit https://software.intel.com/content/www/us/en/develop/tools/oneapi/hpc-toolkit.html

6.4 Supported Interpreters

6.4.1 Linux

- Python https://www.python.org/downloads/
- Java https://jdk.java.net/15/

6.4.2 Mac OS X

- Python https://www.python.org/downloads/
- Java https://jdk.java.net/15/

6.4.3 Windows

- Python https://www.python.org/downloads/
- Java https://jdk.java.net/15/

6.5 Supported Documentation Tools

6.5.1 Linux

- Doxygen https://www.doxygen.nl/download.html
- Sphinx - install using `pip` https://www.sphinx-doc.org/en/master/

6.5.2 Mac OS X

- Doxygen https://www.doxygen.nl/download.html
- Sphinx - install using `pip` https://www.sphinx-doc.org/en/master/

6.5.3 Windows

- Doxygen https://www.doxygen.nl/download.html
- Sphinx - install using `pip` https://www.sphinx-doc.org/en/master/

6.6 Supported Build Tools

Anything that doesn't fit the other categories.

CHAPTER

SEVEN

CUSTOMIZING BUILDNIS

CHAPTER

EIGHT

EXTENDING BUILDNIS

CHAPTER

NINE

DEVELOPER REFERENCE

9.1 modules.config package

9.1.1 Submodules

9.1.2 modules.config.build_config module

class BuildCfg(*build_config: str*, *json_path: str*, *load_json: bool = True*)

Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Holds a build tool configuration read from a JSON build tool configuration file.

fromReadJSON()

Converts the *SimpleNamespace* instance read from a JSON file to a BuildCfg instance to use.

writeJSON()

Writes the configuration to file (not used, because it is part of the project configuration JSON file).

classmethod fromReadJSON(*instance: object*) → *buildnis.modules.config.build_config.BuildCfg*

Converts a *SimpleNamespace* instance load from a JSON build tools configuration file to a BuildCfg instance to use.

Parameters **instance** (*object*) – The *Simplenamespace* instance to convert.

Returns The data of the given object as a *BuildCfg* instance.

Return type *BuildCfg*

initAttribs() → None

Initializes the attributes that the build configuration object must have.

initStages() → None

Sets all needed attributes of a stage.

writeJSON(*json_path: str = ''*, *to_ignore: Optional[List[str]] = None*) → None

Writes the generated config to disk.

Not used, because it is part of the project configuration file.

Parameters

- **json_path** (*str, optional*) – The path to the JSON file to write to. Defaults to "", this uses the saved path.
- **to_ignore** (*List[str]*) – The list of attributes to ignore

9.1.3 modules.config.check module

class Check(*os_name: str*, *arch: str*, *user_path: str*, *do_check: bool = True*)
Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Checks if all build tools are present.

Runs all build tool script_paths in *configure_script_paths* in the subdirectory with the name of the OS passed to the constructor.

Each build tool configuration (item of build_tool_cfgs) has the following attributes:

- *name* The name of the build tool
- *name_long* The full name of the build tool
- *version* The version of the build tool, gathered from its output
- *version_arg* The argument to call the build tool with to get the version
- ***version_regex*** **The regex to parse the output of *version_arg* to get** *version*
- *build_tool_exe* The executable's file name
- *install_path* The path to the executable
- *env_script* The environment script to call before using the executable
- *env_script_arg* The argument to call the environment script with
- ***is_checked*** **Has the executable been run and the version output been** parsed?

os_name
the OS we are building for

Type OSName

arch
the CPU architecture we are building for

Type Arch

build_tool_cfgs
the list of build tool configurations returned from the script_paths in *configure_script_paths/OS*

Type list

isBuildToolCfgOK()
checks if the build tool config has the minimum needed attributes

checkVersions()
runs all build tools with the version argument, to check if the executable works

checkVersions() → None
Runs all configured build tools with the 'show version' argument.

To check, if the configured build tools exist and are working, try to execute each with the argument to get the version string of the build tool.

isBuildToolCfgOK(*cfg: Any*) → bool
Checks if the given object has all the needed attributes of a build tool configuration.

Parameters **cfg** (*obj*) – the object to check

Returns

True, if *cfg* has all needed attributes False else

Return type bool

runScript (*script_path: pathlib.Path*) → None
Runs the script at the given path and saves it's output.

Parameters **script_path** (*pathlib.Path*) – The path to the script to run.

runScriptsInDir (*working_dir: pathlib.Path*) → None
Runs all build tool config scripts in the given path.

Parameters **working_dir** (*pathlib.Path*) – The path in which to run the build tool scripts.

searchBuildTool (*name: str*) → object
Searches for a build tool with the given name.

Parameters **name** (*str*) – The name of the build tool to search for.

Returns

The build tool object with the given name on success, *None* if not found or another error occurred.

Return type object

9.1.4 modules.config.config module

class Config (*project_config: str*, *json_path: str*)
Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Loads all JSON configurations.

Parses the project's JSON configuration, all module JSON configurations and all build tool JSON configurations.

Attributes: config_path (FilePath): the path to the project's main JSON configuration. project_cfg_dir (FilePath): The directory part of *config_path* project_cfg (obj): the project's JSON configuration stored in a Python class. module_cfgs (Dict[FilePath, Any]) the module JSON configurations

(mentioned in project_cfg)

build_cfgs (Dict[FilePath, Any]) the build JSON configurations (mentioned in the module JSONs)

_logger (logging.Logger): the logger to use

Methods: parseModuleCfgs: Parses the module JSON configurations setup in the project

JSON

parseBuildCfgs: Parses the build JSON configurations setup in the module JSONs

setHostConfigPath: Sets the path to the generated host config file setBuildToolCfgPath: Sets the path to the generated build tool config file setProjDepCfgPath: Sets the path to the generated project dependency config file getProjCfgDict: Get the project configuration as a JSON sequenceable dict

checkDependencies (*force_check: bool = False*) → None
Calls the *checkDependencies* method of the project dependency configuration.

Parameters **force_check** (*bool, optional*) – if this is *True*, check the dependency even if it has been checked before - if *is_checked* is *True*. Defaults to False.

connectInTarget (*target: object*) → None
Connects the build configuration with the given target.

Parameters **target** (*object*) – The target to search the build config for and connect it.

connectModulesBuildTools() → None
For each target in each module: search for the build tool and put it into this module's target.

expandAllPlaceholders(*parents: Optional[List[object]] = None*) → None
Goes through all configurations and replaces placeholders in their elements. A Placeholder is a string like *${PLACEHOLDER}*, a dollar sign followed by a curly opening brace, the string to replace and the closing curly brace.

Parameters **parents** (*List[object], optional*) – The list of parent objects. Defaults to None.

parseBuildCfgs() → None
Parses the build JSON configurations.

Parses all JSON build configurations in *./build_conf*.

parseModuleCfgs() → None
Parses the module JSON configurations.

Parses and stores all module JSON configurations configured in the project's setup stored in self.project_cfg. Stores the configurations in module_cfgs.

readBuildCfgs() → None
Reads the build configs from the generated JSON configuration. Rereads them if necessary.

readConfigsJSON() → None
Reads all module and build configurations from their JSON files.

static readSingleTarget(*target: object*) → None
Transforms the deserialized JSON object into a *BuildCfg* object. Rereads the original from disk if it has changed.

Parameters **target** (*object*) – The read deserialized build config object.

searchBuildTools(*build_tool_cfg:* buildnis.modules.config.check.Check) → None
Searches for a build tool in all found build tools. Connects it with a build configuration.

Parameters **build_tool_cfg** (Check) – The build tools config to search for build tools.

searchInStage(*build_tool_cfg:* buildnis.modules.config.check.Check, *stage: object*) → None
Searches for a build tool configured in a stage of a build configuration.

Parameters

- **build_tool_cfg** (Check) – The build tool configuration to search in.
- **stage** (*object*) – The stage containing the build tool name to search for.

setBuildToolCfgPath(*path: str*) → None
Sets the path to the generated build tools config JSON file.

Parameters **path** (*FilePath*) – the path to the build tools config file

setHostConfigPath(*path: str*) → None
Sets the path to the generated host config JSON file.

Parameters **path** (*FilePath*) – the path to the host config file

setProjDepCfgPath(*path: str*) → None
Sets the path to the generated project dependency config JSON file.

Parameters **path** (*FilePath*) – the path to the project dependency config file.

setProjectConstants() → None
Sets the global project constants to be replaced by placeholders.

writeJSON(*json_path: str = ''*, *to_ignore=None*) → None
Writes the project's config to a JSON file.

Parameters

- **json_path** (*FilePath, optional*) – The path to the JSON file to write to. Defaults to "", this uses the saved path.
- **to_ignore** (*list, optional*) – List of attributes to ignore, to not save to disk. Defaults to ["project_dep_cfg"].

9.1.5 modules.config.config_dir_json module

class ConfigDirJson(*file_name: str*, *working_dir: str*, *cfg_path: str = ''*)
Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Class to handle the configuration of the directory all generated configuration files are written to.

file_name
The path to the JSON configuration file

Type FilePath

cfg_path
The directory to save generated configurations to

Type FilePath

writeJSON()
Writes the configuration directory configuration to a JSON file with path *file_name*.

writeJSON(*json_path: str = ''*, *to_ignore: Optional[List[str]] = None*) → None
Writes the path to the project config directory to a JSON file with the given path *json_path*.

Parameters **json_path** (*str, optional*) –

The path to the JSON file to write to. Defaults to "", this uses the saved path.

to_ignore (List[str]): The list of attributes to ignore

9.1.6 modules.config.config_files module

class ConfigFiles(*host_cfg: ConfigTuple*, *build_tools_cfg: ConfigTuple*, *project_dep_cfg: ConfigTuple*, *project_cfg: ConfigTuple*)
Bases: tuple

Holds the path and state of all JSON configuration files.

host_cfg
The host configuration, the path to the host configuration JSON file and its status.

Type *ConfigTuple*

build_tools_Cfg
The build tools configuration JSON file path and its status.

Type *ConfigTuple*

project_dep_cfg
The project dependency configuration JSON file path and its status.

Type *ConfigTuple*

project_cfg
THe project configuration JSON file path and its status.

Type *ConfigTuple*

property build_tools_cfg
Alias for field number 1

property host_cfg
Alias for field number 0

property project_cfg
Alias for field number 3

property project_dep_cfg
Alias for field number 2

class ConfigTuple(*path: FilePath = ''*, *exists: bool = False*)
Bases: `tuple`

Class to hold the path of a file and a *bool* that is *True* if the file already exists.

path
The path of the file.

Type FilePath

exists
True`if the file already exists on disk, `False otherwise.

Type bool

property exists
Alias for field number 1

property path
Alias for field number 0

9.1.7 modules.config.config_values module

HOST_CPU_ARCH: str = ''
The host's CPU architecture, like *x64* or *x86*.

HOST_NAME: str = 'localhost'
The host's name.

HOST_NUM_CORES: int = 1
The number of physical cores of the host's CPU.

HOST_NUM_LOG_CORES: int = 1
The number of logical cores (including Hyperthreading) cores of this host's CPU.

HOST_OS: str = ''
The OS we are running on.

PROJECT_AUTHOR: str = ''
The build project author's name(s).

PROJECT_COMPANY: str = ''
The build project company's name.

PROJECT_CONFIG_DIR_PATH: str = ''
The config path of the build project, the argument of *–generated-conf-dir* on the command line.

PROJECT_COPYRIGHT_INFO: str = ''
The build project's copyright info string.

PROJECT_EMAIL: str = ''
The build project's email address.

PROJECT_NAME: str = 'Build Project'
The build project's name.

PROJECT_ROOT: str = './'
The root directory of the project, the directory *project_config.json* is located in.

PROJECT_VERSION: str = ''
The build project's name.

PROJECT_WEB_URL: str = ''
The build project's web URL.

9.1.8 modules.config.configure_build module

configureBuild(*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments, *logger:* *logging.Logger*, *config_dir_config:* buildnis.modules.config.config_dir_json.ConfigDirJson, *host_cfg:* buildnis.modules.config.host.Host, *host_cfg_filename:* *str*, *json_config_files:* buildnis.modules.config.config_files.ConfigFiles) → None
Configures the build.

Parameters

- **commandline_args** (CommandlineArguments) – The object holding the command line arguments.
- **logger** (*logging.Logger*) – The logger instance to use.
- **config_dir_config** (ConfigDirJson) – The object holding the path to the directory, all generated JSON files are written to.
- **host_cfg** (Host) – The host configuration instance.
- **host_cfg_filename** (*FilePath*) – Path to the host configuration JSON file to write.
- **json_config_files** (ConfigFiles) – Holds paths to all JSON configuration files to write.

ifConfigureDeleteProjectJSON(*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArgur
logger: *logging.Logger*, *json_config_files:* buildnis.modules.config.config_files.ConfigFiles) → None
Deletes the project configuration JSON if it exists and *–configure* has been called.

Parameters

- **commandline_args** (CommandlineArguments) – The object holding the command line arguments
- **logger** (*logging.Logger*) – THe logger to use.

- **json_config_files** (ConfigFiles) – The path to the project configuration JSON and whether it exists.

setupProjectCfg(*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments, *json_config_files:* buildnis.modules.config.config_files.ConfigFiles) → *buildnis.modules.config.config.Config*

Sets up the project configuration, including the project dependency configuration.

Parameters

- **commandline_args** (CommandlineArguments) – The object holding all command line arguments.
- **json_config_files** (ConfigFiles) – Holds the path to the project configuration JSON file's path.

Returns The project config containing all values

Return type *Config*

writeBuildTools(*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments, *logger:* *logging.Logger*, *host_cfg:* buildnis.modules.config.host.Host, *json_config_files:* buildnis.modules.config.config_files.ConfigFiles) → *buildnis.modules.config.check.Check*

Writes the build tools configuration to disk.

Parameters

- **commandline_args** (CommandlineArguments) – The object holding all command line arguments.
- **logger** (*logging.Logger*) – The logger to use.
- **host_cfg** (Host) – The host configuration instance.
- **json_config_files** (ConfigFiles) – Holds the Path to the build tools JSON configuration path, the path to write to.

Returns The build tools configuration object to use.

Return type *Check*

writeHostCfg(*host_cfg:* buildnis.modules.config.host.Host, *host_cfg_filename: str*) → None

Write the host configuration JSON file.

Parameters

- **host_cfg** (Host) – The host configuration to save.
- **host_cfg_filename** (*FilePath*) – The path of the JSON file to write to.

writeProjectJSON(*host_cfg_filename: str*, *json_config_files:* buildnis.modules.config.config_files.ConfigFiles, *cfg:* buildnis.modules.config.config.Config) → None

Writes the project configuration JSON to disk.

Parameters

- **host_cfg_filename** (*FilePath*) – Path to the host configuration.
- **json_config_files** (ConfigFiles) – The object holding the path to the project configuration JSON to write.
- **cfg** (Config) – The project configuration instance to write to disk.

9.1.9 modules.config.host module

class Host

Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Holds all information about the host this is running on.

Stores hostname, OS, OS version, CPU architecture, RAM, CPU name, ...

Attributes host_name (str): This host's name os (str): The OS this is running on (like "Windows", "Linux") os_vers_major (str): Major version of the OS (like "10" for Windows 10) os_vers (str): Excact version string cpu_arch (str): CPU architecture, like "x64" or "x86" cpu (str): The detailed name of the CPU file_name (str): The JSON identifier of the host config file, part

of the file name

level2_cache (int): Size of the CPU's level 2 cache, in bytes level3_cache (int): Size of the CPU's level 3 cache, in bytes num_cores (int): The number of physical cores num_logical_cores (int): The number of logical, 'virtual' cores ram_total (int): Amount of physical RAM in bytes gpu List[str]: The list of names of all GPUs python_version (str): The version of this host's Python interpreter json_path(str): The path to the written JSON host config file

Methods

collectWindowsConfig: adds information, that has to be collected in a Windows specific way

collectLinuxConfig: adds information, that has to be collected in a Linux specific way

collectOSXConfig: adds information, that has to be collected in a Mac OS X specific way

GetCPUInfo() → None

Gets the CPU info, like cache sizes, number of cores.

collectLinuxConfig() → None

Collect information about the hardware we're running on on Linux.

Calls the following commands:

cat /etc/os-release NAME="Red Hat Enterprise Linux" VERSION="8.3 (Ootpa)"

grep "model name" /proc/cpuinfo |uniq|cut -d':' -f2 getconf -a|grep LEVEL2_CACHE_SIZE|awk '{print $2}' getconf -a|grep LEVEL3_CACHE_SIZE|awk '{print $2}' grep "cpu cores" /proc/cpuinfo |uniq|cut -d':' -f2 grep "siblings" /proc/cpuinfo |uniq |cut -d':' -f2 free -b|grep "Mem:"|awk '{print $2}' grep "DISTRIB_DESCRIPTION" /etc/lsb-release lspci|grep VGA|cut -f3 -d':'

collectLinuxCpuGpuRam()

Collects information about this host's CPU, GPU, and so on on Linux.

collectOSXConfig() → None

Collect information about the hardware we're running on on MacOS X.

Using this commands: sysctl -n hw.memsize sysctl -n hw.physicalcpu sysctl -n hw.logicalcpu sysctl -n hw.l2cachesize sysctl -n hw.l3cachesize sysctl -n machdep.cpu.brand_string sw_vers -productVersion

TODO get GPU info: system_profiler SPDisplaysDataType

collectWinCpuGpuRam()

Collects the Windows CPU, GPU and RAM size information.

collectWindowsConfig() → None

Collect information about the hardware we're running on on Windows.

Calls these commands and parses their outputs:

wmic cpu get L2CacheSize,L3CacheSize,NumberOfLogicalProcessors, NumberOfCores,Name

wmic memorychip get capacity wmic path win32_VideoController get name

getCPU() → None
Sets the CPU name.

getGPU() → None
Sets the GPU name list.

getGPULspci() → None
Gets the GPU names using */sbin/lspci*.

getGPUNamesLinux() → None
Gets the list of GPU names.

getGPUSbinLspci() → None
Gets the GPU names using *lspci*.

getOSInfo() → None
Gets Info about the OS, like Name, CPU architecture, and similar.

getRAM() → None
Sets the RAM size.

parseCPUInfoLine(*line: str*) → None
Parses the output of the CPU info wmic command.

Parameters **line** (*str*) – The line of output to parse.

parseRAMline(*line: str*) → None
Parses a single line of output to get the RAM size.

Parameters **line** (*str*) – The line of output to parse.

retryCPUInfo(*line: str*) → None
Retry getting the CPU info, this time ignore L2Cache (GitHub Windows runners doesn't display L2 cache).

Parameters **line** (*str*) – The line of output to parse.

setConstants() → None
Set the host constants to use for configuration files.

printHostInfo() → None
To test the collection of the host's information, print all to stdout.

9.1.10 modules.config.host_linux module

getCPUNameLinux() → *buildnis.modules.config.CmdOutput*
Returns the CPU's name.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getGPUNamesLinux() → *buildnis.modules.config.CmdOutput*
Returns the names of the GPUs.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getGPUNamesSbinLinux() → *buildnis.modules.config.CmdOutput*
Returns the names of the GPUs, using */sbin/lspci* because some distributions don't have *lspci* in */usr/bin*

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getL2CacheLinux() → *buildnis.modules.config.CmdOutput*
Returns the size of the CPU's level 2 cache.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getL3CacheLinux() → *buildnis.modules.config.CmdOutput*
Returns the size of the CPU's level 3 cache.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getNumCoresLinux() → *buildnis.modules.config.CmdOutput*
Returns the number of physical cores.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getNumLogCoresLinux() → *buildnis.modules.config.CmdOutput*
Returns the number of logical cores, including the hyperthreading.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getOSMajVers() → *buildnis.modules.config.CmdOutput*
Returns the major OS version.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getOSVer() → *buildnis.modules.config.CmdOutput*
Returns the minor OS version.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getRAMSizeLinux() → *buildnis.modules.config.CmdOutput*
Returns the RAM size in bytes.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

9.1.11 modules.config.host_osx module

getCPUNameOSX() → *buildnis.modules.config.CmdOutput*
Returns the CPU name.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getGPUOSX() → *buildnis.modules.config.CmdOutput*
Return the GPU names.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getL2CacheOSX() → *buildnis.modules.config.CmdOutput*
Returns the size of the CPU's level 2 cache.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getL3CacheOSX() → *buildnis.modules.config.CmdOutput*
Returns the size of the CPU's level 3 cache.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getNumCoresOSX() → *buildnis.modules.config.CmdOutput*
Returns the number of physical cores.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getNumLogCoresOSX() → *buildnis.modules.config.CmdOutput*
Returns the number of logical cores, including hyperthreading.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getOSName() → *buildnis.modules.config.CmdOutput*
Returns the OS version of OS X.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getRAMSizeOSX() → *buildnis.modules.config.CmdOutput*
Returns the RAM size in bytes.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

9.1.12 modules.config.host_windows module

getCPUInfo() → *buildnis.modules.config.CmdOutput*
Gets CPU info using *wmic*.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getCPUName() → *buildnis.modules.config.CmdOutput*
Gets the CPU name using *wmic*

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getGPUInfo() → *buildnis.modules.config.CmdOutput*
Returns the GPU names.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

getMemInfo() → *buildnis.modules.config.CmdOutput*
Returns the RAM size in bytes.

Returns

The output of the command, as a *CmdOutput* instance containing *stdout* and *stderr* as attributes.

Return type *CmdOutput*

9.1.13 modules.config.json_base_class module

class JSONBaseClass(*config_file_name: str*, *config_name: str*)
Bases: `object`

Base class for all objects that read and/or write JSON configuration files. Holds the values of the JSON file in it'S attributes (all except *_logger*, the *loggin.Logger* instance).

hasConfigChangedOnDisk()
Returns *True* if the JSON configuration has changed since the time the checksum has been calculated.

expandAllPlaceholders()
Replaces all placeholders (like *${PLACEHOLDER}*) in the instance's attribute values.

readJSON()
Reads the JSON config file and saves the values to attributes.

reReadIfChangedOnDisk()
Checks if the original read file has changed on disk, and if yes, rereads the file from disk.

writeJSON()
Writes the attributes and their values to the JSON file.

reWriteIfChangedOnDisk()
Checks if the original read file has changed on disk, and if yes, rereads the file from disk and rewrites it to the generated JSON's file path.

addAttributesIfNotExist(*attributes: Dict[str, object]*) → None
Adds each attribute in the given dictionary of attributes.

The key of the dictionary is the attribute's name, the value the deault value to set the attribute to, if it didn't exist.

Parameters **attributes** (`Dict[str, object]`) – The dictionary of attributes and default values

expandAllPlaceholders(*parents: Optional[List[object]] = None*) → None
Goes through all configurations and replaces placeholders in their elements. A Placeholder is a string like *${PLACEHOLDER}*, a dollar sign followed by a curly opening brace, the string to replace and the closing curly brace.

See also: *modules.helpers.config_parser.parseConfigElement*

Parameters **parents** (`List[object]`) – The hierarchical list of parent objects.

hasConfigChangedOnDisk() → bool
Returns *True* if the JSON file has been changed since the time the checksum has been calculated.

Returns

***True* if the original JSON file has changed since the time**

the file's checksum has been saved.

False else

Return type bool

reReadIfChangedOnDisk() → None
Checks if the JSON configuration file has been changed since the time the checksum has been calculated, if yes, it is reread from disk.

reWriteIfChangedOnDisk() → None
Checks if the JSON configuration file has been changed since the time the checksum has been calculated, if yes, it is reread from disk and the generated JSON file is written to it's file path.

readJSON(*json_path: str*) → None
Reads a JSON file and puts it's items into attributes of this object.

Parameters **json_path** (*FilePath*) – The path of the JSON file to load.

writeJSON(*json_path: str*, *to_ignore: Optional[List[str]] = None*) → None
Writes the class instance to the JSON file.

Parameters

- **json_path** (*FilePath*) – The path to the file to write the JSON to
- **to_ignore** (*List[str]*) – The list of attributes to ignore

setAttrIfNotExist(*instance: object*, *attributes: Dict[str, object]*) → None
Check if the given object has each of the given attributes, if not, set it to the given value.

Parameters

- **instance** (*object*) – The instance to check for attributes.
- **attributes** (*Dict[str, object]*) – The dictionary of attributes and attribute values.

9.1.14 modules.config.module module

class ModuleCfg(*module_config: str*, *json_path: str*, *load_json: bool = True*)
Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Holds a single module configuration read from a JSON module configuration file.

fromReadJSON()
Converts the *SimpleNamespace* instance read from a JSON file to a ModuleCfg instance to use.

writeJSON()
Writes the configuration to file (not used, because it is part of the project configuration JSON file).

classmethod fromReadJSON(*instance: object*) → *buildnis.modules.config.module.ModuleCfg*
Converts a *SimpleNamespace* instance load from a JSON module configuration file to a ModuleCfg instance to use.

Parameters **instance** (*object*) – The *Simplenamespace* instance to convert.

Returns The data of the given object as a *ModuleCfg* instance.

Return type *ModuleCfg*

writeJSON(*json_path: str = ''*, *to_ignore: Optional[List[str]] = None*) → None
Writes the generated config to disk.

Not used, because it is part of the project configuration file.

Parameters

- **json_path** (*FilePath, optional*) – The path to the JSON file to write to. Defaults to "", this uses the saved path.
- **to_ignore** (*List[str]*) – The list of attributes to ignore.

9.1.15 modules.config.project_dependency module

class ProjectDependency(*dependency_config: str*, *json_path: str*)

Bases: *buildnis.modules.config.json_base_class.JSONBaseClass*

Parses the project dependency configuration file and checks the dependencies.

Parses the project dependency JSON file and saves the config file to *dependency_cfg*. Checks the dependencies and tries to download and install missing dependencies. Writes the altered configuration to the JSON file.

config_path
Path to the project dependency JSON file

Type FilePath

dependencies
The list of dependencies

Type List[object]

Dependency object attributes

name
the dependency's name

Type str

website_url
website to get information about the dependency

Type str

download_url
website to download the dependency

Type str

download_dir
path to the directory to download the dependency to

Type str

install_cmd
command line to call to install the dependency

Type str

install_arguments
arguments to pass to the install command

Type List[str]

ok_if_exists
if this file exists, the dependency has been successfully installed

Type FilePath

ok_if_executable
if this file is executable, the dependency has been successfully installed

Type FilePath

executable_argument
the argument to call *ok_if_executable* with to test it

Type str

executable_check_regex
the regex to parse the output of *ok_if_executable* with. If a match is found, the dependency has been successfully installed

Type str

is_checked
if this is true, the dependency has been successfully installed

Type bool

checkDependencies()
Checks all dependencies in the list of dependencies *dependencies*, if the dependency is installed.

isDependencyFulfilled()
Returns `True`if the dependency has been installed.

installDep()
Installs the given dependency.

isExecuteableDep()
Checks, if the given dependency's executable works, that is, returns the expected string.

checkDependencies(*force_check: bool = False*) → None
Runs all configured dependency checks.

Checks for each dependency if the configured file exists or the configured executable works. If not, it tries to download and/or install the dependency and tries again.

Parameters **force_check** (*bool, optional*) – if this is *True*, check the dependency even if it has been checked before - if *is_checked* is *True*. Defaults to False.

checkIFInstalled(*dep: object*) → None
Check if the dependency is installed, if not, install it and check again.

Parameters **dep** (*object*) – The dependency object to check.

download(*dep: object*) → None
Download the dependency.

Parameters **dep** (*object*) – The dependency object to download.

install(*dep: object*) → None
Install the dependency.

Parameters **dep** (*object*) – The dependency object to install.

installDep(*dep: object*) → None
Download and/or install the given dependency.

Parameters **dep** (*object*) – the dependency to install or download

isDependencyFulfilled(*dep: object*) → bool
Checks if the given dependency is installed.

Checks if the configured path exist or the configured executable works.

Parameters **dep** (*object*) – the dependency object to check

Returns

***True*, if the dependency has been found** *False* else

Return type bool

isExecuteableDep(*dep: object*) → bool
Execute the dependency, if that works, returns *True*.

Parameters **dep** (*object*) – the dependency to run

Returns

***True* if the executable has been running OK and returns the**

configured string.

False else

Return type bool

okIfExecutable(*dep: object*) → bool
Check if the executable of the object is executable, that is, exists and generates output.

Parameters **dep** (*object*) – The object to check if it'S configured executable is actually executable.

Returns *True*, if the configured executable is callable, *False* else.

Return type bool

okIfExists(*dep: object*) → bool
Check if the dependency is installed by checking the existence of a path.

Parameters **dep** (*object*) – The object to check if it is installed.

Returns *True*, if the path exists, *False* else.

Return type bool

static setMustHaveAttribs(*dep: object*) → None
Sets the attributes a dependency object instance must have.

Parameters **dep** (*object*) – The object to check for must-have attributes.

writeJSON(*json_path: str = ''*, *to_ignore: Optional[List[str]] = None*) → None
Writes the generated dependency configuration to disk.

Parameters

- **json_path** (*FilePath, optional*) – The path to the JSON file to write to. Defaults to "", this uses the saved path.
- **to_ignore** (*List[str]*) – The list of attributes to ignore.

9.1.16 Module contents

Arch
alias of `str`

class CmdOutput(*std_out: str = ''*, *err_out: str = ''*)
Bases: `tuple`

The return type of executed commands.

std_out
the *stdout* output of the executed command

Type str

err_out
the *stderr* output of the executed command

Type str

property err_out
Alias for field number 1

property std_out
Alias for field number 0

class ConfigVersion(*major: str = ''*, *minor: str = ''*)
Bases: `tuple`

The version of a JSON configuration file.

property major
Alias for field number 0

property minor
Alias for field number 1

FilePath
alias of `str`

OSName
alias of `str`

9.2 modules.builds package

9.2.1 Module contents

9.3 modules.helpers package

9.3.1 Submodules

9.3.2 modules.helpers.commandline module

checkCmdLineArgs(*cmd_line_parser: argparse.ArgumentParser*, *cmdline_args: object*) → *buildnis.modules.helpers.commandline_arguments.CommandlineArguments*
Sets all needed attributes to a default value if they aren't present.

Parameters

- **cmd_line_parser** (*argparse.ArgumentParser*) – the *argparse.ArgumentParser* instance to use
- **cmdline_args** (*object*) – the object returned by *cmd_line_parser.parse_args*

Returns the checked and filled *CommandlineArguments* instance

Return type *CommandlineArguments*

parseCommandLine() → *buildnis.modules.helpers.commandline_arguments.CommandlineArguments*

Parses the command line arguments.

Parses the command line arguments, exits the program if an illegal argument has been given.

Parameters **arguments** – the command line arguments passed to the program

Returns An *CommandlineArguments* instance containing the command line arguments as attributes.

9.3.3 modules.helpers.commandline_arguments module

class CommandlineArguments(*src: object*)

Bases: object

Holds information about the command line arguments passed to the program.

Its attributes are the possible command line arguments of the program.

project_config_file

the path to the project config file to use

Type FilePath

conf_dir

the path to the directory to write generated configurations to

Type FilePath

conf_scripts_dir

the path to the directory to search for additional build tool configure scripts.

Type FilePath

log_file

the path to the log file to write.

Type FilePath

log_level

the minimum log level

Type int

do_configure

run only the configure phase of the build

Type bool

do_build

run only the build phase of the build

Type bool

build_targets

list of build targets that should be build

Type List[str]

do_install
only run the install phase of the build

Type bool

install_targets
the list of targets to install

Type List[str]

do_clean
delete all files generated by the build phase

Type bool

do_distclean
delete all generated files (build and configuration)

Type bool

do_check_what_to_do
do everything that has not been done yet.

Type bool

checkTargetArgs (*name: str*) → None
Checks the list stored in the attribute with the given name and flattens it to a single list if it contains another list.

Parameters **name** (*str*) – the name of the attribute to check the stored list of

static doAppendTarget (*tmp_targets: List*, *target: object*) → None
Appends a target to the list of targets.

Parameters

- **tmp_targets** (*List*) – the list of targets to append
- **target** (*object*) – the target to check if it is a list or a single element.

flattenList (*name: str*, *tmp_targets: List*)
Flattens the given list to a single list.

Parameters

- **name** (*str*) – The attribute's name, the list to flatten.
- **tmp_targets** (*List*) – Where to sore the flattened list.

initAttribs (*src: object*) → None
Initializes all attributes to default values if not set from the command-line, that means, that attribute isn't an attribute of *src*.

Parameters **src** (*object*) – The object holding the parsed command-line arguments.

setCleanStages (*src: object*) → None
Set arguments for the clean stages of the build.

Parameters **src** (*object*) – The original object holding the command line arguments.

setConfigs (*src: object*) → None
Set configuration related arguments.

Parameters **src** (*object*) – The original object holding the command line arguments.

setStages (*src: object*) → None
Set arguments for the stages of the build.

Parameters **src** (*object*) – The original object holding the command line arguments.

deleteConfigs (*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments, *logger: logging.Logger*, *list_of_generated_files: List[str]*, *list_of_generated_dirs: List[str]*)

Deletes all configuration files and directories.

Parameters

- **commandline_args** (*object*) – Command line argument object instance
- **logger** (*logging.Logger*) – The logger to use and stop
- **list_of_generated_files** (*List[FilePath]*) – The list of files to delete
- **list_of_generated_dirs** (*List[FilePath]*) – The list of directories to delete. Attention: each directory must be empty!

deleteLogfiles (*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments) → None

Deletes the log file, if one has been configured from the command-line.

Parameters **commandline_args** (CommandlineArguments) – Instance holding the command-line arguments, to check, if a log file has been used.

doDistClean (*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments, *logger: logging.Logger*, *list_of_generated_files: List[str]*, *list_of_generated_dirs: List[str]*) → None

Helper: if argument *distclean* is set, delete all generated files.

WARNING: Shuts down the logging mechanism, no more logging after this function!

Parameters

- **commandline_args** (*object*) – Command line argument object instance
- **logger** (*logging.Logger*) – The logger to use and stop
- **list_of_generated_files** (*List[FilePath]*) – The list of files to delete
- **list_of_generated_dirs** (*List[FilePath]*) – The list of directories to delete. Attention: each directory must be empty!

setupLogger (*commandline_args:* buildnis.modules.helpers.commandline_arguments.CommandlineArguments) → logging.Logger

Sets up the logger.

Parameters **commandline_args** (CommandlineArguments) – The object holding the command line arguments.

Returns The commandline object instance to use.

Return type logging.Logger

9.3.4 modules.helpers.config_parser module

expandItem(*item: str*, *parents: List[object]*) → object
Parses the given item, if it contains a placeholder, that placeholder is expanded. If the item doesn't contain a placeholder, the item's unaltered string is returned.

Parameters

- **item** (*str*) – The item to parse and expand its placeholder
- **parents** (*List[object]*) – The parents of the item to search for the placeholder's content.

Returns

The expanded string if the item contained a placeholder, the original string else. If the placeholder points to another object that is not a string, this object is returned.

Return type object

getPlaceholder(*parents: List[object]*, *parent_to_use_id: int*, *placeholder: str*) → object

Returns the expanded placeholder. Searches for the attribute with name *placeholder* or the value of the key *placeholder* in the parent.

Parameters

- **parents** (*List[object]*) – The list of parents to search the expansion in.
- **parent_to_use_id** (*int*) – The id of the parent in the list, to use for the replacement.
- **placeholder** (*str*) – The string to replace with an element of the same name

Raises Exception – if the replacement of the placeholder from the parent element throws an exception.

Returns The replacement for the placeholder.

Return type object

parseConfigElement(*element: object*, *parents: Optional[List[object]] = None*) → object
Parses the given config element and replaces placeholders. Placeholders are strings of the form *${PLACEHOLDER}*, with start with a dollar sign followed by an opening curly brace and end with a curly brace. The string between the two curly braces is changed against it's value.

Parameters

- **element** (*object*) – The configuration element to parse and expand.
- **parent** (*List[object], optional*) – The parent and the parent's parent and it's
- **as a list** (*parent*) –
- **with the parent as first element. Defaults to None.** (*starting*) –

Returns The parsed and expanded object.

Return type object

parseList(*element: List[object]*, *local_parents: List[object]*) → List[object]
Parses the items of a list.

Parameters

- **element** (*List[object]*) – The list to parse

- **local_parents** (*List[object]*) – The list of parents

Returns The parsed and, if applicable, expanded, list of items.

Return type List[object]

9.3.5 modules.helpers.execute module

class EnvArgs(*script: FilePath = ''*, *args: List[str] = None*, *do_source: bool = False*)
Bases: `tuple`

Class to hold the arguments needed for the environment script of a command to run.

script
The path to the environment script to run or source.

Type FilePath

args
The arguments to pass to the environment script.

Type List[str]

do_source
If this is true, the environment script is sourced in the current command interpreter and not executed.

Type bool

property args
Alias for field number 1

property do_source
Alias for field number 2

property script
Alias for field number 0

class ExeArgs(*exe: FilePath = ''*, *args: List[str] = None*)
Bases: `tuple`

Class to hold the arguments needed to run a command.

exe
The path to the executable to call.

Type FilePath

args
The list of arguments to pass to the executable.

Type List[str]

property args
Alias for field number 1

property exe
Alias for field number 0

exception ExecuteException
Bases: `Exception`

The Exception is thrown if the execution of the given commandline fails.

class RunRegex(*regex: str = ''*, *group: int = 0*)
Bases: tuple

Class to hold a regex tuple to parse a command output with.

regex
The actual regex to use for parsing the output of the command.

Type str

group
The match group of the regex to use for the result.

Type int

property group
Alias for field number 1

property regex
Alias for field number 0

doesExecutableWork(*exe_args: buildnis.modules.helpers.execute.ExeArgs*, *check_regex: buildnis.modules.helpers.execute.RunRegex*, *env_args: buildnis.modules.helpers.execute.EnvArgs = EnvArgs(script='', args=None, do_source=False)*) → str
Checks if the given command line works.

Tries to run the command with the given arguments (see *runCommand*) and parses the output of the program, tries to match the given regex *check_regex* in the output (*stdout* and *stderr*) of the command. If the match group *regex_group* (defaults to 0, the whole regex) is found in the output, the function returns the matched string.

Parameters

- **exe_args** (ExeArgs) – The name of the executable or path to the executable to call and the arguments to pass to the executable.
- **env_args** (EnvArgs) – The arguments needed for the environment script, if applicable. Holds the command to call the environment script in *env_args.script*, the arguments to call the environment script with in *env_args.args* and if the environment script has to be sourced instead of running it, *env_args.do_source* ir *True*.
- **env_args** – (EnvArgs): The arguments needed to setup the environment for the executable, if applicable. Defaults to (“”, None, False).

Raises ***ExecuteException*** – if something goes wrong

Returns

the matched string if the regex matches the output, the empty string ” otherwise.

Return type str

runCommand(*exe_args: buildnis.modules.helpers.execute.ExeArgs*, *env_args: buildnis.modules.helpers.execute.EnvArgs = EnvArgs(script='', args=None, do_source=False)*) → *buildnis.modules.config.CmdOutput*
Executes the given command with the given arguments.

The argument *exe_args.exe* is the executable’s name or path, needed arguments can be passed in the list *exe_args.args*. In *env_args* the command to set up an environment can be given, with needed arguments to this environment script in the list *env_args.args*.

Parameters

- **exe_args** (ExeArgs) – The name of the executable or path to the executable to call and the arguments to pass to the executable.

- **env_args** (EnvArgs) – The arguments needed for the environment script, if applicable. Holds the command to call the environment script in *env_args.script*, the arguments to call the environment script with in *env_args.args* and if the environment script has to be sourced instead of running it, *env_args.do_source* ir *True*.

Raises ***ExecuteException*** – if something goes wrong

Returns The output of the executed command as tuple (stdout, stderr)

Return type *CmdOutput*

setEnv(*exe_args:* buildnis.modules.helpers.execute.ExeArgs, *env_args:* buildnis.modules.helpers.execute.EnvArgs, *exe_args_real: List[str]*, *env_args_real: List[str]*, *cmd_line_args: List[str]*) → None

Set up the command line arguments for the environment script.

Parameters

- **exe_args** (ExeArgs) – The object holding the needed arguments for the executable itself.
- **env_args** (EnvArgs) – The object holding the needed arguments to set the environment for the executable.
- **exe_args_real** (*List[str]*) – The real list of executable arguments to use.
- **env_args_real** (*List[str]*) – The real list of environment arguments to use.
- **cmd_line_args** (*List[str]*) – The list of arguments to pass to the command interpreter.

9.3.6 modules.helpers.file_compare module

class FileCompare(*file: str*)

Bases: object

Holds information about a file to compare it to another version of itself to see if something has changed.

Does the following to steps:

- compares the file sizes, if they are the same
- compares the Blake2 hashes of both files

path

path to the file as string

Type FilePath

path_obj

the *Path* object of the file

Type pathlib.Path

size

the size of the file in bytes (symlinks don't have a meaningful size)

Type int

hash

the BLAKE2 hash of the file's contend as a hex string.

Type str

isSame (*bool*)
returns true if *self* 'is' the same as the given file.

generateHash (*str*)
generates the hash of the file, saves it in *hash* and returns it

generateHash () → str
Rehashes the file, saves and returns the hash as hex string.

The hash replaces the old one in the attribute *hash*.

Raises ***FileCompareException*** – if something goes wrong

Returns The hash of the file with path *path* as hex string.

Return type str

hasChanged (*not_exist_is_excp: bool = False*) → bool
Checks if the stored file has changed on disk since taking the last checksum.

If the file has changed (another file size or checksum) or it doesn't exist any more, *True* is returned. If the file still has the same checksum as the stored one, *False* is returned.

Parameters **not_exist_is_excp** (*bool, optional*) – Should an exception be raised if the file doesn't exist anymore? Defaults to False.

Raises ***FileCompareException*** – if something went wrong

Returns

***True*, if the file has changed since calculating the checksum,** *False* else.

Return type bool

isSame (*file: str, not_exist_is_excp: bool = False*) → bool
Checks whether *self* and the file with path *file* are the same.

Problem: a symlink to the file and the file itself are NOT the same using this.

Parameters

- **file** (*FilePath*) – path to the file to check
- **not_exist_is_excp** (*bool*) – if this is *True* an exception is raised if *file* does not exist. If this is *False*, *False* is returned - the files data is not the same. Default: *False*

Raises ***FileCompareException*** – if something goes wrong

Returns

***True*, if the file is the same** *False* else

Return type bool

isSameFile (*other:* buildnis.modules.helpers.file_compare.FileCompare) → bool
Checks whether two *FileComare* instances hold the same file content.

Compares *self* against another FileCompare instance, comparing file size and hash.

Returns *True* if both files have the same content.

Attention: does NOT compare the filenames, only file size and hash of the files are compared.

Parameters **other** (FileCompare) – the instance to compare *self* to

Raises ***FileCompareException*** – if something doesn't work out well ....

Returns

***True*, if both instances have the same filesize and hash.** *False* else

Return type bool

areHashesSame (*file1: str*, *file2: str*, *not_exist_is_excp: bool = False*) → bool
Compares the BLAKE2 hashes of the given files.

Returns *True* if the contents of both files are the same (have the same hash).

Parameters

- **file1** (*FilePath*) – first file to compare
- **file2** (*FilePath*) – second file to compare
- **not_exist_is_excp** (*bool*) – if this is *True* an exception is raised if a file does not exist. If this is *False*, *False* is returned - the files hashes are not the same. Default: *False*

Raises ***FileCompareException*** – if something goes wrong

Returns

***True*, if both files' content have the same BLAKE2 hash value.** *False* else

Return type bool

9.3.7 modules.helpers.files module

exception FileCompareException
Bases: buildnis.modules.BuildnisException

Exception raised if a given file path can't be accessed or read to generate the hash.

checkIfExists (*file: str*) → bool
Returns *True* if the given file exists.

Parameters **file** (*FilePath*) – Path to the file to test

Raises ***FileCompareException*** – if something went wrong

Returns

***True*`, if the file exists** *False* else

Return type bool

checkIfIsDir (*directory: str*) → bool
Returns *True* if the given file exists and is a directory.

Parameters **directory** (*FilePath*) – Path to the directory to test

Raises ***FileCompareException*** – if something went wrong

Returns

***True*`, if the file exists and is a directory** *False* else

Return type bool

checkIfIsFile (*file: str*) → bool
Returns *True* if the given file exists and is a file.

Parameters **file** (*FilePath*) – Path to the file to test

Raises ***FileCompareException*** – if something went wrong

Returns

***True*`, if the file exists and is a file** *False* else

Return type bool

checkIfIsLink (*link: str*) → bool
Returns *True* if the given file exists and is a symlink.

Parameters **link** (*FilePath*) – Path to the file to test

Raises ***FileCompareException*** – if something went wrong

Returns

***True*`, if the file exists and is a symlink** *False* else

Return type bool

deleteDirs (*logger: logging.Logger*, *list_of_dirs: List[str]*) → None
Deletes all directories in the given list of directories.

Attention: directory has to be empty!

Raises ***FileCompareException*** – if something goes wrong

Parameters

- **logger** (*logging.Logger*) – The logger instance to use for logging.
- **list_of_dirs** (*List[FilePath]*) – The list of directories to delete. As a list of paths to the directories to delete.

deleteFiles (*logger*, *list_of_files: List[str]*) → None
Deletes all files in the given list of files to delete.

Raises ***FileCompareException*** – if something goes wrong

Parameters

- **logger** (*[type]*) – The logger instance to use for logging.
- **list_of_files** (*List[FilePath]*) – The list of files to delete. As a list of file paths to the files to delete.

hashFile (*file: str*) → str
Generates a BLAKE2 hash of the file with the given path.

Returns the hash as a hex string. If something goes wrong, it returns an *FileCompareException* instance.

Raises ***FileCompareException*** – if something goes wrong

Parameters **file** (*FilePath*) – the file to return the BLAKE2 hash of

Returns the hex hash of the file's contend

Return type str

makeDirIfNotExists (*directory: str*) → None
Creates the directory *directory* if it doesn't exist yet.

Parameters **directory** (*FilePath*) – the directory to create

Raises ***FileCompareException*** – if something goes wrong

returnExistingFile (*file_list: List[str]*) → str
Returns the first existing path in the list of given paths, and "" the empty string, if none of the paths points to an existing file.

Raises ***FileCompareException*** – if something goes wrong

Parameters **file_list** (*List[FilePath]*) – The list of file paths to check for existence.

Returns

The first of the given file paths that exists as a file, the empty string ("") if none exists.

Return type FilePath

9.3.8 modules.helpers.json module

checkConfigName(*json_path: str*, *conf_file_name: str*, *ret_val: object*) → None
Check the name of the JSON configuration file.

Parameters

- **json_path** (*FilePath*) – The path of the JSOn file to check.
- **conf_file_name** (*str*) – The configuration name of the JSON file.
- **ret_val** (*object*) – The deserialized JSON file.

checkConfigVersion(*json_path: str*, *ret_val: object*) → None
Check the version of the JSON configuration file.

Parameters

- **json_path** (*FilePath*) – The path to the JSON file.
- **ret_val** (*object*) – The deserialized JSON configuration.

getJSONDict(*src: object*, *to_ignore: Optional[List[str]] = None*) → Dict
Returns a dictionary suitable to pass to *json.dump(s)*.

Attention: only works with simple classes obtained from *json.load(s)*.

Parameters

- **src** (*object*) – The class to serialize.
- **to_ignore** (*List[str]*) – The list of attribute names to ignore.

Returns The dictionary suitable to pass to *json.dump(s)*.

Return type Dict

parseItem(*src: object*, *to_ignore: List[str]*, *ret_val: Dict[str, object]*, *item: object*) → None
Parses an item of *src*'s dictionary of attributes.

Parameters

- **src** (*object*) – The object to serialize.
- **to_ignore** (*List[str]*) – The list of attributes to ignore and not serialize.
- **ret_val** (*Dict[str, object]*) – The dictionary to return, the serialized object src.
- **item** (*object*) – The current item to serialize.

parseList(*src*, *ret_val*, *item*) → None
Parse the elements of a list.

Parameters

- **src** (*object*) – The object to serialize.
- **ret_val** (*Dict[str, object]*) – The dictionary to return, the serialized object src.
- **item** (*object*) – The current item to serialize.

readJSON (*json_path: str*, *file_text: str = ''*, *conf_file_name: str = ''*) → object
Reads the JSON from the given file and saves it to a class object with the JSON elements as attributes.

If an error occurs, the program is exited with an error message! The JSON must have an element *file_version* that has a value of at least *CFG_VERSION*, if not, the program is exited with an error message.

Parameters

- **json_path** (*FilePath*) – The path to the JSON file to read.
- **file_text** (*str, optional*) – The name of the JSON configuration file for logging proposes. Defaults to "", which will be logged as 'a', like in "Writing _a_ JSON configuration file ...".
- **conf_file_name** (*str, optional*) – The string that has to be the value of *file_name* in the JSON file, if not, the program exits. Defaults to "".

Returns A class instance with the JSON elements as attributes.

Return type object

setFileCompare (*src: object*, *ret_val: Dict[str, object]*, *item: object*) → None
Set the attributes of the *FileCompare* instance in the JSON dictionary.

Parameters

- **src** (*object*) – The object to serialize.
- **ret_val** (*Dict[str, object]*) – The dictionary to return, the serialized object src.
- **item** (*object*) – The current item to serialize.

setOrigFile (*json_path: str*, *ret_val: object*) → None
Set the *FileCompare* instance *orig_file* to the original JSON configuration.

Parameters

- **json_path** (*FilePath*) – The path to the JSON configuration file.
- **ret_val** (*object*) – The deserialized JSON file.

writeJSON (*json_dict: Dict*, *json_path: str*, *file_text: str = ''*, *conf_file_name: str = ''*) → None
Writes the information contained in the dictionary *json_dict* as JSON.

If an error occurs, the program is exited with an error message!

Parameters

- **json_dict** (*Dict*) – The JSON serializeable dict to generate the JSON of
- **json_path** (*FilePath*) – Path to the JSON file to write
- **file_text** (*str, optional*) – The name of the JSON configuration file for logging proposes. Defaults to "", which will be logged as 'a', like in "Writing _a_ JSON configuration file ...".
- **conf_file_name** (*str, optional*) – The string that has to be the value of *file_name* in the JSON file, if not, the program exits. Defaults to "".

9.3.9 modules.helpers.logging module

getProgramLogger (*level: int*, *logfile: str*) → logging.Logger
Returns the logger to use for the program.

Always logs *DEBUG*, *INFO* and *WARNING* to *stdout*, *ERROR* and *CRITICAL* go to *stderr*. If a *logfile* is given, everything is logged to this file too. *level* is the minimum log level to actually output messages.

When you want to use this logger, simply call

my_logger = modules.helpers.logging.getProgramLogger()

my_logger.info ("Info level log") my_logger.error ("Error level log")

Parameters

- **level** (*int*) – minimumm log level to output
- **logfile** (*FilePath*) – if this is not *None* or the empty string *""*, log to this file too.

Returns the *logging.Logger* instance to use to log

Return type logging.Logger

9.3.10 modules.helpers.placeholder_regex module

current_date_regex = re.compile('\\$\\{(DATE)\\}')
Regex to find the placeholder *${DATE}*.

current_day_regex = re.compile('\\$\\{(DAY)}')
Regex to find the placeholder *${DAY}*.

current_month_regex = re.compile('\\$\\{(MONTH)}')
Regex to find the placeholder *${MONTH}*.

current_time_regex = re.compile('\\$\\{(TIME)\\}')
Regex to find the placeholder *${TIME}*.

current_year_regex = re.compile('\\$\\{(YEAR)}')
Regex to find the placeholder *${YEAR}*.

host_cpu_arch_regex = re.compile('\\$\\{(HOST_CPU_ARCH)\\}')
Regex to find the placeholder *${HOST_CPU_ARCH}*.

host_name_regex = re.compile('\\$\\{(HOST_NAME)\\}')
Regex to find the placeholder *${HOST_NAME}*.

host_num_cores_regex = re.compile('\\$\\{(HOST_NUM_CORES)\\}')
Regex to find the placeholder *${HOST_NUM_CORES}*.

host_num_log_cores_regex = re.compile('\\$\\{(HOST_NUM_LOG_CORES)\\}')
Regex to find the placeholder *${HOST_NUM_LOG_CORES}*.

host_os_regex = re.compile('\\$\\{(HOST_OS)\\}')
Regex to find the placeholder *${HOST_OS}*.

os_name_linux_regex = re.compile('\\$\\{(OS_NAME_LINUX)\\}')
Regex to find the placeholder *${OS_NAME_LINUX}*.

os_name_osx_regex = re.compile('\\$\\{(OS_NAME_OSX)\\}')
Regex to find the placeholder *${OS_NAME_OSX}*.

os_name_windows_regex = re.compile('\\$\\{(OS_NAME_WINDOWS)\\}')
Regex to find the placeholder *${OS_NAME_WINDOWS}*.

placeholder_regex = re.compile('\\$\\{(.*)\\}')
Regex to find general placefolders, of the form *${STRING}*, where string is to be substituted for a value of another configuration item.

project_author_regex = re.compile('\\$\\{(PROJECT_AUTHOR)\\}')
Regex to find the placeholder *${PROJECT_AUTHOR}*.

project_cfg_dir_regex = re.compile('\\$\\{(PROJECT_CONFIG_DIR_PATH)\\}')
Regex to find the placeholder *${PROJECT_CONFIG_DIR_PATH}*.

project_company_regex = re.compile('\\$\\{(PROJECT_COMPANY)\\}')
Regex to find the placeholder *${PROJECT_COMPANY}*.

project_copyright_info_regex = re.compile('\\$\\{(PROJECT_COPYRIGHT_INFO)\\}')
Regex to find the placeholder *${PROJECT_COPYRIGHT_INFO}*.

project_email_regex = re.compile('\\$\\{(PROJECT_EMAIL)\\}')
Regex to find the placeholder *${PROJECT_EMAIL}*.

project_name_regex = re.compile('\\$\\{(PROJECT_NAME)\\}')
Regex to find the placeholder *${PROJECT_NAME}*.

project_root_regex = re.compile('\\$\\{(PROJECT_ROOT)\\}')
Regex to find the placeholder *${PROJECT_ROOT}*.

project_version_regex = re.compile('\\$\\{(PROJECT_VERSION)\\}')
Regex to find the placeholder *${PROJECT_VERSION}*.

project_web_url_regex = re.compile('\\$\\{(PROJECT_WEB_URL)\\}')
Regex to find the placeholder *${PROJECT_WEB_URL}*.

replaceConstants(*item: str*) → str
Replaces all known constants defined in *config_values.py* in the given string.

These are placeholders like *${PROJECT_ROOT}*, *${PROJECT_NAME}*, ...

Parameters **item** (*str*) – The string to parse for known constants.

Returns

The substitution if a placeholder has been found, the unaltered string else.

Return type str

replaceDateTimeConstants(*ret_val: str*) → str
Replaces date and time placeholders with the current date and/or time.

Parameters **ret_val** (*str*) – The string to parse.

Returns

The replaced date and or time if a match has been found, the original string *ret_val* else.

Return type str

replaceHostConstants(*ret_val: str*) → str
Replaces host placeholders with the corresponding values.

Parameters **ret_val** (*str*) – The string to parse.

Returns

The replaced values if a match has been found, the original string *ret_val* else.

Return type str

replaceProjectConstants(*ret_val: str*) → str
Replaces all project constant placeholders.

Parameters **ret_val** (*str*) – The string to parse for placeholders.

Returns

If a project placeholder has been found, the replaced value, the original string *ret_val* else.

Return type str

9.3.11 modules.helpers.web module

exception WebException
Bases: Exception

Exception that is raised from functions in this module.

doDownload(*url: str*, *to: str = ''*, *use_proxy: bool = False*) → None
Download data from the given URL to the given path.

Parameters

- **url** (*str*) – The URL to download from
- **to** (*str, optional*) – The path to save the download to. Defaults to "".
- **use_proxy** (*bool, optional*) – Should a proxy be used. Defaults to False.

9.3.12 Module contents

So you want to help developing Buildnis?

CHAPTER

TEN

INDEX

This is a placeholder to include genindex.html in the toctree.

CHAPTER

ELEVEN

PYTHON MODULE INDEX

This is a placeholder to include py-modindex.html in the toctree.

PYTHON MODULE INDEX

b

INDEX

D

E

F

G

H

I

J

L

M

N

O

P

R

S

W