benchbuild

Release 3.4.2.dev3+g0712eee

Oct 30, 2018

Contents

CHAPTER 1

Core packages

1.1 benchbuild.experiment module

Experiments

An experiment in benchbuild is a simple list of actions that need to be executed on every project that is part of the experiment. Every *callable* can serve as an action.

However, benchbuild provides many predefined actions that can be reused by implementations in the *benchbuild.utils.actions* module. Furthermore, if you do not need to control the default order of actions for a run-time or a compile-time experiment you can reuse the *Experiment.default_runtime_actions* or *Experiment.default_compiletime_actions*.

Besides the list of actions, it is also the responsibility of an experiment to configure each single project that should take part in an experiment. This includes setting appropriate *CFLAGS*, *LDFLAGS* and any additional metadata that has to be added to binary runs for later evaluation.

Example

‘‘‘python class HelloExperiment(Experiment):

pass

‘‘‘

class benchbuild.experiment.**Configuration**(*project=None*, *config=None*)

Bases: object

Build a set of experiment actions out of a list of configurations.

class benchbuild.experiment.**Experiment**(*name=NOTHING*, *projects=NOTHING*, *id=NOTHING*, *schema=NOTHING*)

Bases: object

A series of commands executed on a project that form an experiment.

The default implementation should provide a sane environment for all derivates.

One important task executed by the basic implementation is setting up the default set of projects that belong to this project. As every project gets registered in the ProjectFactory, the experiment gets a list of experiment names that work as a filter.

name
str – The name of the experiment, defaults to NAME

projects
`list` of *benchbuild.project.Project* – A list of projects that is assigned to this experiment.

id
str – A uuid encoded as `str` used to identify this instance of experiment. Equivalent to the *experiment_group* in the database scheme.

NAME = None

SCHEMA = None

actions()

actions_for_project(*project*)
Get the actions a project wants to run.

Parameters **project** (*benchbuild.Project*) – the project we want to run.

static default_compiletime_actions(*project*)
Return a series of actions for a compile time experiment.

default_id()

static default_runtime_actions(*project*)
Return a series of actions for a run time experiment.

default_schema()

validate_id(*_*, *new_id*)

validate_schema(*_*, *new_schema*)

class benchbuild.experiment.**ExperimentRegistry**(*name*, *bases*, *_dict*)
Bases: `type`

Registry for benchbuild experiments.

experiments = {'empty': <class 'benchbuild.experiments.empty.Empty'>, 'no-measurement

1.2 benchbuild.project module

Project

A project in benchbuild is an abstract representation of a software system that can live in various stages throughout an experiment. It defines two extension points for an experiment to attach on, the compile-time phase and the (optional) run-time phase.

An experiment can intercept the compilation phase of a project and perform any experiment that requires the source artifacts as input.

Furthermore, it is possible to intercept a project's run-time pahse with a measurement.

The project definition ensures that all experiments run through the same series of commands in both phases and that all experiments run inside a separate build directory in isolation of one another.

class benchbuild.project.**Project**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: `object`

Abstract class for benchbuild projects.

A project is an arbitrary software system usable by benchbuild in experiments. Subclasses of Project are registered automatically by benchbuild, if imported in the same interpreter session. For this to happen, you must list the in the settings under plugins -> projects.

A project implementation *must* provide the following method: compile: Downloads & Compiles the source.

A project implementation *may* provide the following functions:

run_tests: Wrap any binary that has to be run under the runtime_extension wrapper and execute an implementation defined set of run-time tests. Defaults to a call of a binary with the name *run_f* in the build directory without arguments.

clean: Clean the project's build directory. Defaults to recursive 'rm' on the build directory and can be disabled by setting the environment variable `BB_CLEAN=false`.

Raises

- `AttributeError` – Class definition raises an attribute error, if the implementation does not provide a value for the attributes *NAME*, *DOMAIN*, and *GROUP*
- `TypeError` – Validation of properties may throw a TypeError.

experiment
benchbuild.experiment.Experiment – The experiment this project is assigned to.

name
str, optional – The name of this project. Defaults to *NAME*.

domain
str, optional – The application domain of this project. Defaults to *DOMAIN*.

group
str, optional – The group this project belongs to. Defaults to *GROUP*.

src_file
str, optional – A main src_file this project is assigned to. Defaults to *SRC_FILE*

container
benchbuild.utils.container.Container, optional – A uchroot compatible container that we can use for this project. Defaults to *benchbuild.utils.container.Gentoo*.

version
str, optional – A version information for this project. Defaults to *VERSION*.

builddir
str, optional – The build directory for this project. Auto generated, if not set.

testdir
str, optional – The location of any additional test-files for this project, usually stored out of tree. Auto generated, if not set. Usually a project implementation will define this itself.

cflags
list of str, optional – A list of cflags used, for compilation of this project.

ldflags
list of str, optional – A list of ldflags used, for compilation of this project.

run_f
str, optional – A filename that points to the binary we want to track. Usually a project implementation will define this itself.

run_uuid
uuid.UUID, optional – An UUID that identifies all binaries executed by a single run of this project. In the database schema this is named the 'run_group'.

compiler_extension
Callable[str, iterable[str], RunInfo], optional – A composable extension that will be used in place of the real compiler. Defaults to running the compiler with a timeout command wrapped around it.

runtime_extension
Callable[str, iterable[str], RunInfo], optional – A composable extension that will be used in place of any binary this project wants to execute. Which binaries to replace is defined by the implementation using *benchbuild.utils.wrapping.wrap*. Defaults to None.

CONTAINER = None

DOMAIN = None

GROUP = None

NAME = None

SRC_FILE = None

VERSION = None

clean()
Clean the project build directory.

clone()
Create a deepcopy of ourself.

compile()
Compile the project.

download(*version=None*)
Auto-generated by with_* decorators.

id

prepare()
Prepare the build diretory.

redirect()
Redirect execution to a containerized benchbuild instance.

run()
Run the tests of this project.

This method initializes the default environment and takes care of cleaning up the mess we made, after a successfull run.

Parameters **experiment** – The experiment we run this project under

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

class benchbuild.project.**ProjectDecorator**(*name*, *bases*, *attrs*)
Bases: *benchbuild.project.ProjectRegistry*

Decorate the interface of a project with the in_builddir decorator.

This is just a small safety net for benchbuild users, because we make sure to run every project method in the project's build directory.

decorated_methods = ['redirect', 'compile', 'run_tests']

class benchbuild.project.**ProjectRegistry**(*name*, *bases*, *attrs*)
Bases: type

Registry for benchbuild projects.

projects = Trie((('gramschmidt/polybench', <class 'benchbuild.projects.polybench.polyb

benchbuild.project.**populate**(*projects_to_filter=None*, *group=None*)
Populate the list of projects that belong to this experiment.

Parameters

- **projects_to_filter** (*list(Project)*) – List of projects we want to assign to this experiment. We intersect the list of projects with the list of supported projects to get the list of projects that belong to this experiment.
- **group** (*list(str)*) – In addition to the project filter, we provide a way to filter whole groups.

1.3 benchbuild.extensions package

1.3.1 Submodules

benchbuild.extensions.base module

Extension base-classes for compile-time and run-time experiments.

class benchbuild.extensions.base.**Extension**(**extensions*, *config=None*, ***kwargs*)
Bases: object

An experiment functor to implement composable experiments.

An experiment extension is always callable with an arbitrary amount of arguments. The varargs component of an extension's *__call__* operator is fed the binary command that we currently execute and all arguments to the binary. Any customization necessary for the extension (e.g, dynamic configuration options) has to be passed by keyword argument.

Parameters

- ***extensions** – Variable length list of child extensions we manage.

- **config** (dict, optional) – Dictionary of name value pairs to be stored for this extension.

next_extensions
Variable length list of child extensions we manage.

config
dict, optional – Dictionary of name value pairs to be stored for this extension.

call_next(**args*, ***kwargs*) → List[benchbuild.utils.run.RunInfo]
Call all child extensions with the given arguments.

This calls all child extensions and collects the results for our own parent. Use this to control the execution of your nested extensions from your own extension.

Returns

A list of collected results of our child extensions.

Return type list of RunInfo

print(*indent=0*)
Print a structural view of the registered extensions.

benchbuild.extensions.compiler module

class benchbuild.extensions.compiler.**RunCompiler**(*project*, *experiment*, **extensions*, *config=None*)
Bases: *benchbuild.extensions.base.Extension*

Default extension for compiler execution.

This extension silences a few warnings, e.g., unused-arguments and handles database tracking for compiler commands. It is used as the default action for compiler execution.

benchbuild.extensions.log module

class benchbuild.extensions.log.**LogAdditionals**(**extensions*, *config=None*, ***kwargs*)
Bases: *benchbuild.extensions.base.Extension*

Log any additional log files that were registered.

class benchbuild.extensions.log.**LogTrackingMixin**
Bases: object

Add log-registering capabilities to extensions.

add_log(*path*)
Add a log to the tracked list.

Parameters **path** (*str*) – Filename of a new log we want to track.

logs
Return list of tracked logs.

benchbuild.extensions.run module

class benchbuild.extensions.run.**Rerun**(**extensions*, *config=None*, ***kwargs*)
Bases: *benchbuild.extensions.base.Extension*

class benchbuild.extensions.run.**RuntimeExtension**(*project*, *experiment*, **extensions*, *config=None*)

Bases: *benchbuild.extensions.base.Extension*

Default extension to execute and track a binary.

This can be used for runtime experiments to have a controlled, tracked execution of a wrapped binary.

class benchbuild.extensions.run.**SetThreadLimit**(**extensions*, *config=None*, ***kwargs*)

Bases: *benchbuild.extensions.base.Extension*

Sets the OpenMP thread limit, based on your settings.

This extension uses the 'jobs' settings and controls the environment variable OMP_NUM_THREADS.

class benchbuild.extensions.run.**WithTimeout**(**extensions*, *limit='10m'*, ***kwargs*)

Bases: *benchbuild.extensions.base.Extension*

Guard a binary with a timeout.

This wraps a any binary with a call to *timeout* and sets the limit to a given value on extension construction.

benchbuild.extensions.time module

class benchbuild.extensions.time.**RunWithTime**(**extensions*, *config=None*, ***kwargs*)

Bases: *benchbuild.extensions.base.Extension*

Wrap a command with time and store the timings in the database.

benchbuild.extensions.time.**fetch_time_output**(*marker*, *format_s*, *ins*)

Fetch the output /usr/bin/time from a.

Parameters

- **marker** – The marker that limits the time output
- **format_s** – The format string used to parse the timings
- **ins** – A list of lines we look for the output.

Returns A list of timing tuples

1.4 benchbuild.utils.settings module

Configuration utilities.

Settings are stored in a dictionary-like configuration object. All settings are modifiable by environment variables that encode the path in the dictionary tree.

Inner nodes in the dictionary tree can be any dictionary. A leaf node in the dictionary tree is represented by an inner node that contains a value key.

class benchbuild.utils.settings.**ConfigDumper**(*stream*, *default_style=None*, *default_flow_style=None*, *canonical=None*, *indent=None*, *width=None*, *allow_unicode=None*, *line_break=None*, *encoding=None*, *explicit_start=None*, *explicit_end=None*, *version=None*, *tags=None*)

Bases: yaml.dumper.Dumper

Avoid polluting yaml's namespace with our modifications.

```
yaml_implicit_resolvers = {'':  [('tag:yaml.org,2002:null', re.compile('^(?:  ~\n |nul
```

```
>>> p = ConfigPath([]); str(p)
'/'
```

static path_to_str(*components*)

validate()
Make sure this configuration path exists.

class benchbuild.utils.settings.**Configuration**(*parent_key*, *node=None*, *parent=None*, *init=True*)

Bases: object

Dictionary-like data structure to contain all configuration variables.

This serves as a configuration dictionary throughout benchbuild. You can use it to access all configuration options that are available. Whenever the structure is updated with a new subtree, all variables defined in the new subtree are updated from the environment.

Environment variables are generated from the tree paths automatically. CFG["build_dir"] becomes BB_BUILD_DIR CFG["llvm"]["dir"] becomes BB_LLVM_DIR

The configuration can be stored/loaded as YAML.

Examples

```
>>> from benchbuild.utils import settings as s
>>> c = s.Configuration('bb')
>>> c['test'] = 42
>>> c['test']
BB_TEST=42
>>> str(c['test'])
'42'
>>> type(c['test'])
<class 'benchbuild.utils.settings.Configuration'>
```

filter_exports()

has_default()
Check, if the node contains a 'default' value.

has_value()
Check, if the node contains a 'value'.

init_from_env()
Initialize this node from environment.

If we're a leaf node, i.e., a node containing a dictionary that consist of a 'default' key, compute our env variable and initialize our value from the environment. Otherwise, init our children.

is_leaf()
Check, if the node is a 'leaf' node.

load(*_from*)
Load the configuration dictionary from file.

store(*config_file*)
Store the configuration dictionary to a file.

to_env_dict()
Convert configuration object to a flat dictionary.

value
Return the node value, if we're a leaf node.

Examples

```
>>> c = Configuration("test")
>>> c['x'] = { "y" : { "value" : None }, "z" : { "value" : 2 }}
>>> c['x']['y'].value == None
True
>>> c['x']['z'].value
2
>>> c['x'].value
TEST_X_Y=null
TEST_X_Z=2
```

exception benchbuild.utils.settings.**InvalidConfigKey**
Bases: RuntimeWarning

Warn, if you access a non-existing key benchbuild's configuration.

benchbuild.utils.settings.**available_cpu_count**()
Get the number of available CPUs.

Number of available virtual or physical CPUs on this system, i.e. user/real as output by time(1) when called with an optimally scaling userspace-only program.

Returns Number of avaialable CPUs.

benchbuild.utils.settings.**convert_components**(*value*)

benchbuild.utils.settings.**escape_yaml**(*raw_str*)
Shell-Escape a yaml input string.

Parameters **raw_str** – The unescaped string.

benchbuild.utils.settings.**find_config**(*test_file=None*, *defaults=None*, *root='.'*)
Find the path to the default config file.

We look at :root: for the :default: config file. If we can't find it there we start looking at the parent directory recursively until we find a file named :default: and return the absolute path to it. If we can't find anything, we return None.

Parameters

- **default** – The name of the config file we look for.
- **root** – The directory to start looking for.

Returns Path to the default config file, None if we can't find anything.

benchbuild.utils.settings.**is_yaml**(*cfg_file*)

benchbuild.utils.settings.**path_constructor**(*loader*, *node*)
" Construct a ConfigPath object form a scalar YAML node.

Tests:

```
>>> yaml.add_constructor("!create-if-needed", path_constructor)
>>> yaml.load("{'test': !create-if-needed '/tmp/test/foo'}")
{'test': ConfigPath(components=['tmp', 'test', 'foo'])}
```

benchbuild.utils.settings.**path_representer**(*dumper*, *data*)
Represent a ConfigPath object as a scalar YAML node.

Tests:

```
>>> yaml.add_representer(ConfigPath, path_representer)
>>> yaml.dump({'test': ConfigPath('/tmp/test/foo')})
"{test: !create-if-needed '/tmp/test/foo'}\n"
```

benchbuild.utils.settings.**setup_config**(*cfg*, *config_filenames=None*, *env_var_name=None*)
This will initialize the given configuration object.

The following resources are available in the same order:

1. Default settings.
2. Config file.
3. Environment variables.

WARNING: Environment variables do _not_ take precedence over the config file right now. (init_from_env will refuse to update the value, if there is already one.)

Parameters

- **config_filenames** – list of possible config filenames
- **env_var_name** – name of the environment variable holding the config path

benchbuild.utils.settings.**to_env_var**(*env_var*, *value*)

benchbuild.utils.settings.**to_yaml**(*value*)

benchbuild.utils.settings.**update_env**(*cfg*)

benchbuild.utils.settings.**upgrade**(*cfg*)
Provide forward migration for configuration files.

benchbuild.utils.settings.**uuid_add_implicit_resolver**(*Loader=<class 'benchbuild.utils.settings.ConfigLoader'>*, *Dumper=<class 'benchbuild.utils.settings.ConfigDumper'>*)

Attach an implicit pattern resolver for UUID objects.

Tests:

```
>>> class TestDumper(yaml.Dumper): pass
>>> class TestLoader(yaml.Loader): pass
>>> TUUID = 'cc3702ca-699a-4aa6-8226-4c938f294d9b'
>>> IN = {'test': uuid.UUID(TUUID)}
>>> OUT = '{test: cc3702ca-699a-4aa6-8226-4c938f294d9b}'
```

```
>>> yaml.add_representer(uuid.UUID, uuid_representer, Dumper=TestDumper)
>>> yaml.add_constructor('!uuid', uuid_constructor, Loader=TestLoader)
>>> uuid_add_implicit_resolver(Loader=TestLoader, Dumper=TestDumper)
```

```
>>> yaml.dump(IN, Dumper=TestDumper)
'{test: cc3702ca-699a-4aa6-8226-4c938f294d9b}\n'
>>> yaml.load(OUT, Loader=TestLoader)
{'test': UUID('cc3702ca-699a-4aa6-8226-4c938f294d9b')}
```

benchbuild.utils.settings.**uuid_constructor**(*loader*, *node*)

” Construct a uuid.UUID object form a scalar YAML node.

Tests:

```
>>> yaml.add_constructor("!uuid", uuid_constructor)
>>> yaml.load("{'test': !uuid 'cc3702ca-699a-4aa6-8226-4c938f294d9b'}")
{'test': UUID('cc3702ca-699a-4aa6-8226-4c938f294d9b')}
```

benchbuild.utils.settings.**uuid_representer**(*dumper*, *data*)

” Represent a uuid.UUID object as a scalar YAML node.

Tests:

```
>>> yaml.add_representer(uuid.UUID, uuid_representer)
>>> yaml.dump({'test': uuid.UUID('cc3702ca-699a-4aa6-8226-4c938f294d9b')})
"{test: !uuid 'cc3702ca-699a-4aa6-8226-4c938f294d9b'}\n"
```

CHAPTER 2

Experiments

2.1 benchbuild.experiments package

Experiments module.

Experiments are discovered automatically by benchbuild. You can configure the modules we search for experiments with the settings:

BB_PLUGINS_AUTOLOAD=True BB_PLUGINS_EXPERIMENTS=[...]

Any subclass of benchbuild.experiments.Experiment will be automatically registered and made available on the command line.

benchbuild.experiments.**discover**()

Import all experiments listed in PLUGINS_EXPERIMENTS.

Tests:

```
>>> from benchbuild.settings import CFG
>>> from benchbuild.experiments import discover
>>> import logging as lg
>>> import sys
>>> l = lg.getLogger('benchbuild')
>>> lg.getLogger('benchbuild').setLevel(lg.DEBUG)
>>> lg.getLogger('benchbuild').handlers = [lg.StreamHandler(stream=sys.
↪stdout)]
>>> CFG["plugins"]["experiments"] = ["benchbuild.non.existing", "benchbuild.
↪experiments.raw"]
>>> discover()
Could not find 'benchbuild.non.existing'
ImportError: No module named 'benchbuild.non'
```

2.1.1 Submodules

benchbuild.experiments.empty module

The 'empty' Experiment.

This experiment is for debugging purposes. It only prepares the basic directories for benchbuild. No compilation & no run can be done with it.

class benchbuild.experiments.empty.**Empty**(*name=NOTHING*, *projects=NOTHING*, *id=NOTHING*, *schema=NOTHING*)

Bases: *benchbuild.experiment.Experiment*

The empty experiment.

NAME = 'empty'

actions_for_project(*project*)
Do nothing.

class benchbuild.experiments.empty.**NoMeasurement**(*name=NOTHING*, *projects=NOTHING*, *id=NOTHING*, *schema=NOTHING*)

Bases: *benchbuild.experiment.Experiment*

Run everything but do not measure anything.

NAME = 'no-measurement'

actions_for_project(*project*)
Execute all actions but don't do anything as extension.

benchbuild.experiments.raw module

The 'raw' Experiment.

This experiment is the basic experiment in the benchbuild study. It simply runs all projects after compiling it with -O3. The binaries are wrapped with the time command and results are written to the database.

This forms the baseline numbers for the other experiments.

Measurements

3 Metrics are generated during this experiment: time.user_s - The time spent in user space in seconds (aka virtual time) time.system_s - The time spent in kernel space in seconds (aka system time) time.real_s - The time spent overall in seconds (aka Wall clock)

class benchbuild.experiments.raw.**RawRuntime**(*name=NOTHING*, *projects=NOTHING*, *id=NOTHING*, *schema=NOTHING*)

Bases: *benchbuild.experiment.Experiment*

The polyjit experiment.

NAME = 'raw'

actions_for_project(*project*)
Compile & Run the experiment with -O3 enabled.

CHAPTER 3

Extensions

CHAPTER 4

Projects

4.1 benchbuild.projects package

Projects module.

By default, only projects that are listed in the configuration are loaded automatically. See configuration variables:

*_PLUGINS_AUTOLOAD *_PLUGINS_PROJECTS

benchbuild.projects.**discover**()

4.1.1 Subpackages

benchbuild.projects.apollo package

Submodules

benchbuild.projects.apollo.rodinia module

class benchbuild.projects.apollo.rodinia.**BFS**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/bfs', 'flags': ['-fopenmp', '-UOPEN'], 'src': {'bfs': ['b

NAME = 'bfs'

static select_compiler(*_*, *cc*)

class benchbuild.projects.apollo.rodinia.**BPlusTree**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/b+tree', 'flags': ['-fopenmp', '-lm'], 'src': {'b+tree.out

NAME = 'b+tree'

class benchbuild.projects.apollo.rodinia.**Backprop**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/backprop', 'flags': ['-fopenmp', '-lm'], 'src': {'backprop

NAME = 'backprop'

class benchbuild.projects.apollo.rodinia.**CFD**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/cfd', 'src': {'euler3d_cpu': ['euler3d_cpu.cpp']}}

NAME = 'cfd'

static select_compiler(*_*, *cc*)

class benchbuild.projects.apollo.rodinia.**HeartWall**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/heartwall', 'flags': ['-I./AVI', '-fopenmp', '-lm'], 'src':

NAME = 'heartwall'

class benchbuild.projects.apollo.rodinia.**Hotspot**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/hotspot', 'flags': ['-fopenmp'], 'src': {'hotspot': ['hot

NAME = 'hotspot'

static select_compiler(*_*, *cc*)

class benchbuild.projects.apollo.rodinia.**Hotspot3D**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/hotspot3D', 'flags': ['-fopenmp', '-lm'], 'src': {'3D': ['

NAME = 'hotspot3D'

class benchbuild.projects.apollo.rodinia.**KMeans**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/kmeans', 'flags': ['-lm', '-fopenmp'], 'src': {'./kmeans_s

NAME = 'kmeans'

class benchbuild.projects.apollo.rodinia.**LUD**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/lud', 'flags': ['-I./common', '-lm', '-fopenmp'], 'src': {

NAME = 'lud'

class benchbuild.projects.apollo.rodinia.**LavaMD**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/lavaMD', 'flags': ['-lm', '-fopenmp'], 'src': {'lavaMD': [

NAME = 'lavaMD'

class benchbuild.projects.apollo.rodinia.**Leukocyte**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/leukocyte', 'flags': ['-DSPARSE', '-DCOMPLEX', '-DREAL_FLT'

NAME = 'leukocyte'

class benchbuild.projects.apollo.rodinia.**Myocyte**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/myocyte', 'flags': ['-lm', '-fopenmp'], 'src': {'./myocyte

NAME = 'myocyte'

class benchbuild.projects.apollo.rodinia.**NN**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/nn', 'flags': ['-lm', '-fopenmp'], 'src': {'nn': ['./nn_o

NAME = 'nn'

class benchbuild.projects.apollo.rodinia.**NW**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/nw', 'flags': ['-lm', '-fopenmp'], 'src': {'needle': ['./

NAME = 'nw'

static select_compiler(*_*, *cc*)

class benchbuild.projects.apollo.rodinia.**ParticleFilter**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/particlefilter', 'flags': ['-lm', '-fopenmp'], 'src': {'pa

NAME = 'particlefilter'

class benchbuild.projects.apollo.rodinia.**PathFinder**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/pathfinder', 'flags': ['-fopenmp'], 'src': {'pathfinder':

NAME = 'pathfinder'

static select_compiler(*_*, *cc*)

class benchbuild.projects.apollo.rodinia.**RodiniaGroup**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.project.Project*

Generic handling of Rodinia benchmarks.

CONFIG = {}

DOMAIN = 'rodinia'

GROUP = 'rodinia'

SRC_FILE = 'rodinia.tar.bz2'

VERSION = '3.1'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static select_compiler(*c_compiler*, *_*)

static versions()
Return a list of versions from the url_dict keys.

class benchbuild.projects.apollo.rodinia.**SRAD1**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/srad/srad_v1', 'flags': ['-I.', '-lm', '-fopenmp'], 'src':

NAME = 'srad-1'

class benchbuild.projects.apollo.rodinia.**SRAD2**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/srad/srad_v2', 'flags': ['-lm', '-fopenmp'], 'src': {'srad

NAME = 'srad-2'

static select_compiler(*_*, *cc*)

class benchbuild.projects.apollo.rodinia.**StreamCluster**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *config=NOTHING*)

Bases: *benchbuild.projects.apollo.rodinia.RodiniaGroup*

CONFIG = {'dir': 'openmp/streamcluster', 'flags': ['-lpthread', '-fopenmp'], 'src':

NAME = 'streamcluster'

static select_compiler(*_*, *cc*)

benchbuild.projects.apollo.scimark module

class benchbuild.projects.apollo.scimark.**SciMark**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'scientific'

GROUP = 'apollo'

NAME = 'scimark'

SRC_FILE = 'scimark.zip'

VERSION = '2.1c'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild package

Submodules

benchbuild.projects.benchbuild.bots module

class benchbuild.projects.benchbuild.bots.**Alignment**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'alignment'

class benchbuild.projects.benchbuild.bots.**BOTSGroup**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

Barcelona OpenMP Task Suite.

Barcelona OpenMP Task Suite is a collection of applications that allow to test OpenMP tasking implementations and compare its behaviour under certain circumstances: task tiedness, throttle and cut-offs mechanisms, single/multiple task generators, etc.

Alignment: Aligns sequences of proteins. FFT: Computes a Fast Fourier Transformation. Floorplan: Computes the optimal placement of cells in a floorplan. Health: Simulates a country health system. NQueens: Finds solutions of the N Queens problem. Sort: Uses a mixture of sorting algorithms to sort a vector. SparseLU: Computes the LU factorization of a sparse matrix. Strassen: Computes a matrix multiply with Strassen's method.

DOMAIN = 'bots'

GROUP = 'bots'

SRC_FILE = 'bots.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

input_dict = {'alignment': ['prot.100.aa', 'prot.20.aa'], 'floorplan': ['input.15',

path_dict = {'alignment': 'serial/alignment', 'fft': 'serial/fft', 'fib': 'serial/f

repository = 'https://github.com/bsc-pm/bots'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

class benchbuild.projects.benchbuild.bots.**FFT**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)
Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'fft'

class benchbuild.projects.benchbuild.bots.**Fib**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)
Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'fib'

class benchbuild.projects.benchbuild.bots.**FloorPlan**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'floorplan'

class benchbuild.projects.benchbuild.bots.**Health**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'health'

class benchbuild.projects.benchbuild.bots.**Knapsack**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'knapsack'

class benchbuild.projects.benchbuild.bots.**NQueens**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'nqueens'

class benchbuild.projects.benchbuild.bots.**Sort**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'sort'

class benchbuild.projects.benchbuild.bots.**SparseLU**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'sparselu'

class benchbuild.projects.benchbuild.bots.**Strassen**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'strassen'

class benchbuild.projects.benchbuild.bots.**UTS**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.benchbuild.bots.BOTSGroup*

NAME = 'uts'

benchbuild.projects.benchbuild.bzip2 module

class benchbuild.projects.benchbuild.bzip2.**Bzip2**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'compression'

GROUP = 'benchbuild'

NAME = 'bzip2'

SRC_FILE = 'bzip2.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://gitlab.com/bzip/bzip2'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.ccrypt module

class benchbuild.projects.benchbuild.ccrypt.**Ccrypt**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

ccrypt benchmark

DOMAIN = 'encryption'

GROUP = 'benchbuild'

NAME = 'ccrypt'

SRC_FILE = 'ccrypt.tar.gz'

VERSION = '1.10'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.crafty module

class benchbuild.projects.benchbuild.crafty.**Crafty**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

crafty benchmark

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'crafty'

SRC_FILE = 'crafty.zip'

VERSION = '25.2'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.crocopat module

class benchbuild.projects.benchbuild.crocopat.**Crocopat**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

crocopat benchmark

DOMAIN = 'verification'

GROUP = 'benchbuild'

NAME = 'crocopat'

SRC_FILE = 'crocopat.zip'

VERSION = '2.1.4'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.ffmpeg module

class benchbuild.projects.benchbuild.ffmpeg.**LibAV**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

LibAV benchmark

DOMAIN = 'multimedia'

GROUP = 'benchbuild'

NAME = 'ffmpeg'

SRC_FILE = 'ffmpeg.tar.bz2'

VERSION = '3.1.3'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

fate_dir = 'fate-samples'

fate_uri = 'rsync://fate-suite.libav.org/fate-suite/'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.gzip module

class benchbuild.projects.benchbuild.gzip.**Gzip**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'compression'

GROUP = 'benchbuild'

NAME = 'gzip'

SRC_FILE = 'gzip.tar.xz'

VERSION = '1.6'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.js module

class benchbuild.projects.benchbuild.js.**SpiderMonkey**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

SpiderMonkey requires a legacy version of autoconf: autoconf-2.13

DOMAIN = 'compilation'

GROUP = 'benchbuild'

NAME = 'js'

SRC_FILE = 'gecko-dev.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/mozilla/gecko-dev.git'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

src_uri = 'https://github.com/mozilla/gecko-dev.git'

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.lammps module

class benchbuild.projects.benchbuild.lammps.**Lammps**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

LAMMPS benchmark

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'lammps'

SRC_FILE = 'lammps.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/lammps/lammps'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.lapack module

class benchbuild.projects.benchbuild.lapack.**Lapack**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'lapack'

SRC_FILE = 'clapack.tgz'

VERSION = '3.2.1'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

class benchbuild.projects.benchbuild.lapack.**OpenBlas**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'openblas'

SRC_FILE = 'OpenBLAS'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/xianyi/OpenBLAS'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.leveldb module

class benchbuild.projects.benchbuild.leveldb.**LevelDB**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'database'

GROUP = 'benchbuild'

NAME = 'leveldb'

SRC_FILE = 'leveldb.src'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/google/leveldb'

run_tests(*runner*)
Execute LevelDB's runtime configuration.

Parameters **experiment** – The experiment's run function.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.linpack module

class benchbuild.projects.benchbuild.linpack.**Linpack**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

Linpack (C-Version)

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'linpack'

SRC_FILE = 'linpack.c'

VERSION = '5/88'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.lulesh module

class benchbuild.projects.benchbuild.lulesh.**Lulesh**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

LULESH, Serial

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'lulesh'

SRC_FILE = 'lulesh.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/LLNL/LULESH/'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

class benchbuild.projects.benchbuild.lulesh.**LuleshOMP**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

LULESH, OpenMP

DOMAIN = 'scientific'

GROUP = 'benchbuild'

NAME = 'lulesh-omp'

SRC_FILE = 'lulesh.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/LLNL/LULESH/'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.mcrypt module

class benchbuild.projects.benchbuild.mcrypt.**MCrypt**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

MCrypt benchmark

DOMAIN = 'encryption'

GROUP = 'benchbuild'

NAME = 'mcrypt'

SRC_FILE = 'mcrypt.tar.gz'

VERSION = '2.6.8'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

libmcrypt_dir = 'libmcrypt-2.5.8'

libmcrypt_file = 'libmcrypt-2.5.8.tar.gz'

libmcrypt_uri = 'http://sourceforge.net/projects/mcrypt/files/Libmcrypt/2.5.8/libmcrypt

mhash_dir = 'mhash-0.9.9.9'

mhash_file = 'mhash-0.9.9.tar.gz'

mhash_uri = 'http://sourceforge.net/projects/mhash/files/mhash/0.9.9/mhash-0.9.9.t

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.minisat module

class benchbuild.projects.benchbuild.minisat.**Minisat**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

minisat benchmark

DOMAIN = 'verification'

GROUP = 'benchbuild'

NAME = 'minisat'

SRC_FILE = 'minisat.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/niklasso/minisat'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.openssl module

class benchbuild.projects.benchbuild.openssl.**LibreSSL**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

OpenSSL

BINARIES = ['aeadtest', 'aes_wrap', 'asn1test', 'base64test', 'bftest', 'bntest', 'byt

DOMAIN = 'encryption'

GROUP = 'benchbuild'

NAME = 'libressl'

SRC_FILE = 'libressl.tar.gz'

VERSION = '2.1.6'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.povray module

class benchbuild.projects.benchbuild.povray.**Povray**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

povray benchmark

DOMAIN = 'multimedia'

GROUP = 'benchbuild'

NAME = 'povray'

SRC_FILE = 'povray.git'

VERSION = 'HEAD'

boost_src_dir = 'boost_1_59_0'

boost_src_file = 'boost_1_59_0.tar.bz2'

boost_src_uri = 'http://sourceforge.net/projects/boost/files/boost/1.59.0/boost_1_59_0

compile()
Compile the project.

download()
Download the selected version.

repository = 'https://github.com/POV-Ray/povray'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.python module

class benchbuild.projects.benchbuild.python.**Python**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

python benchmarks

DOMAIN = 'compilation'

GROUP = 'benchbuild'

NAME = 'python'

SRC_FILE = 'python.tar.xz'

VERSION = '3.4.3'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.rasdaman module

class benchbuild.projects.benchbuild.rasdaman.**Rasdaman**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'database'

GROUP = 'benchbuild'

NAME = 'Rasdaman'

SRC_FILE = 'rasdaman.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

gdal_dir = 'gdal'

gdal_uri = 'https://github.com/OSGeo/gdal'

repository = 'git://rasdaman.org/rasdaman.git'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.ruby module

class benchbuild.projects.benchbuild.ruby.**Ruby**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'compilation'

GROUP = 'benchbuild'

NAME = 'ruby'

SRC_FILE = 'ruby.tar.gz'

VERSION = '2.2.2'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.sdcc module

class benchbuild.projects.benchbuild.sdcc.**SDCC**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'compilation'

GROUP = 'benchbuild'

NAME = 'sdcc'

SRC_FILE = 'sdcc'

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

src_uri = 'svn://svn.code.sf.net/p/sdcc/code/trunk/sdcc'

benchbuild.projects.benchbuild.sevenz module

class benchbuild.projects.benchbuild.sevenz.**SevenZip**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

7Zip

DOMAIN = 'compression'

GROUP = 'benchbuild'

NAME = '7z'

SRC_FILE = 'p7zip.tar.bz2'

VERSION = '16.02'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.sqlite3 module

class benchbuild.projects.benchbuild.sqlite3.**SQLite3**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'database'

GROUP = 'benchbuild'

NAME = 'sqlite3'

SRC_FILE = 'sqlite.zip'

VERSION = '3080900'

build_leveldb()

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

static fetch_leveldb()

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.tcc module

class benchbuild.projects.benchbuild.tcc.**TCC**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'compilation'

GROUP = 'benchbuild'

NAME = 'tcc'

SRC_FILE = 'tcc.tar.bz2'

VERSION = '0.9.26'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.benchbuild.x264 module

class benchbuild.projects.benchbuild.x264.**X264**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

x264

DOMAIN = 'multimedia'

GROUP = 'benchbuild'

NAME = 'x264'

SRC_FILE = 'x264.git'

VERSION = 'HEAD'

compile()
Compile the project.

download()
Download the selected version.

inputfiles = {'Sintel.2010.720p.raw': ['--input-res', '1280x720'], 'tbbt-small.y4m':

repository = 'git://git.videolan.org/x264.git'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

src_uri = 'git://git.videolan.org/x264.git'

static versions()
Return a list of versions from the git hashes up to :limit:.

benchbuild.projects.benchbuild.xz module

class benchbuild.projects.benchbuild.xz.**XZ**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'compression'

GROUP = 'benchbuild'

NAME = 'xz'

SRC_FILE = 'xz.tar.gz'

VERSION = '5.2.1'

compile()
Compile the project.

download()
Download the selected version from the url_dict value.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

static versions()
Return a list of versions from the url_dict keys.

benchbuild.projects.gentoo package

Import all gentoo based modules.

All manually entered modules can be placed in the following import section. Portage_Gen based projects will be generated automatically as soon as we can find an index generated by portage info.

Submodules

benchbuild.projects.gentoo.autoportage module

class benchbuild.projects.gentoo.autoportage.**AutoPortage**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

Generic portage experiment.

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

benchbuild.projects.gentoo.bzip2 module

bzip2 experiment within gentoo chroot.

class benchbuild.projects.gentoo.bzip2.**BZip2**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/bzip2

DOMAIN = 'app-arch'

NAME = 'bzip2'

compile()

Compile the project.

run_tests(*runner*)

Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

test_archive = 'compression.tar.gz'

test_url = 'http://lairosiel.de/dist/'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.gentoo.crafty module

crafty experiment within gentoo chroot.

class benchbuild.projects.gentoo.crafty.**Crafty**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

games-board/crafty

DOMAIN = 'games-board'

NAME = 'crafty'

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

benchbuild.projects.gentoo.eix module

eix experiment within gentoo chroot

class benchbuild.projects.gentoo.eix.**Eix**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

Represents the package eix from the portage tree.

DOMAIN = 'app-portage'

NAME = 'eix'

run_tests(*runner*)
Runs runtime tests for eix

benchbuild.projects.gentoo.gentoo module

The Gentoo module for running tests on builds from the portage tree.

This will install a stage3 image of gentoo together with a recent snapshot of the portage tree. For building / executing arbitrary projects successfully it is necessary to keep the installed image as close to the host system as possible. In order to speed up your experience, you can replace the stage3 image that we pull from the distfiles mirror with a new image that contains all necessary dependencies for your experiments. Make sure you update the hash alongside the gentoo image in benchbuild's source directory.

class benchbuild.projects.gentoo.gentoo.**GentooGroup**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.project.Project*

Gentoo ProjectGroup is the base class for every portage build.

CONTAINER = <benchbuild.utils.container.Gentoo object>

GROUP = 'gentoo'

SRC_FILE = None

compile()
Compile the project.

configure_benchbuild(*cfg*)

redirect()
Redirect execution to a containerized benchbuild instance.

benchbuild.projects.gentoo.gentoo.**configure_portage**()

benchbuild.projects.gentoo.gentoo.**emerge**(*package*, **args*, *env=None*)

benchbuild.projects.gentoo.gentoo.**find_benchbuild**()

benchbuild.projects.gentoo.gentoo.**requires_update**(*benchbuild*)

benchbuild.projects.gentoo.gentoo.**setup_benchbuild**()
Setup benchbuild inside a container.

This will query a for an existing installation of benchbuild and try to upgrade it to the latest version, if possible.

benchbuild.projects.gentoo.gentoo.**setup_compilers**(*_path*)

benchbuild.projects.gentoo.gentoo.**setup_networking**()

benchbuild.projects.gentoo.gentoo.**setup_virtualenv**(*_path='/benchbuild'*)

benchbuild.projects.gentoo.gentoo.**write_bashrc**(*_path*)
Write a valid gentoo bashrc file to :path:.

Parameters – The output path of the make.conf(*path*) –

benchbuild.projects.gentoo.gentoo.**write_layout**(*_path*)
Write a valid gentoo layout file to :path:.

Parameters – The output path of the layout.conf(*path*) –

benchbuild.projects.gentoo.gentoo.**write_makeconfig**(*_path*)
Write a valid gentoo make.conf file to :path:.

Parameters – **The output path of the make.conf** (*path*) –

benchbuild.projects.gentoo.gentoo.**write_sandbox_d**(*_path*)

benchbuild.projects.gentoo.gentoo.**write_wgetrc**(*_path*)

Write a valid gentoo wgetrc file to :path:.

Parameters – **The output path of the wgetrc** (*path*) –

benchbuild.projects.gentoo.gzip module

gzip experiment within gentoo chroot.

class benchbuild.projects.gentoo.gzip.**GZip**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/gzip

DOMAIN = 'app-arch'

NAME = 'gzip'

compile()

Compile the project.

run_tests(*runner*)

Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

test_archive = 'compression.tar.gz'

test_url = 'http://lairosiel.de/dist/'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.gentoo.info module

Get package infos, e.g., specific ebuilds for given languages, from gentoo chroot.

class benchbuild.projects.gentoo.info.**Info**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.autoportage.AutoPortage*

Info experiment to retrieve package information from portage.

DOMAIN = 'debug'

NAME = 'info'

compile()

Compile the project.

benchbuild.projects.gentoo.info.**get_string_for_language**(*language_name*)

Maps language names to the corresponding string for qgrep.

benchbuild.projects.gentoo.lammps module

LAMMPS (sci-physics/lammps) project within gentoo chroot.

class benchbuild.projects.gentoo.lammps.**Lammps**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

sci-physics/lammps

DOMAIN = 'sci-physics'

NAME = 'lammps'

compile()

Compile the project.

run_tests(*runner*)

Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

test_archive = 'lammps.tar.gz'

test_url = 'http://lairosiel.de/dist/'

benchbuild.projects.gentoo.portage_gen module

Generic experiment to test portage packages within gentoo chroot.

class benchbuild.projects.gentoo.portage_gen.**FuncClass**(*name*, *domain*, *_container*)
Bases: object

Finds out the current version number of a gentoo package.

The package name is created by combining the domain and the name. Then uchroot is used to switch into a gentoo shell where the 'emerge' command is used to recieve the version number. The function then parses the version number back into the file.

Parameters

- **Name** – Name of the project.
- **Domain** – Categorie of the package.

benchbuild.projects.gentoo.portage_gen.**PortageFactory**(*name*, *NAME*, *DOMAIN*, *BaseClass=<class 'benchbuild.projects.gentoo.autoportage.AutoPortage'>*)

Create a new dynamic portage project.

Auto-Generated projects can only be used for compilie-time experiments, because there simply is no run-time test defined for it. Therefore, we implement the run symbol as a noop (with minor logging).

This way we avoid the default implementation for run() that all projects inherit.

Parameters

- **name** – Name of the dynamic class.
- **NAME** – NAME property of the dynamic class.
- **DOMAIN** – DOMAIN property of the dynamic class.
- **BaseClass** – Base class to use for the dynamic class.

Returns A new class with NAME,DOMAIN properties set, unable to perform run-time tests.

Examples

```
>>> from benchbuild.projects.gentoo.portage_gen import PortageFactory
>>> from benchbuild.experiments.empty import Empty
>>> c = PortageFactory("test", "NAME", "DOMAIN")
>>> c
<class '__main__.test'>
>>> i = c(Empty())
>>> i.NAME
'NAME'
>>> i.DOMAIN
'DOMAIN'
```

benchbuild.projects.gentoo.postgresql module

postgresql experiment within gentoo chroot.

class benchbuild.projects.gentoo.postgresql.**Postgresql**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

dev-db/postgresql

DOMAIN = 'dev-db/postgresql'

NAME = 'postgresql'

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

benchbuild.projects.gentoo.sevenz module

p7zip experiment within gentoo chroot.

class benchbuild.projects.gentoo.sevenz.**SevenZip**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/p7zip

DOMAIN = 'app-arch'

NAME = 'p7zip'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

benchbuild.projects.gentoo.x264 module

media-video/x264-encoder within gentoo chroot.

class benchbuild.projects.gentoo.x264.**X264**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)
Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

media-video/x264-encoder

DOMAIN = 'media-libs'

NAME = 'x264'

compile()
Compile the project.

inputfiles = {'Sintel.2010.720p.raw': ['--input-res', '1280x720'], 'tbbt-small.y4m':

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

test_url = 'http://lairosiel.de/dist/'

benchbuild.projects.gentoo.xz module

xz experiment within gentoo chroot.

class benchbuild.projects.gentoo.xz.**XZ**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*, *emerge_env={}*)

Bases: *benchbuild.projects.gentoo.gentoo.GentooGroup*

app-arch/xz

DOMAIN = 'app-arch'

NAME = 'xz'

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

test_archive = 'compression.tar.gz'

test_url = 'http://lairosiel.de/dist/'

testfiles = ['text.html', 'chicken.jpg', 'control', 'input.source', 'liberty.jpg']

benchbuild.projects.lnt package

Submodules

benchbuild.projects.lnt.lnt module

LNT based measurements.

class benchbuild.projects.lnt.lnt.**LNTGroup**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

LNT ProjectGroup for running the lnt test suite.

DOMAIN = 'lnt'

GROUP = 'lnt'

NAME_FILTERS = ['(?P<name>.+)\\.simple', '(?P<name>.+)-(dbl|flt)']

SRC_FILE = 'lnt.git'

SUBDIR = None

VERSION = 'HEAD'

static after_run_tests(*sandbox_dir*)

binary = None

clang = None

clang_cxx = None

compile()
Compile the project.

download()
Download the selected version.

lnt = None

repository = 'http://llvm.org/git/lnt'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

sandbox_dir = None

src_dir = 'lnt'

test_suite_dir = 'test-suite'

test_suite_uri = 'http://llvm.org/git/test-suite'

static versions()
Return a list of versions from the git hashes up to :limit:.

class benchbuild.projects.lnt.lnt.**MultiSourceApplications**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

DOMAIN = 'LNT (MSA)'

NAME = 'MultiSourceApplications'

SUBDIR = 'MultiSource/Applications'

class benchbuild.projects.lnt.lnt.**MultiSourceBenchmarks**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

DOMAIN = 'LNT (MSB)'

NAME = 'MultiSourceBenchmarks'

SUBDIR = 'MultiSource/Benchmarks'

class benchbuild.projects.lnt.lnt.**Povray**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

DOMAIN = 'LNT (Ext)'

NAME = 'Povray'

SUBDIR = 'External/Povray'

compile()

Compile the project.

povray_src_dir = 'Povray'

povray_url = 'https://github.com/POV-Ray/povray'

class benchbuild.projects.lnt.lnt.**SPEC2006**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

DOMAIN = 'LNT (Ext)'

NAME = 'SPEC2006'

SUBDIR = 'External/SPEC'

compile()

Compile the project.

class benchbuild.projects.lnt.lnt.**SingleSourceBenchmarks**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.lnt.lnt.LNTGroup*

DOMAIN = 'LNT (SSB)'

NAME = 'SingleSourceBenchmarks'

SUBDIR = 'SingleSource/Benchmarks'

benchbuild.projects.polybench package

Submodules

benchbuild.projects.polybench.polybench-mod module

benchbuild.projects.polybench.polybench module

class benchbuild.projects.polybench.polybench.**Adi**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'adi'

class benchbuild.projects.polybench.polybench.**Atax**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'atax'

class benchbuild.projects.polybench.polybench.**BicG**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'bicg'

class benchbuild.projects.polybench.polybench.**Cholesky**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'cholesky'

class benchbuild.projects.polybench.polybench.**Correlation**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'correlation'

class benchbuild.projects.polybench.polybench.**Covariance**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'covariance'

class benchbuild.projects.polybench.polybench.**Deriche**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'deriche'

class benchbuild.projects.polybench.polybench.**Doitgen**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'doitgen'

class benchbuild.projects.polybench.polybench.**Durbin**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'durbin'

class benchbuild.projects.polybench.polybench.**FDTD2D**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'fdtd-2d'

class benchbuild.projects.polybench.polybench.**FloydWarshall**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'floyd-warshall'

class benchbuild.projects.polybench.polybench.**Gemm**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gemm'

class benchbuild.projects.polybench.polybench.**Gemver**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gemver'

class benchbuild.projects.polybench.polybench.**Gesummv**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gesummv'

class benchbuild.projects.polybench.polybench.**Gramschmidt**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'gramschmidt'

class benchbuild.projects.polybench.polybench.**Heat3D**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'heat-3d'

class benchbuild.projects.polybench.polybench.**Jacobi1D**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'jacobi-1d'

class benchbuild.projects.polybench.polybench.**Jacobi2Dimper**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'jacobi-2d'

class benchbuild.projects.polybench.polybench.**Lu**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'lu'

class benchbuild.projects.polybench.polybench.**LuDCMP**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'ludcmp'

class benchbuild.projects.polybench.polybench.**Mvt**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'mvt'

class benchbuild.projects.polybench.polybench.**Nussinov**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'nussinov'

class benchbuild.projects.polybench.polybench.**PolyBenchGroup**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

DOMAIN = 'polybench'

GROUP = 'polybench'

SRC_FILE = 'polybench.tar.gz'

VERSION = '4.2'

compile()
Compile the project.

compile_verify(*compiler_args*, *polybench_opts*)

download()
Download the selected version from the url_dict value.

path_dict = {'2mm': 'linear-algebra/kernels', '3mm': 'linear-algebra/kernels', 'adi'

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

static versions()
Return a list of versions from the url_dict keys.

class benchbuild.projects.polybench.polybench.**Seidel2D**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'seidel-2d'

class benchbuild.projects.polybench.polybench.**Symm**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'symm'

class benchbuild.projects.polybench.polybench.**Syr2k**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'syr2k'

class benchbuild.projects.polybench.polybench.**Syrk**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'syrk'

class benchbuild.projects.polybench.polybench.**ThreeMM**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = '3mm'

class benchbuild.projects.polybench.polybench.**Trisolv**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'trisolv'

class benchbuild.projects.polybench.polybench.**Trmm**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = 'trmm'

class benchbuild.projects.polybench.polybench.**TwoMM**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.projects.polybench.polybench.PolyBenchGroup*

NAME = '2mm'

benchbuild.projects.polybench.polybench.**get_dump_arrays_output**(*data*)

benchbuild.projects.test package

Submodules

benchbuild.projects.test.test module

class benchbuild.projects.test.test.**TestProject**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

Test project that does nothing.

DOMAIN = 'test'

GROUP = 'test'

NAME = 'test'

SRC_FILE = 'test.cpp'

VERSION = '1.0'

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

class benchbuild.projects.test.test.**TestProjectRuntimeFail**(*experiment*, *name=NOTHING*, *domain=NOTHING*, *group=NOTHING*, *src_file=NOTHING*, *container=NOTHING*, *version=NOTHING*, *builddir=NOTHING*, *testdir=NOTHING*, *cflags=NOTHING*, *ldflags=NOTHING*, *run_f=NOTHING*, *run_uuid=NOTHING*, *compiler_extension=NOTHING*, *runtime_extension=None*)

Bases: *benchbuild.project.Project*

Test project that _always_ fails at runtime.

DOMAIN = 'test'

GROUP = 'test'

NAME = 'test-fail'

SRC_FILE = 'test.cpp'

VERSION = '1.0'

compile()
Compile the project.

run_tests(*runner*)
Run the tests of this project.

Clients override this method to provide customized run-time tests.

Parameters

- **experiment** – The experiment we run this project under
- **run** – A function that takes the run command.

CHAPTER 5

Settings

CHAPTER 6

Full API

6.1 benchbuild package

Setup plugins.

6.1.1 Subpackages

benchbuild.cli package

The CLI package.

Submodules

benchbuild.cli.bootstrap module

class benchbuild.cli.bootstrap.**BenchBuildBootstrap**(*executable*)
Bases: plumbum.cli.application.Application

Bootstrap benchbuild external dependencies, if possible.

main(**args*)
Implement me (no need to call super)

store_config
Sets an attribute

benchbuild.cli.config module

Subcommand for configuration handling.

class benchbuild.cli.config.**BBConfig**(*executable*)
Bases: plumbum.cli.application.Application

Manage BenchBuild's configuration.

main()
Implement me (no need to call super)

class benchbuild.cli.config.**BBConfigView**(*executable*)
Bases: plumbum.cli.application.Application

View the current configuration.

main()
Implement me (no need to call super)

class benchbuild.cli.config.**BBConfigWrite**(*executable*)
Bases: plumbum.cli.application.Application

Write the current configuration to a file.

main()
Implement me (no need to call super)

benchbuild.cli.experiment module

Subcommand for experiment handling.

class benchbuild.cli.experiment.**BBExperiment**(*executable*)
Bases: plumbum.cli.application.Application

Manage BenchBuild's known experiments.

main()
Implement me (no need to call super)

class benchbuild.cli.experiment.**BBExperimentShow**(*executable*)
Bases: plumbum.cli.application.Application

Show completed experiments.

main()
Implement me (no need to call super)

class benchbuild.cli.experiment.**BBExperimentView**(*executable*)
Bases: plumbum.cli.application.Application

View available experiments.

main()
Implement me (no need to call super)

class benchbuild.cli.experiment.**Choice**(*caption*, *payload*, *top*)
Bases: urwid.widget.WidgetWrap

item_chosen(_)

class benchbuild.cli.experiment.**HorizontalBoxes**
Bases: urwid.container.Columns

clear()

open_box(*box*)

class benchbuild.cli.experiment.**MenuButton**(*caption*, *callback*)
Bases: urwid.wimp.Button

class benchbuild.cli.experiment.**SubMenu**(*caption*, *choices*, *top*)
Bases: urwid.widget.WidgetWrap

open_menu(_)

benchbuild.cli.experiment.**experiments_from_db**(*session*)

benchbuild.cli.experiment.**get_completed_runs**(*session*, *exp*)

benchbuild.cli.experiment.**get_failed_runs**(*session*, *exp*)

benchbuild.cli.experiment.**get_template**()

benchbuild.cli.experiment.**refresh_root_window**(*root*)

benchbuild.cli.experiment.**render_experiment**(*_experiment*)

benchbuild.cli.log module

Analyze the BB database.

class benchbuild.cli.log.**BenchBuildLog**(*executable*)
Bases: plumbum.cli.application.Application

Frontend command to the benchbuild database.

experiment(*experiments*)
Set the experiments to fetch the log for.

experiment_ids(*experiment_ids*)
Set the experiment ids to fetch the log for.

log_type(*types*)
Set the output types to print.

main(**projects*)
Run the log command.

project_ids(*project_ids*)
Set the project ids to fetch the log for.

benchbuild.cli.log.**print_logs**(*query*, *types=None*)
Print status logs.

benchbuild.cli.log.**print_runs**(*query*)
Print all rows in this result query.

benchbuild.cli.main module

Main CLI unit of BenchBuild.

class benchbuild.cli.main.**BenchBuild**(*executable*)
Bases: plumbum.cli.application.Application

Frontend for running/building the benchbuild study framework.

VERSION = '3.4.2.dev3+g0712eee'

debug
Sets an attribute

main(**args*)
Implement me (no need to call super)

verbosity
Sets an attribute

benchbuild.cli.project module

Subcommand for project handling.

class benchbuild.cli.project.**BBProject**(*executable*)
Bases: plumbum.cli.application.Application

Manage BenchBuild's known projects.

main()
Implement me (no need to call super)

class benchbuild.cli.project.**BBProjectView**(*executable*)
Bases: plumbum.cli.application.Application

View available projects.

groups = None

main(**projects*)
Implement me (no need to call super)

set_group(*groups*)

benchbuild.cli.project.**print_projects**(*projects=None*)
Print a list of projects registered for that experiment.

Parameters **exp** – The experiment to print all projects for.

benchbuild.cli.report module

class benchbuild.cli.report.**BenchBuildReport**(*executable*)
Bases: plumbum.cli.application.Application

Generate Reports from the benchbuild db.

experiment_ids(*ids*)

experiments(*_experiments*)

main(**args*)
Implement me (no need to call super)

outfile(*outfile*)

reports(*_reports*)

benchbuild.cli.run module

benchbuild's run command.

This subcommand executes experiments on a set of user-controlled projects. See the output of benchbuild run –help for more information.

class benchbuild.cli.run.**BenchBuildRun**(*executable*)
Bases: plumbum.cli.application.Application

Frontend for running experiments in the benchbuild study framework.

experiment_names = []

group_names = None

main(**projects*)
Main entry point of benchbuild run.

pretend
Sets an attribute

set_experiment_tag(*description*)

set_experiments(*names*)

set_group(*groups*)

static setup_progress(*cfg*, *num_actions*)
Setup a progress bar.

Parameters

- **cfg** – Configuration dictionary.
- **num_actions** (*int*) – Number of actions in the plan.

Returns The configured progress bar.

show_progress
Sets an attribute

test_full
Sets an attribute

benchbuild.cli.run.**execute_plan**(*plan*)
"Execute the plan.

Parameters **plan** (list of actions.Step) – The plan we want to execute.

Returns A list of failed actions.

Return type (list of actions.Step)

benchbuild.cli.run.**print_summary**(*num_actions*, *failed*, *duration*)
Print a small summary of the executed plan.

Parameters

- **num_actions** (*int*) – Total size of the executed plan.
- **failed** (list of actions.Step) – List of failed actions.
- **duration** – Time we spent executing the plan.

benchbuild.cli.slurm module

Dump SLURM script that executes the selected experiment with all projects.

This basically provides the same as benchbuild run, except that it just dumps a slurm batch script that executes everything as an array job on a configurable SLURM cluster.

class benchbuild.cli.slurm.**Slurm**(*executable*)
Bases: plumbum.cli.application.Application

Generate a SLURM script.

experiment(*cfg_experiment*)
Specify experiments to run

experiment_tag(*description*)
A description for this experiment run

group(*groups*)
Run a group of projects under the given experiments

main(**projects*)
Main entry point of benchbuild run.

benchbuild.db package

Subpackages

benchbuild.db.versions package

Submodules

benchbuild.db.versions.001_Remove_RegressionTest_table module

Remove unneeded Regressions table.

This table can and should be reintroduced by an experiment that requires it.

benchbuild.db.versions.001_Remove_RegressionTest_table.**downgrade**(*migrate_engine*)

benchbuild.db.versions.001_Remove_RegressionTest_table.**upgrade**(*migrate_engine*)

benchbuild.db.versions.002_Remove_GlobalConfig_table module

Remove unneeded GlobalConfig table.

This table can and should be reintroduced by an experiment that requires it.

benchbuild.db.versions.002_Remove_GlobalConfig_table.**downgrade**(*migrate_engine*)

benchbuild.db.versions.002_Remove_GlobalConfig_table.**upgrade**(*migrate_engine*)

benchbuild.db.versions.003_Unmanage_Events module

Remove 'benchbuild_events' from the managed part of the schema.

We do not delete this table during our upgrades, because we do not want to wipe measurement data.

During downgrade we will make sure to create the table as needed.

benchbuild.db.versions.003_Unmanage_Events.**downgrade**(*migrate_engine*)

benchbuild.db.versions.003_Unmanage_Events.**upgrade**(*migrate_engine*)

Submodules

benchbuild.db.manage module

benchbuild.reports package

Register reports for an experiment

class benchbuild.reports.**Report**(*experiment_name*, *exp_ids*, *out_path*, *session*, *name=NOTHING*, *supported_experiments=NOTHING*, *experiment_ids=None*)

Bases: object

NAME = None

SUPPORTED_EXPERIMENTS = []

class benchbuild.reports.**ReportRegistry**(*name*, *bases*, *_dict*)

Bases: type

reports = {'full': <class 'benchbuild.reports.status.FullDump'>, 'raw': <class 'benchbuild.reports.raw.Raw'>}

benchbuild.reports.**discover**()

Import all experiments listed in *_PLUGINS_REPORTS.

Tests:

```
>>> from benchbuild.settings import CFG
>>> from benchbuild.reports import discover
>>> import logging as lg
>>> import sys
>>> l = lg.getLogger('benchbuild')
>>> l.setLevel(lg.DEBUG)
>>> l.handlers = [lg.StreamHandler(stream=sys.stdout)]
>>> CFG["plugins"]["reports"] = ["benchbuild.non.existing", "benchbuild.
↪reports.raw"]
>>> discover()
Could not find 'benchbuild.non.existing'
Found report: benchbuild.reports.raw
```

benchbuild.reports.**load_experiment_ids_from_names**(*session*, *names*)

Submodules

benchbuild.reports.raw module

class benchbuild.reports.raw.**RawReport**(*experiment_name*, *exp_ids*, *out_path*, *session*, *name=NOTHING*, *supported_experiments=NOTHING*, *experiment_ids=None*)

Bases: *benchbuild.reports.Report*

NAME = 'raw'

SUPPORTED_EXPERIMENTS = ['raw']

generate()

report()

benchbuild.reports.status module

class benchbuild.reports.status.**FullDump**(*experiment_name*, *exp_ids*, *out_path*, *session*, *name=NOTHING*, *supported_experiments=NOTHING*, *experiment_ids=None*)

Bases: *benchbuild.reports.Report*

Generate a dump of all rows associated with this experiment.

NAME = 'full'

SUPPORTED_EXPERIMENTS = ['empty', 'raw', 'no-measurement']

exp_name()

generate()

Fetch all rows associated with this experiment.

This will generate a huge .csv.

report()

class benchbuild.reports.status.**StatusReport**(*experiment_name*, *exp_ids*, *out_path*, *session*, *name=NOTHING*, *supported_experiments=NOTHING*, *experiment_ids=None*)

Bases: *benchbuild.reports.Report*

NAME = 'status'

QUERY_STATUS = <sqlalchemy.sql.selectable.Select at 0x7f09087fe2b0; Select object>

SUPPORTED_EXPERIMENTS = ['empty', 'raw', 'no-measurement']

generate()

report()

benchbuild.utils package

Module handler that makes sure the modules for our commands are build similar to plumbum. The built modules are only active during a run of an experiment and get deleted afterwards.

benchbuild.utils.**cmd**
Module-hack, adapted from plumbum.

Submodules

benchbuild.utils.actions module

Actions

Actions are enhanced callables that are used by *Experiments* to define the order of operations a project is put through when the experiment executes.

Example

TODO `python `

class benchbuild.utils.actions.**Any**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*, *actions=NOTHING*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Just run all actions, no questions asked.'

NAME = 'ANY'

class benchbuild.utils.actions.**Clean**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*, *check_empty=False*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Cleans the build directory'

NAME = 'CLEAN'

static clean_mountpoints(*root: str*)
Unmount any remaining mountpoints under :root.

Parameters **root** – All UnionFS-mountpoints under this directory will be unmounted.

class benchbuild.utils.actions.**CleanExtra**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Cleans the extra directories.'

NAME = 'CLEAN EXTRA'

class benchbuild.utils.actions.**Compile**(*project*)
Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Compile the project'

NAME = 'COMPILE'

class benchbuild.utils.actions.**Containerize**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*, *actions=NOTHING*)
Bases: *benchbuild.utils.actions.RequireAll*

DESCRITPION = 'Redirect into container'

NAME = 'CONTAINERIZE'

requires_redirect()

class benchbuild.utils.actions.**Echo**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*, *message=''*)

Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Print a message.'

NAME = 'ECHO'

class benchbuild.utils.actions.**Experiment**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*, *actions=NOTHING*)

Bases: *benchbuild.utils.actions.Any*

DESCRIPTION = 'Run a experiment, wrapped in a db transaction'

NAME = 'EXPERIMENT'

begin_transaction()

static end_transaction(*experiment*, *session*)

class benchbuild.utils.actions.**MakeBuildDir**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*)

Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Create the build directory'

NAME = 'MKDIR'

class benchbuild.utils.actions.**RequireAll**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*, *actions=NOTHING*)

Bases: *benchbuild.utils.actions.Step*

class benchbuild.utils.actions.**Run**(*project*)

Bases: *benchbuild.utils.actions.Step*

DESCRIPTION = 'Execute the run action'

NAME = 'RUN'

class benchbuild.utils.actions.**Step**(*obj=None*, *action_fn=None*, *status=<StepResult.UNSET: 0>*)

Bases: `object`

Base class of a step.

This stores all common attributes for step classes.

metaclass ([type], optional): Defaults to StepClass. Takes care of wrapping Steps correctly.

Raises `StopIteration` – If we do not encapsulate more substeps.

DESCRIPTION = None

NAME = None

ON_STEP_BEGIN = []

ON_STEP_END = []

onerror()

class benchbuild.utils.actions.**StepClass**

Bases: `abc.ABCMeta`

Decorate *steps* with logging and result conversion.

class benchbuild.utils.actions.**StepResult**
Bases: enum.IntEnum

Result type for action results.

CAN_CONTINUE = 2

ERROR = 3

OK = 1

UNSET = 0

benchbuild.utils.actions.**log_before_after**(*name: str*, *desc: str*)
Log customized stirng before & after running func.

benchbuild.utils.actions.**notify_step_begin_end**(*func*)
Print the beginning and the end of a *func*.

benchbuild.utils.actions.**num_steps**(*steps*)

benchbuild.utils.actions.**prepend_status**(*func*)
Prepends the output of *func* with the status.

benchbuild.utils.actions.**print_steps**(*steps*)

benchbuild.utils.actions.**step_has_failed**(*step_results*, *error_status=None*)

benchbuild.utils.actions.**to_step_result**(*func*)
Convert a function return to a list of StepResults.

All Step subclasses automatically wrap the result of their __call__ method's result with this wrapper. If the result is not a list of StepResult values, one will be generated.

result of *[StepResult.OK]*, or convert the given result into a list.

Parameters **func** – The function to wrap.

benchbuild.utils.bootstrap module

Helper functions for bootstrapping external dependencies.

benchbuild.utils.bootstrap.**check_uchroot_config**()

benchbuild.utils.bootstrap.**find_package**(*binary*)

benchbuild.utils.bootstrap.**install_package**(*pkg_name*)

benchbuild.utils.bootstrap.**install_uchroot**(*_*)
Installer for erlent (contains uchroot).

benchbuild.utils.bootstrap.**linux_distribution_major**()

benchbuild.utils.bootstrap.**provide_package**(*pkg_name*, *installer=<function install_package>*)

benchbuild.utils.bootstrap.**provide_packages**(*pkg_names*)

benchbuild.utils.compiler module

Helper functions for dealing with compiler replacement.

This provides a few key functions to deal with varying/measuring the compilers used inside the benchbuild study. From a high-level view, there are 2 interesting functions:

- cc(project, detect_project=True)
- cxx(project, detect_project=True)

These generate a wrapped clang/clang++ in the current working directory and hide the given cflags/ldflags from the calling build system. Both will give you a working plumbum command and call a python script that redirects to the real clang/clang++ given the additional cflags&ldflags.

The wrapper-script generated for both functions can be found inside:

- wrap_cc()

Are just convencience methods that can be used when interacting with the configured llvm/clang source directories.

benchbuild.utils.compiler.**cc**(*project*, *detect_project=False*)
Return a clang that hides CFLAGS and LDFLAGS.

This will generate a wrapper script in the current directory and return a complete plumbum command to it.

Parameters

- **cflags** – The CFLAGS we want to hide.
- **ldflags** – The LDFLAGS we want to hide.
- **func** (*optional*) – A function that will be pickled alongside the compiler. It will be called before the actual compilation took place. This way you can intercept the compilation process with arbitrary python code.

Returns (benchbuild.utils.cmd): Path to the new clang command.

benchbuild.utils.compiler.**compiler**(*name*)
Get a usable clang++ plumbum command.

This searches for a usable clang++ in the llvm binary path

Returns plumbum Command that executes clang++

benchbuild.utils.compiler.**cxx**(*project*, *detect_project=False*)
Return a clang++ that hides CFLAGS and LDFLAGS.

This will generate a wrapper script in the current directory and return a complete plumbum command to it.

Parameters

- **cflags** – The CFLAGS we want to hide.
- **ldflags** – The LDFLAGS we want to hide.
- **func** (*optional*) – A function that will be pickled alongside the compiler. It will be called before the actual compilation took place. This way you can intercept the compilation process with arbitrary python code.

Returns (benchbuild.utils.cmd): Path to the new clang command.

benchbuild.utils.container module

Container utilites.

class benchbuild.utils.container.**Container**
Bases: object

filename

local
Finds the current location of a container. Also unpacks the project if necessary.

Returns The path, where the container lies in the end.

Return type target

name = 'container'

remote

class benchbuild.utils.container.**Gentoo**
Bases: *benchbuild.utils.container.Container*

name = 'gentoo'

remote
Get a remote URL of the requested container.

src_file(***kwargs*)

benchbuild.utils.container.**cached**(*func*)
Memoize a function result.

benchbuild.utils.container.**in_container**()
Check, whether we are running inside a container.

benchbuild.utils.container.**is_valid**(*container*, *path*)
Checks if a container exists and is unpacked.

Parameters **path** – The location where the container is expected.

Returns True if the container is valid, False if the container needs to unpacked or if the path does not exist yet.

benchbuild.utils.container.**unpack**(*container*, *path*)
Unpack a container usable by uchroot.

Method that checks if a directory for the container exists, checks if erlent support is needed and then unpacks the container accordingly.

Parameters **path** – The location where the container is, that needs to be unpacked.

benchbuild.utils.db module

Database support module for the benchbuild study.

benchbuild.utils.db.**create_run**(*cmd*, *project*, *exp*, *grp*)
Create a new 'run' in the database.

This creates a new transaction in the database and creates a new run in this transaction. Afterwards we return both the transaction as well as the run itself. The user is responsible for committing it when the time comes.

Parameters

- **cmd** – The command that has been executed.
- **prj** – The project this run belongs to.
- **exp** – The experiment this run belongs to.
- **grp** – The run_group (uuid) we blong to.

Returns The inserted tuple representing the run and the session opened with the new run. Don't forget to commit it at some point.

benchbuild.utils.db.**create_run_group**(*prj*)
Create a new 'run_group' in the database.

This creates a new transaction in the database and creates a new run_group within this transaction. Afterwards we return both the transaction as well as the run_group itself. The user is responsible for committing it when the time comes.

Parameters **- The project for which we open the run_group.** (*prj*) –

Returns A tuple (group, session) containing both the newly created run_group and the transaction object.

benchbuild.utils.db.**persist_compilestats**(*run*, *session*, *stats*)
Persist the run results in the database.

Parameters

- **run** – The run we attach the compilestats to.
- **session** – The db transaction we belong to.
- **stats** – The stats we want to store in the database.

benchbuild.utils.db.**persist_config**(*run*, *session*, *cfg*)
Persist the configuration in as key-value pairs.

Parameters

- **run** – The run we attach the config to.
- **session** – The db transaction we belong to.
- **cfg** – The configuration we want to persist.

benchbuild.utils.db.**persist_experiment**(*experiment*)
Persist this experiment in the benchbuild database.

Parameters **experiment** – The experiment we want to persist.

benchbuild.utils.db.**persist_perf**(*run*, *session*, *svg_path*)
Persist the flamegraph in the database.

The flamegraph exists as a SVG image on disk until we persist it in the database.

Parameters

- **run** – The run we attach these perf measurements to.
- **session** – The db transaction we belong to.
- **svg_path** – The path to the SVG file we want to store.

benchbuild.utils.db.**persist_project**(*project*)
Persist this project in the benchbuild database.

Parameters **project** – The project we want to persist.

benchbuild.utils.db.**persist_time**(*run*, *session*, **args*, ***kwargs*)

benchbuild.utils.db.**validate**(*func*)

benchbuild.utils.dict module

An extensible dictionary.

class benchbuild.utils.dict.**ExtensibleDict**(*extender_fn=None*)
Bases: object

A dictionary that provides temporary modification.

clear()

get(*name*, **default*)

getdict()

items()

keys()

pop(*name*, **default*)

update(*extender_fn*, **args*, ***kwargs*)

values()

benchbuild.utils.dict.**extend_as_list**(*original_dict*, ***kwargs*)

benchbuild.utils.download module

Downloading helper functions for benchbuild.

The helpers defined in this module provide access to some common Downloading methods for the source code of benchbuild projects. All downloads will be cached in BB_TMP_DIR and locked down with a hash that is generated after the first download. If the hash matches the file/folder found in BB_TMP_DIR, nothing will be downloaded at all.

Supported methods: Copy, CopyNoFail, Wget, Git, Svn, Rsync

benchbuild.utils.download.**Copy**(*From*, *To*)
Small copy wrapper.

Parameters

- **From** (*str*) – Path to the SOURCE.
- **To** (*str*) – Path to the TARGET.

benchbuild.utils.download.**CopyNoFail**(*src*, *root=None*)
Just copy fName into the current working directory, if it exists.

No action is executed, if fName does not exist. No Hash is checked.

Parameters

- **src** – The filename we want to copy to '.'.
- **root** – The optional source dir we should pull fName from. Defaults to benchbuild.settings.CFG["tmpdir"].

Returns True, if we copied something.

benchbuild.utils.download.**Git**(*repository*, *directory*, *rev=None*, *prefix=None*, *shallow_clone=True*)
Get a clone of the given repo

Parameters

- **repository** (*str*) – Git URL of the SOURCE repo.
- **directory** (*str*) – Name of the repo folder on disk.
- **tgt_root** (*str*) – TARGET folder for the git repo. Defaults to CFG["tmpdir"]

- **shallow_clone** (*bool*) – Only clone the repository shallow Defaults to true

benchbuild.utils.download.**Rsync**(*url*, *tgt_name*, *tgt_root=None*)
RSync a folder.

Parameters

- **url** (*str*) – The url of the SOURCE location.
- **fname** (*str*) – The name of the TARGET.
- **to** (*str*) – Path of the target location. Defaults to `CFG["tmpdir"]`.

benchbuild.utils.download.**Svn**(*url*, *fname*, *to=None*)
Checkout the SVN repo.

Parameters

- **url** (*str*) – The SVN SOURCE repo.
- **fname** (*str*) – The name of the repo on disk.
- **to** (*str*) – The name of the TARGET folder on disk. Defaults to `CFG["tmpdir"]`

benchbuild.utils.download.**Wget**(*src_url*, *tgt_name*, *tgt_root=None*)
Download url, if required.

Parameters

- **src_url** (*str*) – Our SOURCE url.
- **tgt_name** (*str*) – The filename we want to have on disk.
- **tgt_root** (*str*) – The TARGET directory for the download. Defaults to `CFG["tmpdir"]`.

benchbuild.utils.download.**get_hash_of_dirs**(*directory*)
Recursively hash the contents of the given directory.

Parameters **directory** (*str*) – The root directory we want to hash.

Returns A hash of all the contents in the directory.

benchbuild.utils.download.**source_required**(*src_file*)
Check, if a download is required.

Parameters

- **src_file** – The filename to check for.
- **src_root** – The path we find the file in.

Returns True, if we need to download something, False otherwise.

benchbuild.utils.download.**update_hash**(*src_file*)
Update the hash for the given file.

Parameters

- **src** – The file name.
- **root** – The path of the given file.

benchbuild.utils.download.**with_git**(*repo*, *target_dir=None*, *limit=None*, *refspec='HEAD'*, *clone=True*, *rev_list_args=None*)
Decorate a project class with git-based version information.

This adds two attributes to a project class:

- A *versions* method that returns a list of available versions for this project.
- A *repository* attribute that provides a repository string to download from later.

We use the *git rev-list* subcommand to list available versions.

Parameters

- **repo** (*str*) – Repository to download from, this will be stored in the *repository* attribute of the decorated class.
- **target_dir** (*str*) – An optional path where we should put the clone. If unspecified, we will use the *SRC_FILE* attribute of the decorated class.
- **limit** (*int*) – Limit the number of commits to consider for available versions. Versions are 'ordered' from latest to oldest.
- **refspec** (*str*) – A git refspec string to start listing the versions from.
- **clone** (*bool*) – Should we clone the repo if it isn't already available in our tmp dir? Defaults to *True*. You can set this to False to avoid time consuming clones, when the project has not been accessed at least once in your installation.
- **ref_list_args** (*list of str*) – Additional arguments you want to pass to *git rev-list*.

benchbuild.utils.download.**with_wget**(*url_dict=None*, *target_file=None*)

Decorate a project class with wget-based version information.

This adds two attributes to a project class:

- A *versions* method that returns a list of available versions for this project.
- A *repository* attribute that provides a repository string to download from later.

We use the *git rev-list* subcommand to list available versions.

Parameters

- **url_dict** (*dict*) – A dictionary that assigns a version to a download URL.
- **target_file** (*str*) – An optional path where we should put the clone. If unspecified, we will use the *SRC_FILE* attribute of the decorated class.

benchbuild.utils.log module

benchbuild.utils.log.**configure**()

Load logging configuration from our own defaults.

benchbuild.utils.log.**configure_migrate_log**()

benchbuild.utils.log.**configure_parse_log**()

benchbuild.utils.log.**configure_plumbum_log**()

benchbuild.utils.log.**set_defaults**()

Configure the loggers default settings.

benchbuild.utils.path module

Path utilities for benchbuild.

benchbuild.utils.path.**determine_path**()
Borrowed from wxglade.py

benchbuild.utils.path.**list_to_path**(*pathlist*)
Convert a list of path elements to a path string.

benchbuild.utils.path.**mkdir_interactive**(*dirpath*)
Create a directory if required.

This will query the user for a confirmation.

Parameters **dirname** – The path to create.

benchbuild.utils.path.**mkdir_uchroot**(*dirpath*, *root='.'*)
Create a file inside a uchroot env.

You will want to use this when you need to create a file with apropriate rights inside a uchroot container with subuid/subgid handling enabled.

Parameters

- **dirpath** – The dirpath that should be created. Absolute inside the uchroot container.
- **root** – The root PATH of the container filesystem as seen outside of the container.

benchbuild.utils.path.**mkfile_uchroot**(*filepath*, *root='.'*)
Create a file inside a uchroot env.

You will want to use this when you need to create a file with apropriate rights inside a uchroot container with subuid/subgid handling enabled.

Parameters

- **filepath** – The filepath that should be created. Absolute inside the uchroot container.
- **root** – The root PATH of the container filesystem as seen outside of the container.

benchbuild.utils.path.**path_to_list**(*pathstr*)
Conver a path string to a list of path elements.

benchbuild.utils.path.**template_files**(*path*, *exts=None*)
Return a list of filenames found at @path.

The list of filenames can be filtered by extensions.

Parameters

- **path** – Existing filepath we want to list.
- **exts** – List of extensions to filter by.

Returns A list of filenames found in the path.

benchbuild.utils.path.**template_path**(*template*)
Return path to template file.

benchbuild.utils.path.**template_str**(*template*)
Read a template file from the resources and return it as str.

benchbuild.utils.progress module

A progress bar based on the plumbum cli.progress.Progress bar, but with a changed string representation to adjust the design.

class benchbuild.utils.progress.**ProgressBar**(*pg_char='*'*, *iterator=None*, *length=None*, *timer=True*, *body=False*, *has_output=False*, *clear=True*, *value=None*, *width=None*)

Bases: plumbum.cli.progress.ProgressBase

Class that modifies the progress bar.

display()
Completely identical to the Progress class from plumbum.

done()
Completely identical to the Progress class from plumbum.

start()
Completely identical to the Progress class from plumbum.

benchbuild.utils.run module

Experiment helpers.

class benchbuild.utils.run.**RunInfo**(*cmd=None*, *failed=False*, *project=None*, *experiment=None*, *retcode=0*, *stdout=NOTHING*, *stderr=NOTHING*)

Bases: object

Execution context of wrapped binaries.

Execution of tracked binaries is guarded with this context object. In here we store everything about a single binary execution for consumption of an experiment.

cmd

failed

project

experiment

retcode

stdout

stderr

db_run

session

add_payload(*name*, *payload*)

commit()

has_failed
Check, whether this run failed.

benchbuild.utils.run.**begin_run_group**(*project*)
Begin a run_group in the database.

A run_group groups a set of runs for a given project. This models a series of runs that form a complete binary runtime test.

Parameters **project** – The project we begin a new run_group for.

Returns (group, session) where group is the created group in the database and session is the database session this group lives in.

benchbuild.utils.run.**end_run_group**(*group*, *session*)
End the run_group successfully.

Parameters

- **group** – The run_group we want to complete.
- **session** – The database transaction we will finish.

benchbuild.utils.run.**exit_code_from_run_infos**(*run_infos: List[benchbuild.utils.run.RunInfo]*) → int
Generate a single exit code from a list of RunInfo objects.

Takes a list of RunInfos and returns the exit code that is furthest away from 0.

Parameters **run_infos** (*t.List[RunInfo]*) – [description]

Returns [description]

Return type int

benchbuild.utils.run.**fail_run_group**(*group*, *session*)
End the run_group unsuccessfully.

Parameters

- **group** – The run_group we want to complete.
- **session** – The database transaction we will finish.

benchbuild.utils.run.**in_builddir**(*sub='.'*)
Decorate a project phase with a local working directory change.

Parameters **sub** – An optional subdirectory to change into.

benchbuild.utils.run.**run**(*command*, *retcode=0*)
Execute a plumbum command, depending on the user's settings.

Parameters **command** – The plumbumb command to execute.

benchbuild.utils.run.**store_config**(*func*)
Decorator for storing the configuration in the project's builddir.

benchbuild.utils.run.**track_execution**(*cmd*, *project*, *experiment*, ***kwargs*)
Guard the execution of the given command.

The given command (*cmd*) will be executed inside a database context. As soon as you leave the context we will commit the transaction. Any necessary modifications to the database can be identified inside the context with the RunInfo object.

Parameters

- **cmd** – The command we guard.
- **project** – The project we track for.
- **experiment** – The experiment we track for.

Yields *RunInfo* –

A context object that carries the necessary database transaction.

benchbuild.utils.run.**with_env_recursive**(*cmd*, ***envvars*)
Recursively updates the environment of cmd and all its subcommands.

Parameters

- **– A plumbum command-like object** (*cmd*) –
- **– The environment variables to update** (***envvars*) –

Returns The updated command.

benchbuild.utils.schedule_tree module

Parsing utilities for Polly's ScheduleTree representation.

class benchbuild.utils.schedule_tree.**ChildNode**(*tok*)
Bases: *benchbuild.utils.schedule_tree.Node*

indent(*level=0*, *idt=' '*)

class benchbuild.utils.schedule_tree.**CoincidenceNode**(*tok*)
Bases: *benchbuild.utils.schedule_tree.Node*

indent(*level=0*, *idt=' '*)

class benchbuild.utils.schedule_tree.**Node**(*tok*)
Bases: object

indent(*level=0*, *idt=' '*)

class benchbuild.utils.schedule_tree.**RootNode**(*tok*)
Bases: *benchbuild.utils.schedule_tree.Node*

indent(*level=0*, *idt=' '*)

class benchbuild.utils.schedule_tree.**SequenceNode**(*tok*)
Bases: *benchbuild.utils.schedule_tree.Node*

indent(*level=0*, *idt=' '*)

benchbuild.utils.schedule_tree.**parse_schedule_tree**(*tree_str*)

benchbuild.utils.schema module

Database schema for benchbuild

The schema should initialize itself on an empty database. For now, we do not support automatic upgrades on schema changes. You might encounter some roadbumps when using an older version of benchbuild.

Furthermore, for now, we are restricted to postgresql databases, although we already support arbitrary connection strings via config.

If you want to use reports that use one of our SQL functions, you need to initialize the functions first using the following command:

“‘**bash** > BB_DB_CREATE_FUNCTIONS=true benchbuild run -E empty -l

“‘

After that you (normally) do not need to do this agains, unless we supply a new version that you are interested in. As soon as we have alembic running, we can provide automatic up/downgrade paths for you.

class benchbuild.utils.schema.**Config**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store customized information about a run.

You can store arbitrary configuration information about a run here. Use it for extended filtering against the run table.

name

run_id

value

class benchbuild.utils.schema.**Experiment**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store metadata about experiments.

begin

description

end

id

name

run_groups

runs

class benchbuild.utils.schema.**GUID**(**args*, *as_uuid=False*, ***kwargs*)
Bases: sqlalchemy.sql.type_api.TypeDecorator

Platform-independent GUID type.

Uses Postgresql's UUID type, otherwise uses CHAR(32), storing as stringified hex values.

as_uuid = False

impl
alias of sqlalchemy.sql.sqltypes.CHAR

load_dialect_impl(*dialect*)
Return a TypeEngine object corresponding to a dialect.

This is an end-user override hook that can be used to provide differing types depending on the given dialect. It is used by the TypeDecorator implementation of type_engine() to help determine what type should ultimately be returned for a given TypeDecorator.

By default returns self.impl.

process_bind_param(*value*, *dialect*)
Receive a bound parameter value to be converted.

Subclasses override this method to return the value that should be passed along to the underlying TypeEngine object, and from there to the DBAPI execute() method.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

This operation should be designed with the reverse operation in mind, which would be the process_result_value method of this class.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be None.
- **dialect** – the Dialect in use.

process_result_value(*value*, *dialect*)
Receive a result-row column value to be converted.

Subclasses should implement this method to operate on data fetched from the database.

Subclasses override this method to return the value that should be passed back to the application, given a value that is already processed by the underlying `TypeEngine` object, originally from the DBAPI cursor method `fetchone()` or similar.

The operation could be anything desired to perform custom behavior, such as transforming or serializing data. This could also be used as a hook for validating logic.

Parameters

- **value** – Data to operate upon, of any type expected by this method in the subclass. Can be `None`.
- **dialect** – the `Dialect` in use.

This operation should be designed to be reversible by the "process_bind_param" method of this class.

class benchbuild.utils.schema.**Metadata**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

Store metadata information for every run.

If you happen to have some free-form data that belongs to the database, this is the place for it.

name

run_id

value

class benchbuild.utils.schema.**Metric**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

Store default metrics, simple name value store.

name

run_id

value

class benchbuild.utils.schema.**Project**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

Store project metadata.

description

domain

group_name

name

runs

src_url

version

class benchbuild.utils.schema.**Run**(***kwargs*)
Bases: `sqlalchemy.ext.declarative.api.Base`

Store a run for each executed test binary.

begin

command

configurations

end

experiment_group

experiment_name

id

logs

metrics

project_group

project_name

run_group

status

stored_data

class benchbuild.utils.schema.**RunGroup**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store information about a run group.

begin

end

experiment

id

status

class benchbuild.utils.schema.**RunLog**(***kwargs*)
Bases: sqlalchemy.ext.declarative.api.Base

Store log information for every run.

Properties like, start time, finish time, exit code, stderr, stdout are stored here.

begin

config

end

run_id

status

stderr

stdout

benchbuild.utils.schema.**Session**()

class benchbuild.utils.schema.**SessionManager**
Bases: object

configure_engine()
Configure the databse connection.

Sets appropriate transaction isolation levels and handle errors.

Returns True, if we did not encounter any unrecoverable errors, else False.

connect_engine()
Establish a connection to the database.

Provides simple error handling for fatal errors.

Returns True, if we could establish a connection, else False.

get()

benchbuild.utils.schema.**enforce_versioning**(*force=False*)
Install versioning on the db.

benchbuild.utils.schema.**exceptions**(*error_is_fatal=True*, *error_messages=None*)
Handle SQLAlchemy exceptions in a sane way.

Parameters

- **func** – An arbitrary function to wrap.
- **error_is_fatal** – Should we exit the program on exception?
- **reraise** – Should we reraise the exception, after logging? Only makes sense if error_is_fatal is False.
- **error_messages** – A dictionary that assigns an exception class to a customized error message.

benchbuild.utils.schema.**get_version_data**()
Retreive migration information.

benchbuild.utils.schema.**init_functions**(*connection*)
Initialize all SQL functions in the database.

benchbuild.utils.schema.**maybe_update_db**(*repo_version*, *db_version*)

benchbuild.utils.schema.**metadata**()

benchbuild.utils.schema.**needed_schema**(*connection*, *meta*)

benchbuild.utils.schema.**setup_versioning**()

benchbuild.utils.slurm module

SLURM support for the benchbuild study.

This module can be used to generate bash scripts that can be executed by the SLURM controller either as batch or interactive script.

benchbuild.utils.slurm.**script**(*experiment*, *projects*)
Prepare a slurm script that executes the experiment for a given project.

Parameters

- **experiment** – The experiment we want to execute
- **projects** – All projects we generate an array job for.

benchbuild.utils.uchroot module

class benchbuild.utils.uchroot.**UchrootEC**
Bases: enum.Enum

An enumeration.

MNT_DEV_FAILED = 253

MNT_FAILED = 255

MNT_PROC_FAILED = 254

MNT_PTS_FAILED = 251

MNT_SYS_FAILED = 252

benchbuild.utils.uchroot.**clean_env**(*uchroot_cmd*, *varnames*)
Returns a uchroot cmd that runs inside a filtered environment.

benchbuild.utils.uchroot.**env**(*mounts*)
Compute the environment of the change root for the user.

Parameters **mounts** – The mountpoints of the current user.

Returns paths ld_libs

benchbuild.utils.uchroot.**mounts**(*prefix*, *__mounts*)
Compute the mountpoints of the current user.

Parameters

- **prefix** – Define where the job was running if it ran on a cluster.
- **mounts** – All mounts the user currently uses in his file system.

Returns mntpoints

benchbuild.utils.uchroot.**no_args**(***kwargs*)
Return the uchroot command without any customizations.

benchbuild.utils.uchroot.**no_llvm**(**args*, *uid=0*, *gid=0*, ***kwargs*)
Return a customizable uchroot command.

The command will be executed inside a uchroot environment.

Parameters **args** – List of additional arguments for uchroot (typical: mounts)

Returns chroot_cmd

benchbuild.utils.uchroot.**retry**(*pb_cmd*, *retries=0*, *max_retries=10*, *retcode=0*, *retry_retcodes=None*)

benchbuild.utils.uchroot.**uchroot**(**args*, ***kwargs*)
Return a customizable uchroot command.

Parameters **args** – List of additional arguments for uchroot (typical: mounts)

Returns chroot_cmd

benchbuild.utils.uchroot.**uretry**(*cmd*, *retcode=0*)

benchbuild.utils.uchroot.**with_mounts**(**args*, *uchroot_cmd_fn=<function no_args>*, ***kwargs*)
Return a uchroot command with all mounts enabled.

benchbuild.utils.unionfs module

exception benchbuild.utils.unionfs.**UnmountError**
Bases: BaseException

benchbuild.utils.unionfs.**unionfs**(*rw='rw'*, *ro=None*, *union='union'*)
Decorator for the UnionFS feature.

This configures a unionfs for projects. The given base_dir and/or image_dir are layered as follows:

image_dir=RW:base_dir=RO

All writes go to the image_dir, while base_dir delivers the (read-only) versions of the rest of the filesystem.

The unified version will be provided in the project's builddir. Unmouting is done as soon as the function completes.

Parameters

- **rw** – writeable storage area for the unified fuse filesystem.
- **ro** – read-only storage area for the unified fuse filesystem.
- **union** – mountpoint of the unified fuse filesystem.

benchbuild.utils.user_interface module

User interface helpers for benchbuild.

benchbuild.utils.user_interface.**ask**(*question*, *default_answer=False*, *default_answer_str='no'*)
Ask for user input.

This asks a yes/no question with a preset default. You can bypass the user-input and fetch the default answer, if you set

Parameters

- **question** – The question to ask on stdout.
- **default_answer** – The default value to return.
- **default_answer_str** – The default answer string that we present to the user.

Tests:

```
>>> os.putenv("TEST", "yes"); ask("Test?", default_answer=True)
True
>>> os.putenv("TEST", "yes"); ask("Test?", default_answer=False)
False
```

benchbuild.utils.user_interface.**query_yes_no**(*question*, *default='yes'*)
Ask a yes/no question via raw_input() and return their answer.

Parameters

- **question** (*str*) – Question hat is presented to the user.
- **default** (*str*) – The presumed answer, if the user just hits <Enter>. It must be "yes" (the default), "no" or None (meaning an answer is required of the user).

Returns (boolean): True, if 'yes', False otherwise.

benchbuild.utils.versions module

Gather version information for BB.

benchbuild.utils.versions.**get_git_hash**(*from_url*)
Get the git commit hash of HEAD from :from_url.

Parameters **from_url** – The file system url of our git repository.

Returns git commit hash of HEAD, or empty string.

benchbuild.utils.versions.**get_version_from_cache_dir**(*src_file*)
Creates a version for a project out of the hash.

The hash is taken from the directory of the source file.

Parameters **src_file** – The source file of the project using this function.

Returns Either returns the first 8 digits of the hash as string, the entire hash as a string if the hash consists out of less than 7 digits or None if the path is incorrect.

benchbuild.utils.wrapping module

Wrapper utilities for benchbuild.

This module provides methods to wrap binaries with extensions that are pickled alongside the original binary. In place of the original binary a new python module is generated that loads the pickle and redirects the program call with all its arguments to it. This allows interception of arbitrary programs for experimentation.

Examples

TODO

Compiler Wrappers: The compiler wrappers substitute the compiler call with a script that produces the expected output from the original compiler call first. Afterwards the pickle is loaded and the original call is forwarded to the pickle. This way the user is not obligated to produce valid output during his own experiment.

Runtime Wrappers: These directly forward the binary call to the pickle without any execution of the binary. We cannot guarantee that repeated execution is valid, therefore, we let the user decide what the program should do.

benchbuild.utils.wrapping.**load**(*filename*)
Load a pickled obj from the filesystem.

You better know what you expect from the given pickle, because we don't check it.

Parameters **filename** (*str*) – The filename we load the object from.

Returns The object we were able to unpickle, else None.

benchbuild.utils.wrapping.**persist**(*id_obj*, *filename=None*, *suffix=None*)
Persist an object in the filesystem.

This will generate a pickled version of the given obj in the filename path. Objects shall provide an id() method to be able to use this persistence API. If not, we will use the id() builtin of python to generate an identifier for you.

The file will be created, if it does not exist. If the file already exists, we will overwrite it.

Parameters **id_obj** (Any) – An identifiable object you want to persist in the filesystem.

benchbuild.utils.wrapping.**strip_path_prefix**(*ipath*, *prefix*)
Strip prefix from path.

Parameters

- **ipath** – input path
- **prefix** – the prefix to remove, if it is found in :ipath:

Examples

```
>>> strip_path_prefix("/foo/bar", "/bar")
'/foo/bar'
>>> strip_path_prefix("/foo/bar", "/")
'foo/bar'
>>> strip_path_prefix("/foo/bar", "/foo")
'/bar'
>>> strip_path_prefix("/foo/bar", "None")
'/foo/bar'
```

benchbuild.utils.wrapping.**unpickle**(*pickle_file*)
Unpickle a python object from the given path.

benchbuild.utils.wrapping.**wrap**(*name*, *project*, *sprefix=None*, *python='/home/docs/checkouts/readthedocs.org/user_builds/pprof-study/envs/master/bin/python'*)
Wrap the binary :name: with the runtime extension of the project.

This module generates a python tool that replaces :name: The function in runner only accepts the replaced binaries name as argument. We use the cloudpickle package to perform the serialization, make sure :runner: can be serialized with it and you're fine.

Parameters

- **name** – Binary we want to wrap
- **project** – The project that contains the runtime_extension we want to run instead of the binary.

Returns A plumbum command, ready to launch.

benchbuild.utils.wrapping.**wrap_cc**(*filepath*, *compiler*, *project*, *python='/home/docs/checkouts/readthedocs.org/user_builds/pprof-study/envs/master/bin/python'*, *detect_project=False*)
Substitute a compiler with a script that hides CFLAGS & LDFLAGS.

This will generate a wrapper script in the current directory and return a complete plumbum command to it.

Parameters

- **filepath** (*str*) – Path to the wrapper script.
- **compiler** (*benchbuild.utils.cmd*) – Real compiler command we should call in the script.
- **project** (benchbuild.project.Project) – The project this compiler will be for.
- **python** (*str*) – Path to the python interpreter we should use.
- **detect_project** – Should we enable project detection or not.

Returns (benchbuild.utils.cmd): Command of the new compiler we can call.

benchbuild.utils.wrapping.**wrap_dynamic**(*project*, *name*, *sprefix=None*, *python='/home/docs/checkouts/readthedocs.org/user_builds/pprof-study/envs/master/bin/python'*, *name_filters=None*)

Wrap the binary :name with the function :runner.

This module generates a python tool :name: that can replace a yet unspecified binary. It behaves similar to the :wrap: function. However, the first argument is the actual binary name.

Parameters

- **name** – name of the python module
- **runner** – Function that should run the real binary
- **sprefix** – Prefix that should be used for commands.
- **python** – The python executable that should be used.
- **name_filters** – List of regex expressions that are used to filter the real project name. Make sure to include a match group named 'name' in the regex, e.g., [

  r'foo(?P<name>.)-flt'

  ]

Returns: plumbum command, readty to launch.

6.1.2 Submodules

benchbuild.container module

Container construction tool.

This tool assists in the creation of customized uchroot containers. You can define strategies and apply them on a given container base-image to have a fixed way of creating a user-space environment.

class benchbuild.container.**BashStrategy**

Bases: *benchbuild.container.ContainerStrategy*

The user interface for setting up a bash inside the container.

run(*context*)

Execute a container strategy.

Parameters **context** – A context object with attributes used for the strategy.

class benchbuild.container.**Container**(*executable*)

Bases: plumbum.cli.application.Application

Manage uchroot containers.

VERSION = '3.4.2.dev3+g0712eee'

builddir(*tmpdir*)

Set the current builddir of the container.

input_file(*_container*)

Find the input path of a uchroot container.

main(**args*)

Implement me (no need to call super)

mounts(*user_mount*)

Save the current mount of the container into the settings.

output_file(*_container*)
Find and writes the output path of a chroot container.

shell(*custom_shell*)
The command to run inside the container.

verbosity
Sets an attribute

class benchbuild.container.**ContainerBootstrap**(*executable*)
Bases: plumbum.cli.application.Application

Check for the needed files.

install_cmake_and_exit()
Tell the user to install cmake and aborts the current process.

main(**args*)
Implement me (no need to call super)

class benchbuild.container.**ContainerCreate**(*executable*)
Bases: plumbum.cli.application.Application

Create a new container with a predefined strategy.

We offer a variety of creation policies for a new container. By default a basic 'spawn a bash' policy is used. This just leaves you inside a bash that is started in the extracted container. After customization you can exit the bash and pack up the result.

main(**args*)
Implement me (no need to call super)

strategy(*strategy*)
Select strategy based on key.

Parameters **strategy** (*str*) – The strategy to select.

Returns A strategy object.

class benchbuild.container.**ContainerList**(*executable*)
Bases: plumbum.cli.application.Application

Prints a list of the known containers.

main(**args*)
Implement me (no need to call super)

class benchbuild.container.**ContainerRun**(*executable*)
Bases: plumbum.cli.application.Application

Execute commannds inside a prebuilt container.

main(**args*)
Implement me (no need to call super)

class benchbuild.container.**ContainerStrategy**
Bases: object

Interfaces for the different containers chosen by the experiment.

run(*context*)
Execute a container strategy.

Parameters **context** – A context object with attributes used for the strategy.

class benchbuild.container.**MockObj**(***kwargs*)
Bases: object

Context object to be used in strategies.

This object's attributes are initialized on construction.

class benchbuild.container.**SetupPolyJITGentooStrategy**
Bases: *benchbuild.container.ContainerStrategy*

Interface of using gentoo as a container for an experiment.

run(*context*)
Setup a gentoo container suitable for PolyJIT.

benchbuild.container.**clean_directories**(*builddir*, *in_dir=True*, *out_dir=True*)
Remove the in and out of the container if confirmed by the user.

benchbuild.container.**find_hash**(*container_db*, *key*)
Find the first container in the database with the given key.

benchbuild.container.**main**(**args*)
Main entry point for the container tool.

benchbuild.container.**pack_container**(*in_container*, *out_file*)
Pack a container image into a .tar.bz2 archive.

Parameters

- **in_container** (*str*) – Path string to the container image.
- **out_file** (*str*) – Output file name.

benchbuild.container.**run_in_container**(*command*, *container_dir*)
Run a given command inside a container.

Mounts a directory as a container at the given mountpoint and tries to run the given command inside the new container.

benchbuild.container.**set_input_container**(*_container*, *cfg*)
Save the input for the container in the configurations.

benchbuild.container.**setup_bash_in_container**(*builddir*, *_container*, *outfile*, *shell*)
Setup a bash environment inside a container.

Creates a new chroot, which the user can use as a bash to run the wanted projects inside the mounted container, that also gets returned afterwards.

benchbuild.container.**setup_container**(*builddir*, *_container*)
Prepare the container and returns the path where it can be found.

benchbuild.container.**setup_directories**(*builddir*)
Create the in and out directories of the container.

benchbuild.driver module

benchbuild.driver.**main**(**args*)
Main function.

benchbuild.likwid module

Likwid helper functions.

Extract information from likwid's CSV output.

benchbuild.likwid.**fetch_cols**(*fstream*, *split_char=','*)
Fetch columns from likwid's output stream.

Parameters

- **fstream** – The filestream with likwid's output.
- **split_car** (*str*) – The character we split on, default ','

Returns (list(str)): A list containing the elements of fstream, after splitting at split_char.

benchbuild.likwid.**get_measurements**(*region*, *core_info*, *data*, *extra_offset=0*)
Get the complete measurement info from likwid's region info.

Parameters

- **region** – The region we took a measurement in.
- **core_info** – The core information.
- **data** – The raw data.
- **extra_offset** (*int*) – default = 0

Returns (list((region, metric, core, value))): A list of measurement tuples, a tuple contains the information about the region, the metric, the core and the actual value.

benchbuild.likwid.**perfcounters**(*infile*)
Get a complete list of all measurements.

Parameters **infile** – The filestream containing all likwid output.

Returns A list of all measurements extracted from likwid's file stream.

benchbuild.likwid.**read_struct**(*fstream*)
Read a likwid struct from the text stream.

Parameters **fstream** – Likwid's filestream.

Returns (dict(str: str)): A dict containing all likwid's struct info as key/value pairs.

benchbuild.likwid.**read_structs**(*fstream*)
Read all structs from likwid's file stream.

Parameters **fstream** – Likwid's output file stream.

Returns A generator that can be used to iterate over all structs in the fstream.

benchbuild.likwid.**read_table**(*fstream*)
Read a likwid table info from the text stream.

Parameters **fstream** – Likwid's filestream.

Returns (dict(str: str)): A dict containing likwid's table info as key/value pairs.

benchbuild.likwid.**read_tables**(*fstream*)
Read all tables from likwid's file stream.

Parameters **fstream** – Likwid's output file stream.

Returns A generator that can be used to iterate over all tables in the fstream.

benchbuild.settings module

Settings module for benchbuild.

All settings are stored in a simple dictionary. Each setting should be modifiable via environment variable.

benchbuild.signals module

class benchbuild.signals.**CleanupOnSignal**

Bases: object

deregister(*callback*)

register(*callback*, **args*, ***kwargs*)

stored_procedures

benchbuild.statistics module

Handle all statsitic related classes and methods.

class benchbuild.statistics.**Statistics**(*project*, *experiment*, **extensions*, *config=None*)

Bases: *benchbuild.extensions.base.Extension*

Extend a run to be repeated until it reaches a statistically significance specified by the user.

An example on how to use this extension can be found in the Pollytest Experiment.

t_test(**results*, *significance=0.95*)

Runs a t-test on a given set of results.

Returns True if the null hypothesis that the result was not significant was rejected, False otherwise.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

A

B

C

D

E

F

G

H

I

N

O

P

Q

R

S

T

U

V

W

X

Y