haas Documentation

Release 0.9.0.dev652

Simon Jagoe

Feb 04, 2020

Contents

haas is a Python test runner that is backward-compatible with Python's built-in unittest Test Cases, but is designed to support more advanced features, such as project-specific plugins.

CHAPTER 1

Introduction

haas is intended to iron out some of the wrinkles in Python's unittest test runner. haas features an improved test discover and loading mechanism, allowing the same base command to be used to discover tests in packages as well as run individual test cases. In the future, haas will feature a plugin system allowing the use of environment configuration plugins (e.g. configure Windows SxS manifests or COM before running tests) or even plugins that run code between tests (e.g. report on threads that are not cleaned up by code under test).

Unlike unittest, haas is also usually safe to use within a project's source tree as it features more robust detection of the top-level directory of a project.

Contents:

1.1 Getting Started

1.1.1 Installing haas

haas can be easily installed using pip:

```
$ pip install haas
```

For development versions, the source is available from the GitHub repository. To install haas from GitHub, clone the repository and install using pip:

```
$ git clone https://github.com/sjagoe/haas.git
$ cd haas
$ python setup.py sdist
$ pip install dist/haas*.egg
```

1.1.2 Using haas

To use the basic test discovery feature of haas, simply invoke it at the top-level of a project; this should be enough to detect and run all unit tests:

```
$ haas
.....................................................................
.............................
----------------------------------------------------------------------
Ran 103 tests in 0.116s

OK
```

haas supports some of the same options as Python's unittest runner, with one major difference: the starting directory (or package name) for test discovery is simply specified on the command line:

```
$ haas haas/tests
.....................................................................
.............................
----------------------------------------------------------------------
Ran 103 tests in 0.116s

OK
$ haas haas.tests
.....................................................................
.............................
----------------------------------------------------------------------
Ran 103 tests in 0.116s

OK
```

For the currently available options, use the --help option:

```
$ haas --help
usage: haas [-h] [-v | -q] [-f] [-b] [-p PATTERN]
            [-t TOP_LEVEL_DIRECTORY]
            [--log-level {critical,fatal,error,warning,info,debug}]
            [start]

positional arguments:
  start                 Directory or dotted package/module name to start
                        searching for tests

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Verbose output
  -q, --quiet           Quiet output
  -f, --failfast        Stop on first fail or error
  -b, --buffer          Buffer stdout and stderr during tests
  -p PATTERN, --pattern PATTERN
                        Pattern to match tests ('test*.py' default)
  -t TOP_LEVEL_DIRECTORY, --top-level-directory TOP_LEVEL_DIRECTORY
                        Top level directory of project (defaults to start
                        directory)
  --log-level {critical,fatal,error,warning,info,debug}
                        Log level for haas logging
```

Discovering tests by a test name only

haas is able to discover a subset of the tests when just a test name (or any sub-section of a dotted module name) is given on the command line.

For example, to run all test methods called `test_method`, the following would work:

```
$ haas -v test_method
test_method (haas.tests._test_cases.TestCase) ... ok
test_method (haas.tests.test_loader.TestCaseSubclass) ... ok
test_method (haas.tests.test_discoverer.FilterTestCase) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Note that three tests in three different modules have been run. The string `test_method` is matched at any point in the name `<package>.<module>.<class>.<method>` and across all loadable tests in the project. To restrict this to a single test, we can use the class name as an additional matching parameter:

```
$ haas -v FilterTestCase.test_method
test_method (haas.tests.test_discoverer.FilterTestCase) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

1.2 API Reference

1.2.1 haas package

Subpackages

haas.plugins package

Submodules

haas.plugins.base_hook_plugin module

class haas.plugins.base_hook_plugin.**BaseHookPlugin**(*name*, *enabled*, *enabling_option*)

Bases: *haas.plugins.i_hook_plugin.IHookPlugin*

The base implementation of hook plugins.

classmethod add_parser_arguments(*parser*, *name*, *option_prefix*, *dest_prefix*)

classmethod from_args(*args*, *name*, *dest_prefix*)

haas.plugins.coverage module

class haas.plugins.coverage.**Coverage**(**args*, ***kwargs*)

Bases: *haas.plugins.base_hook_plugin.BaseHookPlugin*

setup()

teardown()

haas.plugins.discoverer module

class haas.plugins.discoverer.**Discoverer**(*loader*, ***kwargs*)
Bases: haas.plugins.i_discoverer_plugin.IDiscovererPlugin

The Discoverer is responsible for finding tests that can be loaded by a *Loader*.

classmethod add_parser_arguments(*parser*, *option_prefix*, *dest_prefix*)
Add options for the plugin to the main argument parser.

Parameters

- **parser** (*argparse.ArgumentParser*) – The parser to extend
- **option_prefix** (*str*) – The prefix that option strings added by this plugin should use.
- **dest_prefix** (*str*) – The prefix that dest strings for options added by this plugin should use.

discover(*start*, *top_level_directory=None*, *pattern=u'test*.py'*)
Do test case discovery.

This is the top-level entry-point for test discovery.

If the start argument is a drectory, then haas will discover all tests in the package contained in that directory.

If the start argument is not a directory, it is assumed to be a package or module name and tests in the package or module are loaded.

FIXME: This needs a better description.

Parameters

- **start** (*str*) – The directory, package, module, class or test to load.
- **top_level_directory** (*str*) – The path to the top-level directoy of the project. This is the parent directory of the project'stop-level Python package.
- **pattern** (*str*) – The glob pattern to match the filenames of modules to search for tests.

discover_by_directory(*start_directory*, *top_level_directory=None*, *pattern=u'test*.py'*)
Run test discovery in a directory.

Parameters

- **start_directory** (*str*) – The package directory in which to start test discovery.
- **top_level_directory** (*str*) – The path to the top-level directoy of the project. This is the parent directory of the project'stop-level Python package.
- **pattern** (*str*) – The glob pattern to match the filenames of modules to search for tests.

discover_by_file(*start_filepath*, *top_level_directory=None*)
Run test discovery on a single file.

Parameters

- **start_filepath** (*str*) – The module file in which to start test discovery.
- **top_level_directory** (*str*) – The path to the top-level directoy of the project. This is the parent directory of the project'stop-level Python package.

discover_by_module(*module_name*, *top_level_directory=None*, *pattern=u'test*.py'*)
Find all tests in a package or module, or load a single test case if a class or test inside a module was specified.

Parameters

- **module_name** (*str*) – The dotted package name, module name or TestCase class and test method.
- **top_level_directory** (*str*) – The path to the top-level directoy of the project. This is the parent directory of the project'stop-level Python package.
- **pattern** (*str*) – The glob pattern to match the filenames of modules to search for tests.

discover_filtered_tests(*filter_name*, *top_level_directory=None*, *pattern=u'test*.py'*)
Find all tests whose package, module, class or method names match the `filter_name` string.

Parameters

- **filter_name** (*str*) – A subsection of the full dotted test name. This can be simply a test method name (e.g. `test_some_method`), the TestCase class name (e.g. `TestMyClass`), a module name (e.g. `test_module`), a subpackage (e.g. `tests`). It may also be a dotted combination of the above (e.g. `TestMyClass.test_some_method`).
- **top_level_directory** (*str*) – The path to the top-level directoy of the project. This is the parent directory of the project'stop-level Python package.
- **pattern** (*str*) – The glob pattern to match the filenames of modules to search for tests.

discover_single_case(*module*, *case_attributes*)
Find and load a single TestCase or TestCase method from a module.

Parameters

- **module** (*module*) – The imported Python module containing the TestCase to be loaded.
- **case_attributes** (*list*) – A list (length 1 or 2) of str. The first component must be the name of a TestCase subclass. The second component must be the name of a method in the TestCase.

classmethod from_args(*args*, *arg_prefix*, *loader*)
Construct the discoverer from parsed command line arguments.

Parameters

- **args** (*argparse.Namespace*) – The `argparse.Namespace` containing parsed arguments.
- **arg_prefix** (*str*) – The prefix used for arguments beloning solely to this plugin.
- **loader** (`haas.loader.Loader`) – The test loader used to construct TestCase and TestSuite instances.

haas.plugins.discoverer.**assert_start_importable**(*top_level_directory*, *start_directory*)

haas.plugins.discoverer.**filter_test_suite**(*suite*, *filter_name*)
Filter test cases in a test suite by a substring in the full dotted test name.

Parameters

- **suite** (`haas.suite.TestSuite`) – The test suite containing tests to be filtered.
- **filter_name** (*str*) – The substring of the full dotted name on which to filter. This should not contain a leading or trailing dot.

haas.plugins.discoverer.**find_module_by_name**(*full_name*)

haas.plugins.discoverer.**find_top_level_directory**(*start_directory*)
Finds the top-level directory of a project given a start directory inside the project.

Parameters **start_directory** (*str*) – The directory in which test discovery will start.

haas.plugins.discoverer.**get_module_name**(*top_level_directory*, *filepath*)

haas.plugins.discoverer.**get_relpath**(*top_level_directory*, *fullpath*)

haas.plugins.discoverer.**match_path**(*filename*, *filepath*, *pattern*)

haas.plugins.i_hook_plugin module

class haas.plugins.i_hook_plugin.**IHookPlugin**
Bases: object

classmethod add_parser_arguments(*parser*, *name*, *option_prefix*, *dest_prefix*)

classmethod from_args(*args*, *dest_prefix*)

setup()

teardown()

haas.plugins.i_result_handler_plugin module

class haas.plugins.i_result_handler_plugin.**IResultHandlerPlugin**
Bases: object

classmethod add_parser_arguments(*parser*, *name*, *option_prefix*, *dest_prefix*)
Add options for the plugin to the main argument parser.

Parameters

- **parser** (*argparse.ArgumentParser*) – The parser to extend
- **name** (*str*) – The name of the plugin.
- **option_prefix** (*str*) – The prefix that option strings added by this plugin should use.
- **dest_prefix** (*str*) – The prefix that dest strings for options added by this plugin should use.

classmethod from_args(*args*, *name*, *dest_prefix*, *test_count*)
Construct the result handler from parsed command line arguments.

Parameters

- **args** (*argparse.Namespace*) – The argparse.Namespace containing parsed arguments.
- **name** (*str*) – The name of the plugin.
- **dest_prefix** (*str*) – The prefix that dest strings for options added by this plugin should use.
- **test_count** (*int*) – The totel number of tests discovered.

start_test(*test*)
Perform tasks at the start of a single test.

start_test_run()
Perform tasks at the very start of the test run.

stop_test(*test*)
Perform tasks at the end of a single test.

stop_test_run()
Perform tasks at the very end of the test run.

haas.plugins.i_runner_plugin module

class haas.plugins.i_runner_plugin.**IRunnerPlugin**
Bases: object

classmethod add_parser_arguments(*parser*, *option_prefix*, *dest_prefix*)
Add options for the plugin to the main argument parser.

Parameters

- **parser** (*argparse.ArgumentParser*) – The parser to extend
- **option_prefix** (*str*) – The prefix that option strings added by this plugin should use.
- **dest_prefix** (*str*) – The prefix that dest strings for options added by this plugin should use.

classmethod from_args(*args*, *arg_prefix*)
Construct the runner from parsed command line arguments.

Parameters

- **args** (*argparse.Namespace*) – The argparse.Namespace containing parsed arguments.
- **arg_prefix** (*str*) – The prefix used for arguments beloning solely to this plugin.

haas.plugins.result_handler module

class haas.plugins.result_handler.**QuietTestResultHandler**(*test_count*)
Bases: *haas.plugins.i_result_handler_plugin.IResultHandlerPlugin*

classmethod add_parser_arguments(*parser*, *name*, *option_prefix*, *dest_prefix*)

classmethod from_args(*args*, *name*, *dest_prefix*, *test_count*)

get_test_description(*test*)

print_error_list(*error_kind*, *errors*)
Print the list of errors or failures.

Parameters

- **error_kind** (*str*) – 'ERROR' or 'FAIL'
- **errors** (*list*) – List of *TestResult*

print_errors()
Print all errors and failures to the console.

print_summary()

separator1 = u'=='

separator2 = u'--'

start_test(*test*)

start_test_run()

stop_test(*test*)

stop_test_run()

was_successful()

class haas.plugins.result_handler.**StandardTestResultHandler**(*test_count*)
Bases: *haas.plugins.result_handler.QuietTestResultHandler*

classmethod from_args(*args*, *name*, *dest_prefix*, *test_count*)

class haas.plugins.result_handler.**TimingResultHandler**(*number_to_summarize*)
Bases: *haas.plugins.i_result_handler_plugin.IResultHandlerPlugin*

OPTION_DEFAULT = <object object>

classmethod add_parser_arguments(*parser*, *name*, *option_prefix*, *dest_prefix*)

classmethod from_args(*args*, *name*, *dest_prefix*, *test_count*)

print_summary()

separator1 = u'==='

separator2 = u'---'

start_test(*test*)

start_test_run()

stop_test(*test*)

stop_test_run()

class haas.plugins.result_handler.**VerboseTestResultHandler**(*test_count*)
Bases: *haas.plugins.result_handler.StandardTestResultHandler*

classmethod from_args(*args*, *name*, *dest_prefix*, *test_count*)

start_test(*test*)

haas.plugins.result_handler.**get_test_description**(*test*, *descriptions=True*)

haas.plugins.result_handler.**sort_result_handlers**(*handlers*)

haas.plugins.runner module

class haas.plugins.runner.**BaseTestRunner**(*warnings=None*)
Bases: *haas.plugins.i_runner_plugin.IRunnerPlugin*

A test runner class that does not print any output itself.

classmethod add_parser_arguments(*parser*, *option_prefix*, *dest_prefix*)

classmethod from_args(*args*, *arg_prefix*)

run(*result_collector*, *test*)
Run the given test case or test suite.

haas.tests package

Submodules

haas.tests.builder module

class haas.tests.builder.**Class**(*name*, *contents=()*, *bases=(<class 'unittest.case.TestCase'>,)*)
Bases: *haas.tests.builder.Importable*

create(*module_fh*)

class haas.tests.builder.**Directory**(*name*, *contents=()*)
Bases: *haas.tests.builder.Importable*

create(*parent_importable*)

class haas.tests.builder.**Importable**(*name*, *contents=()*)
Bases: object

create(*parent_importable*)
Create the importable object.

class haas.tests.builder.**Method**(*name*, *contents=()*)
Bases: *haas.tests.builder.Importable*

create(*module_fh*)

class haas.tests.builder.**Module**(*name*, *contents=()*)
Bases: *haas.tests.builder.Importable*

create(*parent_importable*)

class haas.tests.builder.**Package**(*name*, *contents=()*)
Bases: *haas.tests.builder.Directory*

class haas.tests.builder.**RawText**(*name*, *contents=u''*)
Bases: *haas.tests.builder.Importable*

create(*module_fh*)

Submodules

haas.error_holder module

class haas.error_holder.**ErrorHolder**(*description*)
Bases: object

Placeholder for a TestCase inside a result. As far as a TestResult is concerned, this looks exactly like a unit test. Used to insert arbitrary errors into a test suite run.

countTestCases()

failureException = None

id()

run(*result*)

shortDescription()

haas.exceptions module

exception haas.exceptions.**DotInModuleNameError**
Bases: *haas.exceptions.HaasException*

exception haas.exceptions.**HaasException**
Bases: exceptions.Exception

exception haas.exceptions.**PluginError**
Bases: *haas.exceptions.HaasException*

haas.haas_application module

class haas.haas_application.**HaasApplication**(*argv*, ***kwargs*)
Bases: object

Main haas application entry-point.

run(*plugin_manager=None*)
Run the haas test runner.

This will load and configure the selected plugins, set up the environment and begin test discovery, loading and running.

Parameters **plugin_manager** (haas.plugin_manager.PluginManager) – [Optional] Override the use of the default plugin manager.

haas.haas_application.**create_argument_parser**()
Creates the argument parser for haas.

haas.loader module

class haas.loader.**Loader**(*test_suite_class=None*, *test_case_class=None*, *test_method_prefix=u'test'*, ***kwargs*)
Bases: object

Load individual test cases from modules and wrap them in the Suite container.

create_suite(*tests=()*)
Create a test suite using the confugured test suite class.

Parameters **tests** (*sequence*) – Sequence of TestCase instances.

find_test_method_names(*testcase*)
Return a list of test method names in the provided TestCase subclass.

Parameters **testcase** (*type*) – Subclass of unittest.TestCase

get_test_cases_from_module(*module*)
Return a list of TestCase subclasses contained in the provided module object.

Parameters **module** (*module*) – A module object containing TestCases

is_test_case(*klass*)
Check if a class is a TestCase.

load_case(*testcase*)
Load a TestSuite containing all TestCase instances for all tests in a TestCase subclass.

Parameters **testcase** (*type*) – A subclass of unittest.TestCase

load_module(*module*)
Create and return a test suite containing all cases loaded from the provided module.

Parameters **module** (*module*) – A module object containing TestCases

load_test(*testcase*, *method_name*)
Create and return an instance of `unittest.TestCase` for the specified unbound test method.

unbound_test [unbound method] An unbound method of a `unittest.TestCase`

haas.main module

`haas.main.`**main**()
Execute haas.

Parameters **argv** (*list*) – The script's full argument list including the script itself.

haas.module_import_error module

class `haas.module_import_error.`**ModuleImportError**
Bases: `object`

A base class for generated ModuleImportError placeholder test cases.

haas.plugin_context module

class `haas.plugin_context.`**PluginContext**(*hooks=None*, ***kwargs*)
Bases: `object`

Handles correct setup and teardown of multiple plugins.

setup()

teardown()

haas.plugin_manager module

class `haas.plugin_manager.`**PluginManager**
Bases: `object`

ENVIRONMENT_HOOK = u'haas.hooks.environment'

RESULT_HANDLERS = u'haas.result.handler'

TEST_DISCOVERY = u'haas.discovery'

TEST_RUNNER = u'haas.runner'

add_plugin_arguments(*parser*)
Add plugin arguments to argument parser.

Parameters **parser** (*argparse.ArgumentParser*) – The main haas ArgumentParser.

get_driver(*namespace*, *parsed_args*, ***kwargs*)
Get mutually-exlusive plugin for plugin namespace.

get_enabled_hook_plugins(*hook*, *args*, ***kwargs*)
Get enabled plugins for specified hook name.

classmethod testing_plugin_manager(*hook_managers*, *driver_managers*)
Create a fabricated plugin manager for testing.

haas.result module

class haas.result.**ResultCollecter**(**args*, ***kwargs*)
Bases: *haas.result.ResultCollector*

class haas.result.**ResultCollector**(*buffer=False*, *failfast=False*)
Bases: object

Collecter for test results. This handles creating *TestResult* instances and handing them off the registered result output handlers.

addError(**args*, ***kw*)
Register that a test ended in an error.

Parameters

- **test** (*unittest.TestCase*) – The test that has completed.
- **exception** (*tuple*) – exc_info tuple (type, value, traceback).

addExpectedFailure(*test*, *exception*)
Register that a test that failed and was expected to fail.

Parameters

- **test** (*unittest.TestCase*) – The test that has completed.
- **exception** (*tuple*) – exc_info tuple (type, value, traceback).

addFailure(**args*, ***kw*)
Register that a test ended with a failure.

Parameters

- **test** (*unittest.TestCase*) – The test that has completed.
- **exception** (*tuple*) – exc_info tuple (type, value, traceback).

addSkip(*test*, *reason*)
Register that a test that was skipped.

Parameters

- **test** (*unittest.TestCase*) – The test that has completed.
- **reason** (*str*) – The reason the test was skipped.

addSuccess(*test*)
Register that a test ended in success.

Parameters **test** (*unittest.TestCase*) – The test that has completed.

addUnexpectedSuccess(**args*, ***kw*)
Register a test that passed unexpectedly.

Parameters **test** (*unittest.TestCase*) – The test that has completed.

add_result(*result*)
Add an already-constructed *TestResult* to this *ResultCollector*.

This may be used when collecting results created by other ResultCollectors (e.g. in subprocesses).

add_result_handler(*handler*)
Register a new result handler.

printErrors()

separator2 = u'--'

startTest(*test*, *start_time=None*)
Indicate that an individual test is starting.

Parameters

- **test** (*unittest.TestCase*) – The test that is starting.
- **start_time** (*datetime*) – An internal parameter to allow the parallel test runner to set the actual start time of a test run in a subprocess.

startTestRun()
Indicate that the test run is starting.

stop()
Set the `shouldStop` flag, used by the test cases to determine if they should terminate early.

stopTest(*test*)
Indicate that an individual test has completed.

Parameters **test** (*unittest.TestCase*) – The test that has completed.

stopTestRun()
Indicate that the test run has completed.

wasSuccessful()
Return `True` if the run was successful.

class haas.result.**TestCompletionStatus**
Bases: `enum.Enum`

Enumeration to represent the status of a single test.

error = 3
The test encountered an unexpected error.

expected_failure = 5
A test failed as expected

failure = 2
The test failed, but did not encounter an unexpected error.

skipped = 6
A test was skipped

success = 1
The test completed successfully.

unexpected_success = 4
A test marked as expected to fail unexpected passed.

class haas.result.**TestDuration**(*start_time*, *stop_time=None*)
Bases: `object`

An orderable representation of the duration of an individual test.

as_integer_ratio()

duration

start_time

stop_time

total_seconds

class haas.result.**TestResult**(*test_class*, *test_method_name*, *status*, *duration*, *exception=None*, *message=None*)

Bases: object

Container object for all information related to the run of a single test. This contains the test itself, the actual status including the reason or error associated with status, along with timing information.

classmethod from_dict(*data*)

Create a TestResult from a dictionary created by *to_dict()*

classmethod from_test_case(*test_case*, *status*, *duration*, *exception=None*, *message=None*, *stdout=None*, *stderr=None*)

Construct a *TestResult* object from the test and a status.

Parameters

- **test_case** (*unittest.TestCase*) – The test that this result will represent.
- **status** (haas.result.TestCompletionStatus) – The status of the test.
- **exception** (*tuple*) – exc_info tuple (type, value, traceback).
- **message** (*str*) – Optional message associated with the result (e.g. skip reason).
- **stdout** (*str*) – The test stdout if stdout was buffered.
- **stderr** (*str*) – The test stderr if stderr was buffered.

test

The test case instance this result represents.

to_dict()

Serialize the TestResult to a dictionary.

haas.result.**failfast**(*method*)

haas.suite module

class haas.suite.**TestSuite**(*tests=()*)

Bases: object

A TestSuite is a container of test cases and allows executing many test cases while managing the state of the overall suite.

countTestCases()

Return the total number of tests contained in this suite.

run(*result*, *_state=None*)

Run all tests in the suite.

Parameters **result** (*unittest.result.TestResult*) –

haas.suite.**find_test_cases**(*suite*)

Generate a list of all test cases contained in a test suite.

Parameters **suite** (haas.suite.TestSuite) – The test suite from which to generate the test case list.

haas.testing module

haas.utils module

class haas.utils.**abstractclassmethod**(*callable*)
Bases: classmethod

A decorator indicating abstract classmethods.

Similar to abstractmethod.

Usage:

class C(metaclass=ABCMeta): @abstractclassmethod def my_abstract_classmethod(cls, ...):

...

class haas.utils.**cd**(*destdir*)
Bases: object

haas.utils.**configure_logging**(*level*)

haas.utils.**get_module_by_name**(*name*)
Import a module and return the imported module object.

haas.utils.**uncamelcase**(*string*, *sep=u'_'*)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index

E

F

G

H

I

T

U

V

W