Rōnin API Documentation

Release 1.1.2

Tal Liron

Jan 06, 2018

Contents

A straightforward but powerful build system based on Ninja and Python, suitable for projects both big and small.

This is the API documentation. The complete user manual is on the wiki.

Get source code and report issues on GitHub.

CHAPTER 1

Packages

1.1 ronin

1.1.1 ronin.cli

ronin.cli.**cli**(**projects*)

Delegates control to the Rōnin CLI on one or more projects.

Note that the process is expected to exit after running the CLI, so this should only normally be used as the last call of your build script.

Parameters **projects** ([*Project*]) – projects

1.1.2 ronin.contexts

class ronin.contexts.**Context**(*parent=None*, *immutable=False*)

Bases: object

Keeps track of environmental and user configuration properties per run.

Designed to be attached to a single thread. Supports nesting contexts within the thread: a child context will return its parent's properties if it does not define them itself.

If the context is immutable it will raise *ImmutableContextException* if you try to modify any of the properties.

Parameters

- **parent** (*Context*) – parent context or None
- **immutable** (*bool*) – set to True to make immutable

append_to_import_path(*name*, *default=None*)

Convenience method to append a property in the context, if it exists, to sys.path.

Parameters

- **name** (*str*) – name in the format "key.property"
- **default** – default value

fallback(*value*, *name*, *default=None*)
If the value is not None, returns it. Otherwise works identically to *get()*.

Parameters

- **value** – value
- **name** (*str*) – name in the format "key.property"
- **default** – default value

Returns value, default, or None

get(*name*, *default=None*)
Gets a value from the context or `default` if undefined.

Values of any type can be stored in the context.

Note that if the value is defined and is None, then None is returned and *not* `default`.

Parameters

- **name** (*str*) – name in the format "ns.k"
- **default** – default value

Returns value, default, or None

exception ronin.contexts.**ContextException**(*message=None*)
Bases: exceptions.Exception

Base class for context exceptions.

exception ronin.contexts.**ImmutableContextException**(*message=None*)
Bases: *ronin.contexts.ContextException*

Attempted to modify an immutable context.

exception ronin.contexts.**IncorrectUseOfContextException**(*message=None*)
Bases: *ronin.contexts.ContextException*

Attempted to access a namespace instead of a property.

exception ronin.contexts.**NoContextException**(*message=None*)
Bases: *ronin.contexts.ContextException*

Attempted to access the current context but there is none.

exception ronin.contexts.**NotInContextException**(*message=None*)
Bases: *ronin.contexts.ContextException*

Attempted to access a property that is not in the context.

ronin.contexts.**configure_context**(*root_path=None*, *input_path_relative=None*, *output_path_relative=None*, *binary_path_relative=None*, *object_path_relative=None*, *source_path_relative=None*, *name=None*, *frame=1*)
Configures the current context for builds.

Parameters

- **root_path** (*str or FunctionType*) – the root of the input/output directory structure; defaults to the directory in which the calling script resides

- **input_path_relative** (*str or FunctionType*) – the default input path relative to the root; defaults to the root itself
- **output_path_relative** (*str or FunctionType*) – the default base output path relative to the root; defaults to 'build'
- **binary_path_relative** (*str or FunctionType*) – the default binary output base path relative to the output path; defaults to 'bin'
- **object_path_relative** (*str or FunctionType*) – the default object output base path relative to the output path; defaults to 'obj'
- **source_path_relative** (*str or FunctionType*) – the default source output base path relative to the output path; defaults to 'src'
- **name** (*str or FunctionType*) – optional name to use for descriptions
- **frame** (*int*) – how many call frames to wind back to in order to find the calling script

ronin.contexts.**current_context**(*immutable=True*)

Returns the current context if there is one, otherwise raises *NoContextException*.

By default, the context will be treated as immutable.

Parameters **immutable** (*bool*) – set to False in order to allow changes to the context

Returns current context

Return type *Context*

ronin.contexts.**new_child_context**()

Creates a new context.

If there already is a context in this thread, our new context will be a child of that context.

Returns new child context

Return type *Context*

ronin.contexts.**new_context**(***kwargs*)

Creates a new context and calls *configure_context()* on it. If there already is a context in this thread, our new context will be a child of that context.

Parameters

- **root_path** (*str or FunctionType*) – the root of the input/output directory structure; defaults to the directory in which the calling script resides
- **input_path_relative** (*str or FunctionType*) – the default input path relative to the root; defaults to the root itself
- **output_path_relative** (*str or FunctionType*) – the default base output path relative to the root; defaults to 'build'
- **binary_path_relative** (*str or FunctionType*) – the default binary output base path relative to the output path; defaults to 'bin'
- **object_path_relative** (*str or FunctionType*) – the default object output base path relative to the output path; defaults to 'obj'
- **source_path_relative** (*str or FunctionType*) – the default source output base path relative to the output path; defaults to 'src'
- **name** (*str or FunctionType*) – optional name to use for descriptions
- **frame** (*int*) – how many call frames to wind back to in order to find the calling script

1.1.3 ronin.executors

class ronin.executors.**Executor**
Bases: object

Base class for executors.

Variables

- **command** (*str or FunctionType*) – command
- **command_types** ([str]) – command types supported (used by extensions)
- **output_extension** (*str or FunctionType*) – when calculating outputs, change extension to this
- **output_prefix** (*str or FunctionType*) – when calculating outputs, prefix this to filename
- **hooks** ([FunctionType]) – called when generating the Ninja file

add_input(*value*)

command_as_str(*argument_filter=None*)

write_command(*f*, *argument_filter=None*)

class ronin.executors.**ExecutorWithArguments**
Bases: *ronin.executors.Executor*

Base class for executors with arguments.

add_argument(**value*)

add_argument_unfiltered(**value*)

remove_argument(**value*)

remove_argument_unfiltered(**value*)

write_command(*f*, *argument_filter=None*)

1.1.4 ronin.extensions

class ronin.extensions.**ExplicitExtension**(*inputs=None*, *include_paths=None*, *defines=None*, *library_paths=None*, *libraries=None*)
Bases: *ronin.extensions.Extension*

An extension with explicitly stated data to support gcc-like executors.

Parameters

- **inputs** ([str or FunctionType]) – input paths; note that these should be *absolute* paths
- **include_paths** ([str or FunctionType]) – include paths; note that these should be *absolute* paths
- **defines** ([(str or FunctionType, str or FunctionType)]) – defines in a (name, value) tuple format; use None for value if the define does not have a value
- **library_paths** ([str or FunctionType]) – include paths; note that these should be *absolute* paths

- **libraries** ([str or FunctionType]) – library names

apply_to_executor_gcc_compile(*executor*)

apply_to_executor_gcc_link(*executor*)

apply_to_phase(*phase*)

class ronin.extensions.**Extension**

Bases: object

Base class for extensions.

Extensions can nest child extensions (and so can they).

Variables **extensions** – child extensions

apply_to_executor(*executor*)

apply_to_phase(*phase*)

class ronin.extensions.**OutputsExtension**(*project*, *phase_name*)

Bases: *ronin.extensions.Extension*

An extension that pulls in outputs from another build phase.

Parameters

- **project** (Project) – project
- **phase_name** (*str or FunctionType*) – phase name in project

apply_to_executor_gcc_link(*executor*)

1.1.5 ronin.ninja

class ronin.ninja.**NinjaFile**(*project*, *command=None*, *encoding=None*, *file_name=None*, *columns=None*, *strict=None*)

Bases: object

Manages a Ninja build system file.

Parameters

- **project** (Project) – project
- **command** (*str or FunctionType*) – Ninja command; defaults to the context's ninja.command
- **encoding** (*str or FunctionType*) – Ninja file encoding; defaults to the context's ninja.encoding
- **file_name** (*str or FunctionType*) – Ninja filename (without ".ninja" extension); defaults to the context's ninja.file_name
- **columns** (*int*) – number of columns in Ninja file; defaults to the context's ninja.columns
- **strict** (*bool*) – strict column mode; defaults to the context's ninja.strict

build()

Calls *generate()* and runs Ninja as a subprocess in build mode.

Returns subprocess exit code

Return type int

clean()
Runs Ninja as a subprocess in clean mode, and then deletes the Ninja file if successful. Also makes sure to clean any temporary state for the project in the context.

Returns subprocess exit code

Return type int

command

delegate()
Calls *build()* and then exits the process with the correct exit code.

encoding

file_name
The Ninja file name, not including the path. The `file_name` if set, or else the project's `file_name`, or else `ninja.file_name` in the context.

Type str

generate()
Writes the Ninja file to *path*, overwriting existing contents and making sure to make parent directories.

path
Full path to the Ninja file. A join of the project's `output_path` and *file_name*.

Type str

remove()
Deletes the Ninja file at *path* if it exists.

write(*f*)
Writes the Nina file content.

Parameters **f** (*file-like*) – where to write

ronin.ninja.**configure_ninja**(*ninja_command=None*, *encoding=None*, *file_name=None*, *columns=None*, *strict=None*)

Parameters

- **ninja_command** (*str or FunctionType*) – `ninja` command; defaults to "ninja"
- **encoding** (*str or FunctionType*) – Ninja file encoding; defaults to "utf-8"
- **file_name** (*str or FunctionType*) – Ninja filename (without ".ninja" extension); defaults to "build"
- **columns** (*int*) – number of columns in Ninja file; defaults to 100
- **strict** (*bool*) – strict column mode; defaults to False

ronin.ninja.**escape**(*value*)
Escapes special characters for literal inclusion in a Ninja file.

Parameters **value** (*str or FunctionType*) – literal value to escape

Returns escaped value

Return type str

ronin.ninja.**pathify**(*value*)
Escapes special characters for inclusion in a Ninja file where paths are expected.

Parameters **value** (*str or FunctionType*) – path value to escape

Returns escaped value

Return type str

1.1.6 ronin.phases

class ronin.phases.**Output**(*path*, *the_file*)

Bases: object

Phase output.

Parameters

- **path** (*str*) – absolute path
- **the_file** (*str*) – file name

class ronin.phases.**Phase**(*project=None*, *name=None*, *executor=None*, *description=None*, *inputs=None*, *inputs_from=None*, *input_path=None*, *input_path_relative=None*, *extensions=None*, *output=None*, *output_path=None*, *output_path_relative=None*, *output_strip_prefix=None*, *output_strip_prefix_from=None*, *output_transform=None*, *run_output=False*, *run_command=None*, *rebuild_on=None*, *rebuild_on_from=None*, *build_if=None*, *build_if_from=None*)

Bases: object

A build phase within a project (see *Project*).

Each phase is equivalent to a single `rule` statement within a Ninja file together with the `build` statements that make use of it. Phases can be interrelated in complex ways: indeed, this feature is exactly what makes Rōnin useful (and writing Ninja files by hand difficult).

Phases can work in either "multi-output" or "single-output" mode, the latter triggered by setting the `output` parameter. The former is often used for incremental compilation, the latter often used for linking various outputs to a single binary.

A phase must be set with a *ronin.executors.Executor* to be useful. It was an architectural to separate the two classes in order to make it easier to extend code in each direction, however in the data model they are always joined.

Another important part of the architecture is *ronin.extensions.Extension*. This allows a kind of "live mix-in" for both phases and executors without having to extend those classes, for example to inject `inputs` and/or executor arguments.

As a convenience, if you set the `project` and `name` init arguments, then the phase will automatically be added to that project. You can do this manually instead.

Variables

- **vars** ({`str`: `FunctionType` or `str`}) – custom Ninja variables
- **hooks** ([`FunctionType`]) – called when generating the Ninja file

Parameters

- **project** (*Project*) – project to which this phase will be added (if set must also set `name`)
- **name** (*str or FunctionType*) – name in project to which this phase will be added (if set must also set `project`)
- **executor** (*Executor*) – executor
- **description** (*str or FunctionType*) – Ninja description; may include Ninja variables, such as `$out`; defaults to "[phase name] $out"

- **inputs** ([str or FunctionType]) – input paths; note that these should be *absolute* paths
- **inputs_from** ([str or FunctionType or *Phase*]) – names or instances of other phases in the project, the outputs of which we add to this phase's inputs
- **extensions** (Extension) – extensions
- **output** (*str or FunctionType*) – specifies that the phase has a *single* output; note that actual path of the output will be based on this parameter but not identical to it, for example "lib" might be added as a prefix, ".dll" as an extension, etc., according to the executor and/or project variant
- **output_path** (*str or FunctionType*) – override project's output_path; otherwise will be based on the executor's output_type
- **output_path_relative** (*str or FunctionType*) – joined to the context's paths.output
- **output_strip_prefix** (*str or FunctionType*) – stripped from outputs if they begin with this
- **output_strip_prefix_from** (*str or FunctionType or Phase*) – name or instance of other phase in project, from which the output path is used as this phase's output_strip_prefix
- **output_transform** (*FunctionType*) – called on all outputs
- **run_output** (*int*) – set to non-zero to run the output after a successful build in sequence
- **run_command** ([str or FunctionType]) – arguments for the run command; use "{output}" to insert output
- **rebuild_on** ([str or FunctionType]) – similar to inputs but used as "implicit dependencies" in Ninja (single pipe), meaning that the build will be re-triggered when these files change
- **rebuild_on_from** ([str or FunctionType or *Phase*]) – names or instances of other phases in the project, the outputs of which we add to this phase's rebuild_on
- **build_if** ([str or FunctionType]) – similar to inputs but used as "order dependencies" in Ninja (double pipe), meaning that the build will be triggered only after these files are built
- **build_if_from** ([str or FunctionType or *Phase*]) – names or instances of other phases in the project, the outputs of which we add to this phase's build_if

apply()
Applies all extensions and hooks to this phase.

command_as_str(*argument_filter=None*)
Applies all extensions to the executor and calls its command_as_str.

Returns command as string

Return type str

get_outputs(*inputs*)
Calculates the outputs for this phase depending on the inputs, applying output prefix and extension from the executor and finally the calling the output_transform function.

Parameters **inputs** ([str]) – inputs

Returns (True if "single-output", outputs); length of `outputs` will always be 1 in "single-output" mode, otherwise it will be the same length as `inputs`

Return type (`bool`, [*Output*])

input_path
The set `input_path`, or the context's `paths.input` joined to `input_path_relative`, or the project's `input_path`.

Type `str`

output_path
The set `output_path`, or the context's `paths.output` joined to `output_path_relative`, or the project's `output_path` for the executor's `output_type`.

Type `str`

1.1.7 ronin.projects

class ronin.projects.**Project**(*name*, *version=None*, *variant=None*, *input_path=None*, *input_path_relative=None*, *output_path=None*, *output_path_relative=None*, *file_name=None*, *phases=None*)

Bases: `object`

A container for an interrelated set of build phase (see *Phase*).

Every project is equivalent to a single Ninja file. Projects by default inherit properties from the current context, but can override any of them.

A Rōnin build script can in turn consist of any number of projects, though likely would require at least one to do something useful. Actually, due the dynamic nature of Rōnin build scripts, an entirely different number and nature of projects may be created by each run of a script.

After setting up projects, they are usually handed over to *cli()*. Though, you can also use the *NinjaFile* class directly instead.

Variables

- **phases** ({`str`: *Phase*}) – phases
- **hooks** ([`FunctionType`]) – called when generating the Ninja file
- **run** ({`int`: [`str` or `FunctionType`]}) – executed in order after a successful build

Parameters

- **name** (*str or FunctionType*) – project name
- **version** (*str or FunctionType*) – project version
- **variant** (*str or FunctionType*) – override project variant; defaults to the context's `projects.default_variant` or *host_platform()*
- **input_path** (*str or FunctionType*) – override input path; defaults to the context's `paths.input_path`
- **input_path_relative** (*str or FunctionType*) – override input path (relative)
- **output_path** (*str or FunctionType*) – override output path generation
- **output_path_relative** (*str or FunctionType*) – override output path generation (relative)

- **file_name** (*str or FunctionType*) – override Ninja file name; defaults to the context's `ninja.file_name`
- **phases** ({`str`: *Phase*}) – project phases

executable_extension
The executable extension for the *variant*.

See: *platform_executable_extension()*.

Type `str`

get_output_path(*output_type*)
The context's `paths.[output_type]` or project's *output_path* joined to the context's `paths.[output_type]_relative`.

Parameters **output_type** (*str or FunctionType*) – output type

Returns output path for output the type

Return type str

get_phase_for(*value*, *attr*)
Gets a phase and its name in the project. The argument is a phase name or a phase in the project.

Parameters

- **value** (*str or FunctionType or Phase*) – phase name or phase
- **attr** (*str*) – name of attribute from which the value was taken (for error messages)

Returns phase name, phase

Return type (`str`, `Phase`)

Raises **ValueError** – if phase not found in project

get_phase_name(*phase*)
The name of the phase if it's in the project.

Parameters **phase** (`Phase`) – phase

Returns phase name or `None`

Return type str

input_path
The set `input_path`, or the context's `paths.input`.

Type `str`

is_linux
True if *variant* is a Linux platform.

Type `bool`

is_windows
True if *variant* is a Windows platform.

Type `bool`

output_path
The set `output_path`, or the context's `paths.output` joined to the project's `output_path_relative` and *variant*.

Type `str`

shared_library_extension
The shared library extension for the *variant*.

See: *platform_shared_library_extension()*.

Type str

shared_library_prefix
The shared library prefix for the *variant*.

See: *platform_shared_library_prefix()*.

Type str

variant
Project variant.

Type str

1.2 ronin.binutils

class ronin.binutils.**WindRes**(*command=None*, *extension=None*, *platform=None*)
Bases: *ronin.executors.ExecutorWithArguments*

windres command from binutils.

Parameters

- **command** (*str or FunctionType*) – windres command; default's to context's binutils.windres_command
- **extension** (*str or FunctionType*) – output extensions; defaults to 'o'
- **platform** (*str or FunctionType or Project*) – target platform or project

output_coff()

output_format(*value*)

output_rc()

output_res()

ronin.binutils.**configure_binutils**(*windres_command=None*)
Configures the current context's binutils support.

Parameters **windres_command** (*str or FunctionType*) – windres command; defaults to "windres"

ronin.binutils.**which_windres**(*command*, *platform*, *exception=True*)
A specialized version of *ronin.utils.platform.which()* for windres.

Behind the scenes uses *windres_platform_command()*.

Parameters

- **command** (*str or FunctionType*) – windres command
- **platform** (*str or FunctionType or Project*) – target platform or project
- **exception** (*bool*) – set to False in order to return None upon failure, instead of raising an exception

Returns absolute path to command

Return type str

Raises ***WhichException*** – if `exception` is True and could not find command

`ronin.binutils.`**`windres_platform_command`**(*command*, *platform*)
Finds the `windres` command name for a specific target platform.

Behind the scenes uses *`platform_command()`*.

Parameters

- **`command`** (*`str or FunctionType`*) – `windres` command
- **`platform`** (*`str or FunctionType or Project`*) – target platform or project

Returns command

Return type str

1.3 `ronin.files`

`class` `ronin.files.`**`Copy`**(*command=None*)
Bases: *`ronin.executors.ExecutorWithArguments`*

File copy executor.

The phase inputs are copied to the phase outputs.

Use the phase's `output_strip_prefix` if you need to strip the input paths from the output paths.

Parameters **`command`** (*`str or FunctionType`*) – `cp` command; default's to context's `files.copy_command`

`ronin.files.`**`configure_files`**(*copy_command=None*)
Configures the current context's files support.

Parameters **`copy_command`** (*`str or FunctionType`*) – copy command; defaults to "cp"

1.4 `ronin.gcc`

`class` `ronin.gcc.`**`GccBuild`**(*command=None*, *ccache=True*, *platform=None*)
Bases: `ronin.gcc._GccWithMakefile`

gcc executor combining compilation and linking.

The phase inputs are ".c" source files. The phase output is an executable (the default), an ".so" or ".dll" shared library (call *`GccExecutor.create_shared_library()`*), or a static library (".a").

Parameters

- **`command`** (*`str or FunctionType`*) – `gcc` (or `g++`, etc.) command; defaults to the context's `gcc.gcc_command`
- **`ccache`** (*`bool`*) – whether to use ccache; defaults to True
- **`platform`** (*`str or FunctionType or Project`*) – target platform or project

`class` `ronin.gcc.`**`GccCompile`**(*command=None*, *ccache=True*, *platform=None*)
Bases: `ronin.gcc._GccWithMakefile`

gcc compile executor.

The phase inputs are ".c" source files. The phase outputs are ".o" object files.

Parameters

- **command** (*str or FunctionType*) – gcc (or g++, etc.) command; defaults to the context's gcc.gcc_command
- **ccache** (*bool*) – whether to use ccache; defaults to True
- **platform** (*str or FunctionType or Project*) – target platform or project

class ronin.gcc.**GccExecutor**(*command=None*, *ccache=True*, *platform=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Base class for gcc executors.

For a summary of all options accepted see the documentation.

Parameters

- **command** (*str or FunctionType*) – gcc (or g++, etc.) command; defaults to the context's gcc.gcc_command
- **ccache** (*bool*) – whether to use ccache; defaults to True
- **platform** (*str or FunctionType or Project*) – target platform or project

add_include_path(**value*)

add_input(*value*)

add_library(*value*)

add_library_path(**value*)

add_linker_argument(*name*, *value=None*, *xlinker=True*)
Add a command line argument to the linker.

For options accepted by ld see the documentation

compile_only()

create_makefile()

create_makefile_ignore_system()

create_makefile_only()

create_shared_library()

define(*name*, *value=None*)

disable_warning(*value*)

enable_debug()

enable_openmp()

enable_threads()

enable_warning(*value=u'all'*)

link_static_only()

linker_disable_new_dtags()

linker_exclude_symbols(**values*)

linker_export_all_symbols_dynamically()

linker_no_symbol_table()

linker_no_undefined_symbols()

linker_no_undefined_symbols_in_libraries()

linker_rpath(*value*)
Add a directory to the runtime library search path.

linker_rpath_origin()

linker_undefine_symbols(**values*)

optimize(*value*)

pic(*compact=False*)

set_machine(*value*)

set_machine_floating_point(*value*)

set_machine_tune(*value*)

set_makefile_path(*value*)

standard(*value*)

use_linker(*value*)

class ronin.gcc.**GccLink**(*command=None*, *ccache=True*, *platform=None*)
Bases: *ronin.gcc.GccExecutor*

gcc link executor.

The phase inputs are ".o" object files. The phase output is an executable (the default), an ".so" or ".dll" shared library (call `create_shared_library`), or a static library (".a").

Parameters

- **command** (*str or FunctionType*) – `gcc` (or `g++`, etc.) command; defaults to the context's `gcc.gcc_command`
- **ccache** (*bool*) – whether to use ccache; defaults to True
- **platform** (*str or FunctionType or Project*) – target platform or project

ronin.gcc.**configure_gcc**(*gcc_command=None*, *ccache=None*, *ccache_path=None*)
Configures the current context's gcc support.

Parameters

- **gcc_command** (*str or FunctionType*) – `gcc` (or `g++`, etc.) command; defaults to "gcc"
- **ccache** (*bool*) – whether to use ccache; defaults to True
- **ccache_path** (*str or FunctionType*) – ccache path; defaults to "/usr/lib/ccache"

ronin.gcc.**gcc_platform_command**(*command*, *platform*)
Finds the gcc command name for a specific target platform.

Behind the scenes uses *ronin.utils.platform.platform_command()*.

Parameters

- **command** (*str or FunctionType*) – `gcc` (or `g++`, etc.) command
- **platform** (*str or FunctionType or Project*) – target platform or project

Returns command

Return type str

ronin.gcc.**gcc_platform_machine_bits**(*platform*)
Bits for target platform.

Parameters **platform** (*str or FunctionType or Project*) – target platform or project

Returns '64' or '32'

Return type str

ronin.gcc.**which_gcc**(*command*, *ccache*, *platform*, *exception=True*)
A specialized version of *ronin.utils.platform.which()* for gcc that supports cross-compiling and ccache.

Behind the scenes uses *gcc_platform_command()*.

Parameters

- **command** (*str or FunctionType*) – gcc (or g++, etc.) command
- **ccache** (*bool*) – set to True to attempt to use ccache; if a ccache version is not found, will silently try to use the standard gcc command
- **platform** (*str or FunctionType or Project*) – target platform or project
- **exception** (*bool*) – set to False in order to return None upon failure, instead of raising an exception

Returns absolute path to command

Return type str

Raises ***WhichException*** – if exception is True and could not find command

1.5 ronin.go

class ronin.go.**GoCompile**(*command=None*)
Bases: *ronin.go.GoExecutor*

Go compile executor.

The phase inputs are ".go" source files. The phase outputs are ".o" object files.

Parameters **command** (*str or FunctionType*) – go command; defaults to the context's go.go_command

add_import_path(**values*)

assume_complete()

create_packages()

disable_errors_limit()

disable_inlining()

disable_local_imports()

disable_optimizations()

disable_unsafe_imports()

enable_large_model()

enable_memory_sanitizier()

enable_race_detector()

expected_import_path(**values*)

local_import_path(**values*)

class ronin.go.**GoExecutor**(*command=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Base class for Go executors.

Parameters **command** (*str or FunctionType*) – go command; defaults to the context's go.go_command

class ronin.go.**GoLink**(*command=None*, *platform=None*)
Bases: *ronin.go.GoExecutor*

Go link executor.

The phase inputs are ".o" object files. The phase output is an executable (the default), an ".so" or ".dll" shared library, or a static library (".a").

Parameters

- **command** (*str or FunctionType*) – go command; defaults to the context's go.go_command
- **platform** (*str or FunctionType or Project*) – target platform or project

add_import_path(**values*)

ar(*value*)

build_mode(*value*)

disable_data_checks()

disable_debug()

disable_dynamic_header()

disable_version_checks()

enable_memory_sanitizier()

enable_race_detector()

executable_format(*value*)

link_mode(*value*)

linker(*value*)

class ronin.go.**GoPackage**(*project*, *phase_name*)
Bases: *ronin.extensions.Extension*

A Go package generated by another phase.

Parameters

- **project** (Project) – project
- **phase_name** (*str or FunctionType*) – phase name in project

apply_to_executor_go(*executor*)

apply_to_phase(*phase*)

ronin.go.**configure_go**(*go_command=None*)
Configures the current context's Go support.

Parameters **go_command** (*str or FunctionType*) – go command; defaults to "go"

1.6 ronin.java

class ronin.java.**Jar**(*command=None*, *manifest=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Java Jar creation executor.

The phase inputs are ".class" files. The phase output is a ".jar" file.

Parameters

- **command** (*str or FunctionType*) – jar command; defaults to the context's java.jar_command
- **manifest** (*str or FunctionType*) – absolute path to manifest file

disable_manifest()

preserve_paths()

store_only()

class ronin.java.**JavaClasses**(*project*, *phase_name*)
Bases: *ronin.extensions.Extension*

Java classes generated by another phase.

Usable only with a *Jar* executor.

Parameters

- **project** (Project) – project
- **phase_name** (*str or FunctionType*) – phase name in project

apply_to_executor_java_jar(*executor*)

apply_to_phase(*phase*)

class ronin.java.**JavaCompile**(*command=None*, *classpath=[]*)
Bases: *ronin.executors.ExecutorWithArguments*

Java compile executor.

The phase inputs are ".java" source files. The phase outputs are ".class" files.

Parameters

- **command** (*str or FunctionType*) – javac command; defaults to the context's java.javac_command
- **classpath** ([str or FunctionType]) – initial classpath

add_classpath(*value*)

enable_debug()

ronin.java.**configure_java**(*javac_command=None*, *jar_command=None*)
Configures the current context's Java support.

Parameters

- **javac_command** (*str or FunctionType*) – `javac` command; defaults to "javac"
- **jar_command** (*str or FunctionType*) – `jar` command; defaults to "jar"

1.7 ronin.pkg_config

class ronin.pkg_config.**Package**(*name*, *command=None*, *path=None*, *static=False*)
Bases: *ronin.extensions.Extension*

A library that is configured by the external pkg-config tool.

Supports gcc-like executors.

Parameters

- **name** (*str or FunctionType*) – package name
- **command** (*str or FunctionType*) – `pkg-config` command; defaults to the context's `pkg_config.pkg_config_command`
- **path** (*str or FunctionType*) – `pkg-config` path; defaults to the context's `pkg_config.path`
- **static** (*bool*) – set to True to use static library linking

apply_to_executor_gcc_compile(*executor*)

apply_to_executor_gcc_link(*executor*)

ronin.pkg_config.**configure_pkg_config**(*pkg_config_command=None*, *pkg_config_path=None*)
Configures the current context's pkg-config support.

Parameters

- **pkg_config_command** (*str or FunctionType*) – `pkg-config` command; defaults to "pkg-config"
- **pkg_config_path** (*str or FunctionType*) – `pkg-config` path

1.8 ronin.qt

class ronin.qt.**QtMetaObjectCompile**(*command=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Qt meta-object compile (moc) executor.

The phase inputs are ".h" header files. The phase outputs are ".cpp" source files prefixed with "moc_".

Parameters **command** (*str or FunctionType*) – `moc` command; defaults to the context's `qt.moc_command`

add_framework_path(**value*)

add_include_path(**value*)

define(*name*, *value=None*)

ronin.qt.**configure_qt**(*moc_command=None*)
Configures the current context's Qt support.

Parameters **moc_command** (*str or FunctionType*) – `moc` command; defaults to "moc"

1.9 ronin.rust

class ronin.rust.**CargoBuild**(*command=None*, *jobs=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Cargo executor for Rust.

The phase input is a "Cargo.toml" file.

Cargo will be doing all the heavy lifting. You just want to make sure that Ninja knows when to rebuild, so set the phase's "output=" to equal your "[[bin]]" definition in Cargo.toml, and use "rebuild_on=" with the relevant source files.

Parameters

- **command** (*str or FunctionType*) – `cargo` command; defaults to the context's `rust.cargo_command`
- **jobs** (*int*) – number of jobs; defaults to CPU count + 1

enable_release()

jobs(*value*)

class ronin.rust.**RustBuild**(*command=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Rust build executor.

The phase inputs are ".rs" source files. The phase output is an executable (the default), an ".so" or ".dll" shared library, or a static library (".a").

Parameters **command** (*str or FunctionType*) – `rustc` command; defaults to the context's `rust.rustc_command`

enable_debug()

ronin.rust.**configure_rust**(*rustc_command=None*, *cargo_command=None*)
Configures the current context's Rust support.

Parameters

- **rustc_command** (*str or FunctionType*) – `rustc` command; defaults to "rustc"
- **cargo_command** (*str or FunctionType*) – `cargo` command; defaults to "cargo"

1.10 ronin.sdl

class ronin.sdl.**SDL**(*command=None*, *static=None*, *prefix=None*, *exec_prefix=None*)
Bases: *ronin.extensions.Extension*

The SDL library, configured using the sdl2-config tool that comes with SDL's development distribution. Supports gcc-like executors.

Note that you may also use *Package* to use SDL. However, this tool offers some special options you might need.

Parameters

- **command** (*str or FunctionType*) – `sdl-config` command; defaults to the context's `sdl.config_command`

- **static** (*bool*) – whether to link statically; defaults to the context's `sdl.config_static`
- **prefix** (*str or FunctionType*) – sdl-config prefix; defaults to the context's `sdl.prefix`
- **exec_prefix** (*str or FunctionType*) – sdl-config exec-prefix; defaults to the context's `sdl.exec_prefix`

apply_to_executor_gcc_compile(*executor*)

apply_to_executor_gcc_link(*executor*)

ronin.sdl.**configure_sdl**(*config_command=None*, *static=None*, *prefix=None*, *exec_prefix=None*)

Configures the current context's SDL support.

Parameters

- **config_command** (*str or FunctionType*) – config command; defaults to "sdl2-config"
- **static** (*bool*) – whether to link statically; defaults to False
- **prefix** (*str or FunctionType*) – sdl-config prefix
- **exec_prefix** (*str or FunctionType*) – sdl-config exec-prefix

1.11 ronin.utils

1.11.1 ronin.utils.argparse

class ronin.utils.argparse.**ArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *version=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*)

Bases: `argparse.ArgumentParser`

Enhanced argument parser.

- Support for flag arguments.
- Applied patch to fix this issue.

add_flag_argument(*name*, *help_true=None*, *help_false=None*, *default=False*)

Adds a flag argument as two arguments: `--my-flag` and `--no-my-flag`.

1.11.2 ronin.utils.collections

class ronin.utils.collections.**StrictDict**(*items=None*, *key_type=None*, *value_type=None*, *wrapper_function=None*, *unwrapper_function=None*)

Bases: `collections.OrderedDict`

An ordered dict that raises `TypeError` exceptions when keys or values of the wrong type are used.

Parameters

- **items** (*dict*) – initial dict
- **key_type** (type or str or (type or str)) – type(s) required for dict keys
- **value_type** (type or str or (type or str)) – type(s) required for dict values
- **wrapper_function** (*FunctionType*) – calls this optional function on all values before added to the list
- **unwrapper_function** (*FunctionType*) – calls this optional function on all values when retrieved from the list

class ronin.utils.collections.**StrictList**(*items=None*, *value_type=None*, *wrapper_function=None*, *unwrapper_function=None*)

Bases: list

A list that raises TypeError exceptions when objects of the wrong type are inserted.

Parameters

- **items** (*list*) – initial list
- **value_type** (type or str or (type or str)) – type(s) required for list values
- **wrapper_function** (*FunctionType*) – calls this optional function on all values before added to the list
- **unwrapper_function** (*FunctionType*) – calls this optional function on all values when retrieved from the list

append(*value*)

extend(*values*)

insert(*index*, *value*)

ronin.utils.collections.**dedup**(*values*)

Removes duplicate items from a list. Note that it does not change the original list.

Parameters **values** (*list*) – list

Returns de-duped list

Return type list

1.11.3 ronin.utils.messages

ronin.utils.messages.**announce**(*message*, *prefix=u'r\u014dnin'*, *color=u'green'*)

Writes a message to the terminal with a colored prefix.

Parameters

- **message** (*str*) – message
- **color** (*str*) – color name

ronin.utils.messages.**error**(*message*)

Writes an error message to the terminal with a red prefix.

Parameters **message** (*str or BaseException subclass instance*) – message or exception

ronin.utils.messages.**warning**(*message*)

Writes a warning message to the terminal with a yellow prefix.

Parameters **message** (*str*) – message

1.11.4 ronin.utils.paths

ronin.utils.paths.**base_path**(*path*)
Returns the real base path string of a file.

Parameters **path** (*str|FunctionType*) – path; calls *ronin.utils.strings.stringify()* on it

Returns base path of `path`

Return type str

ronin.utils.paths.**change_extension**(*path*, *new_extension*)
Changes the file extension to a new one.

The extension is defined as the segment following the last "." in the path.

Parameters

- **path** (*str|FunctionType*) – path; calls *ronin.utils.strings.stringify()* on it
- **new_extension** (*str|FunctionType*) – the new extension (if None, will return the path unchanged); calls *ronin.utils.strings.stringify()* on it

Returns path with new extension

Return type str

ronin.utils.paths.**glob**(*pattern*, *path=None*, *hidden=False*, *dirs=False*)
Returns a list of real path strings matching the pattern. If `path` is not specified, the pattern is implicitly joined to the context's `paths.input`.

Use "?" to match a single character, "*" to match zero or more characters, and "**" to match zero or more path segments.

Note that this implementation improves on Python's standard `glob.glob()` by supporting "**" correctly.

Parameters

- **pattern** (*str|FunctionType*) – pattern; calls *ronin.utils.strings.stringify()* on it
- **path** (*str|FunctionType*) – join the pattern to this path (when None, defaults to the context's `paths.input`); calls *ronin.utils.strings.stringify()* on it
- **hidden** (*bool*) – set to True to include hidden files
- **dirs** (*bool*) – set to True to include directories

Returns zero or more full paths to files (and optionally directories) matching the pattern

Return type [str]

ronin.utils.paths.**input_path**(**segments*)
Joins the path segments to the context's `paths.input`.

See *join_path()*.

Parameters **segments** (*[str|FunctionType|None]*) – path segments; calls *ronin.utils.strings.stringify()* on each

Returns path joined to `paths.input`

Return type str

ronin.utils.paths.**join_path**(**segments*)
Joins the path segments into a single path tring. No attempt is made to make it an absolute path, nor to check that it exists on the filesystem.

Null segments are skipped. Also note that unlike `os.path.join`, segments beginning with a path separator character will not cause the path to reset.

Parameters **segments** (*[str|FunctionType|None]*) – path segments; calls *ronin.utils.strings.stringify()* on each

Returns joined path

Return type str

ronin.utils.paths.**join_path_later**(**segments*)
Like *join_path()*, but deferred.

Parameters **segments** (*[str|FunctionType|None]*) – path segments; calls *ronin.utils.strings.stringify()* on each

Returns function that calls *join_path()*

Return type FunctionType

1.11.5 ronin.utils.platform

exception ronin.utils.platform.**WhichException**(*message=None*)
Bases: exceptions.Exception

which() could not find the command.

ronin.utils.platform.**configure_platform**(*prefixes=None*, *which_command=None*)
Configures the current context's platform support.

Parameters

- **prefixes** (*{str: str|FunctionType}*) – overrides for the default platform prefixes; unspecified keys will remain unchanged from their defaults
- **which_command** (*str|FunctionType*) – absolute path to *which()* command; defaults to "/usr/bin/which"

ronin.utils.platform.**host_bits**()
The bits (64 or 32) for the host machine on which we are running.

Returns bits

Return type integer

ronin.utils.platform.**host_operating_system_prefix**()
The operating system prefix for the host machine on which we are running.

Returns operating system

Return type str

ronin.utils.platform.**host_platform**()
The platform for the host machine on which we are running. A combination of *host_operating_system_prefix()* and *host_bits()*.

Returns host platform

Return type str

ronin.utils.platform.**platform_command**(*command*, *platform*)
The command prefixed for the platform, from *platform_prefixes()*.

Parameters

- **command** (*str|FunctionType*) – command
- **platform** (*str|FunctionType*) – platform

Returns prefixed command

Return type str

ronin.utils.platform.**platform_executable_extension**(*platform*)
The executable extension for the platform, e.g. `exe` for windows and None for other platforms.

Parameters **platform** (*str|FunctionType*) – platform

Returns executable extension or None

Return type str

ronin.utils.platform.**platform_prefix**(*platform*)
The prefix for the platform, from *platform_prefixes()*.

Parameters **platform** (*str|FunctionType*) – platform

Returns platform prefixes or ''

Return type str

ronin.utils.platform.**platform_prefixes**()
The current context's `platform.prefixes` or the defaults. See also *configure_platform()*.

Returns platform prefixes

Return type {str: str|FunctionType}

ronin.utils.platform.**platform_shared_library_extension**(*platform*)
The shared library extension for the platform, e.g. `dll` for windows and `so` for other platforms.

Parameters **platform** (*str|FunctionType*) – platform

Returns shared library extension or None

Return type str

ronin.utils.platform.**platform_shared_library_prefix**(*platform*)
The shared library extension for the platform, e.g. `lib` for *nix and None for Windows.

Parameters **platform** (*str|FunctionType*) – platform

Returns shared library prefix or None

Return type str

ronin.utils.platform.**which**(*command*, *exception=True*)
Finds the absolute path to a command on this host machine.

Works by invoking the operating system's `which` command, which configured via the context's `platform.which_command`. See also `configure_which()`.

Parameters

- **command** (*str|FunctionType*) – command
- **exception** (*bool*) – set to False in order to return None upon failure, instead of raising an exception

Returns absolute path to command

Return type str

Raises ***WhichException*** – if `exception` is True and could not find command

1.11.6 ronin.utils.strings

ronin.utils.strings.**bool_stringify**(*value*)

Like *stringify()*, except checks if the return value equals, ignoring case, to `true`.

Parameters **value** (*str|FunctionType*) – value

Returns True if the stringified value is `true`

Return type bool

ronin.utils.strings.**format_later**(*the_format*, **args*, ***kwargs*)

Creates a lambda that calls *stringify_list()* and *stringify_dict()* on the arguments and then `.format` their results on `the_format`.

Parameters

- **the_format** (*str|FunctionType*) – format string
- **values** (*[]*) – values

Returns lambda returning the formatted string

Return type FunctionType

ronin.utils.strings.**join_later**(*values*, *separator=u' '*)

Creates a lambda that calls *stringify_list()* and joins the results on `separator`.

Parameters

- **values** (*[]*) – values
- **separator** (*str|FunctionType*) – separator

Returns lambda returning the joined string

Return type FunctionType

ronin.utils.strings.**stringify**(*value*)

Casts the value to a Unicode string. If the value is a function, calls it using *ronin.contexts.current_context()* as its only argument, and recurses until a non-FunctionType value is returned.

None values are preserved, whether None is directly sent to this function or is the return value of function argument.

This function is the heart of Rōnin's deferred value capability, as it allows lambdas to be passed around instead of strings.

Parameters **value** (*str|FunctionType*) – value or None

Returns stringified value or None

Return type str

ronin.utils.strings.**stringify_dict**(*values*)

Calls *stringify()* on all dict values. Return values of None are preserved.

Parameters **values** (*{}*) – values

Returns values

Return type {object: str}

ronin.utils.strings.**stringify_list**(*values*)
Calls *stringify()* on all elements. Return values of None are preserved.

Parameters **values** ([]) – values

Returns values

Return type [str]

1.11.7 ronin.utils.types

ronin.utils.types.**import_symbol**(*name*)
Imports a symbol based on its fully qualified name.

Parameters **name** (*str*) – symbol name

Returns symbol

Raises

- **ImportError** – if could not import the module
- **AttributeError** – if could not find the symbol in the module

ronin.utils.types.**type_name**(*the_type*)
Human-readable name of type(s). Built-in types will avoid the "__builtin__" prefix.

Tuples are always handled as a join of "|".

Parameters **the_type** (*type* | *(type)*) – type(s)

Returns name of type(s)

Return type str

ronin.utils.types.**verify_subclass**(*value*, *the_type*)
Raises `TypeError` if the value is not a subclass of the type.

Parameters

- **value** – value
- **the_type** (*type* | *str*) – type or type name

Raises

- **TypeError** – if `value` is not a subclass of `the_type`
- **ValueError** – if `the_type` is invalid
- **ImportError** – if could not import the module
- **AttributeError** – if could not find the symbol in the module

ronin.utils.types.**verify_type**(*value*, *the_type*)
Raises `TypeError` if the value is not an instance of the type.

Parameters

- **value** – value
- **the_type** (*type* | *str*) – type or type name

Raises

- **TypeError** – if `value` is not an instance of `the_type`

- **ValueError** – if `the_type` is invalid
- **ImportError** – if could not import the module
- **AttributeError** – if could not find the symbol in the module

`ronin.utils.types.`**verify_type_or_subclass**(*value*, *the_type*)
Raises `TypeError` if the value is not an instance or subclass of the type.

Parameters

- **value** – value
- **the_type** (*type* | *str*) – type or type name

Raises

- **TypeError** – if `value` is not an instance or subclass of `the_type`
- **ValueError** – if `the_type` is invalid
- **ImportError** – if could not import the module
- **AttributeError** – if could not find the symbol in the module

1.11.8 ronin.utils.unicode

1.12 ronin.vala

class `ronin.vala.`**ValaApi**(*command=None*)
Bases: *ronin.vala.ValaExecutor*

Vala executor that generates ".vapi" files. These files are useful for incremental compilation, because they are equivalent to the real source files for the purposes of imports, but are much faster for `valac` to process.

The phase inputs are ".vala" (or ".gs") source files. The phase outputs are ".vapi" files.

Parameters **command** (*str*|*FunctionType*) – `valac` command; defaults to the context's `vala.valac_command`

class `ronin.vala.`**ValaBuild**(*command=None*)
Bases: *ronin.vala.ValaExecutor*

Vala single-phase build executor. Behind the scenes the Vala source code is transpiled to C source code and fed into gcc.

The phase inputs are ".vala" (or ".gs") source files. The single phase output is a binary.

Parameters **command** (*str*|*FunctionType*) – `valac` command; defaults to the context's `vala.valac_command`

class `ronin.vala.`**ValaExecutor**(*command=None*)
Bases: *ronin.executors.ExecutorWithArguments*

Base class for Vala executors.

Parameters **command** (*str*|*FunctionType*) – `valac` command; defaults to the context's `vala.valac_command`

add_cc_argument(*value*)

add_gir_path(**value*)

add_package(*value*)

add_source_path(**value*)

add_vapi_path(**value*)

compile_only()

create_c_code()

create_c_header(**value*)

create_deps(**value*)

create_fast_vapi(**value*)

disable_cc_warnings()

enable_cc_warnings()

enable_debug()

enable_deprecated()

enable_experimental()

enable_threads()

remove_cc_argument(*value*)

set_output_directory(**value*)

target_glib(*value*)

class ronin.vala.**ValaGccCompile**(*command=None*, *ccache=True*, *platform=None*)
Bases: *ronin.gcc.GccCompile*

Identical to *ronin.gcc.GccCompile*, just with a default configuration most suitable for compiling C code generated by *ValaTranspile*.

Parameters

- **command** (*str|FunctionType*) – `gcc` (or `g++`, etc.) command; defaults to the context's `gcc.gcc_command`
- **ccache** (*bool*) – whether to use ccache; defaults to True
- **platform** (str|FunctionTypel:class:*ronin.projects.Project*) – target platform or project

class ronin.vala.**ValaPackage**(*name=None*, *vapi_paths=None*, *c=True*, *c_compile_arguments=None*, *c_link_arguments=None*)
Bases: *ronin.extensions.Extension*

A Vala package.

Internally may also have an extension usable by gcc executors, so it can be used with both Vala and gcc executors.

Parameters

- **name** (*str|FunctionType*) – package name
- **c** (booll:class:*ronin.extensions.Extension*) – set to True (default) to automatically include a *ronin.pkg_config.Package* of the same name (used by gcc-compatible phases), False to disable, or provide any arbitrary extension
- **c_compile_arguments** (*[str|FunctionType]*) – arguments to add to gcc-compatible compile executors
- **c_link_arguments** (*[str|FunctionType]*) – arguments to add to gcc-compatible link executors

apply_to_executor_gcc_compile(*executor*)

apply_to_executor_gcc_link(*executor*)

apply_to_executor_vala(*executor*)

apply_to_executor_vala_build(*executor*)

class ronin.vala.**ValaTranspile**(*command=None*, *apis=None*)
Bases: *ronin.vala.ValaExecutor*

Vala executor that transpiles Vala source code to C source code.

The phase inputs are ".vala" (or ".gs") source files. The phase outputs are ".c" source files.

Due to the nature of the language, if the Vala source code import from other files, then they *must* be transpiled together, *or* a simplified ".vapi" version of these files can be used instead. For this reason, it's useful to precede transpilation with *ValaApi* phases. Feed them into the `apis` arguments here.

Parameters

- **command** (*str|FunctionType*) – `valac` command; defaults to the context's `vala.valac_command`
- **apis** ([strl:class:*ronin.phases.Phase*]) – phases with *ValaApi* executors

ronin.vala.**configure_vala**(*valac_command=None*)
Configures the current context's Vala support.

Parameters **valac_command** (*str|FunctionType*) – valac command; defaults to "valac"

CHAPTER 2

Indices and Tables

- genindex
- modindex
- search

Python Module Index

r

Index

A

B

C

D

E

F

G

H

I

J

L

N

O

P

Q

R

S

T

U

V

W