kentauros Documentation

Release 1.0.8dev

Fabio Valentini

Sep 18, 2018

Contents

CHAPTER 1

kentauros package

1.1 Subpackages

1.1.1 kentauros.actions package

Submodules

kentauros.actions.abstract module

This submodule contains the quasi-abstract *Action* class, which is inherited by specific action implementations.

class kentauros.actions.abstract.**Action**(*pkg_name: str*)

Bases: object

This class is the base class for all defined actions. For every action that can be specified at ktr command line, there is an Action subclass.

Parameters **pkg_name** (*str*) – Package instance this action will be run on

Variables

- **name** (*str*) – stores the package configuration name given at initialisation
- **kpkg** (Package) – stores reference to the package object
- **atype** (ActionType) – stores type of action as enum

execute()

This method runs the action corresponding to the *Action* instance on the package specified at initialisation. It is overridden by subclasses to contain the real code for the action. Also, it is expected that actions update the package database after they have finished successfully.

import_status()

This method imports a package's and all its sub-module's status to the package database.

update_status()
This method writes a package's and all its sub-module's status to the package database. It is executed after actions.

kentauros.actions.build module

This submodule contains the *BuildAction* class.

class kentauros.actions.build.**BuildAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This Action subclass contains information for executing a local build of the package specified at initialisation.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (ActionType) – here: stores ActionType.BUILD

execute() → bool
This method runs the action corresponding to the Action instance on the package specified at initialisation. It is overridden by subclasses to contain the real code for the action. Also, it is expected that actions update the package database after they have finished successfully.

kentauros.actions.chain module

This submodule contains the *ChainAction* class.

class kentauros.actions.chain.**ChainAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This Action subclass contains information for executing a "chain reaction" on the package specified at initialisation, which means the following:

- get sources if they don't already exist (GetAction)
- update sources (UpdateAction)
- if sources already existed, no updates were available and --force was not specified, action execution will terminate at this point and return False
- otherwise, sources are exported (if tarball doesn't already exist) (ExportAction)
- construct source package (ConstructAction), terminate chain if not successful
- build source package locally (BuildAction), terminate chain if not successful
- upload source package to cloud build service (UploadAction)

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (ActionType) – here: stores ActionType.CHAIN

execute() → bool
This method runs the "chain reaction" corresponding to the package specified at initialisation, with the configuration from the package configuration file.

Returns *bool* – True if chain went all the way through, False if not

kentauros.actions.clean module

This submodule contains the *CleanAction* class.

class kentauros.actions.clean.**CleanAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This `Action` subclass contains information for cleaning up the sources of the package specified at initialisation.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (`ActionType`) – here: stores `ActionType.CLEAN`

execute() → bool
This method cleans up the sources of to the package specified at initialisation. It executes the `Source.clean()` method of the `Source` instance in the specified package.

Returns *bool* – always `True` at the moment

kentauros.actions.common module

This submodule contains code that is shared by submodules in this subpackage.

kentauros.actions.common.**LOGPREFIX = 'ktr/actions'**
This string specifies the prefix for log and error messages printed to `stdout` or `stderr` from inside this subpackage.

kentauros.actions.construct module

This submodule contains the *ConstructAction* class.

class kentauros.actions.construct.**ConstructAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This `Action` subclass contains information for constructing the source package from sources and package specification for the package specified at initialisation.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (`ActionType`) – here: stores `ActionType.CONSTRUCT`

execute() → bool
This method runs the action corresponding to the `Action` instance on the package specified at initialisation. It is overridden by subclasses to contain the real code for the action. Also, it is expected that actions update the package database after they have finished successfully.

kentauros.actions.importing module

This submodule contains the *ImportAction* class.

class kentauros.actions.importing.**ImportAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This *Action* subclass contains methods for importing packages which are not yet configured for use with kentauros.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (`ActionType`) – here: stores *ActionType.IMPORT*

execute() → bool
This method prints a pretty summary of a package's configuration values to the console. Currently, this does nothing whatsoever.

kentauros.actions.prepare module

This submodule contains the *PrepareAction* class.

class kentauros.actions.prepare.**PrepareAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This *Action* subclass contains information for preparing the package's sources. Sources will be downloaded or copied to destination, updated if already there, and exported to a tarball, if necessary.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (ActionType) – here: stores *ActionType.PREPARE*

execute() → bool
This method executes the Source.prepare() method to execute source preparation. This includes downloading or copying to destination, updating if necessary, and exporting to tarball if necessary.

Returns *bool* – *True* when successful, *False* if sub-action fails

kentauros.actions.status module

This submodule contains the *StatusAction* class.

class kentauros.actions.status.**StatusAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This *Action* subclass contains information for displaying packages which are configured for use with kentauros. At the moment, this only includes printing a list of packages, which is done by default when kentauros is run. More status messages are plannned for the future.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (ActionType) – here: stores *ActionType.STATUS*

execute() → bool
This method prints a pretty summary of a package's configuration values to the console. Currently, this does nothing whatsoever.

kentauros.actions.upload module

This submodule contains the *UploadAction* class.

class kentauros.actions.upload.**UploadAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This *Action* subclass contains information for uploading the buildable source package to a cloud service (or similar) as specified in the package configuration.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (ActionType) – here: stores *ActionType.UPLOAD*

execute() → bool
This method executes the Uploader.upload() method to execute the source upload, if possible - this only has effect for packages with a valid uploader specified in the package configuration file.

Returns *bool* – always *True*, future error checking still missing

kentauros.actions.verify module

This submodule contains the *VerifyAction* class.

class kentauros.actions.verify.**VerifyAction**(*pkg_name: str*)
Bases: *kentauros.actions.abstract.Action*

This *Action* subclass contains information for making sure the package's configuration file is valid and everything needed for actions is in place.

Parameters **pkg_name** (*str*) – Package name for which status will be printed

Variables **atype** (ActionType) – here: stores *ActionType.VERIFY*

execute() → bool
This method executes a verification of the package configuration and checks if every file necessary for actions is present (and valid). The real checks are done during package initialisation.

Module contents

This subpackage contains the quasi-abstract Action class and its subclasses, which are used to hold information about the action specified at command line and are used to execute their respective actions. Additionally, this file contains a dictionary which maps ActionType enums to their respective class constructors.

kentauros.actions.**ACTION_DICT = {<ActionType.CONSTRUCT: 21>: <class 'kentauros.actions.co**
This dictionary maps ActionType enum members to their respective Action subclass constructors.

1.1.2 kentauros.init package

Submodules

kentauros.init.cli module

This module contains functions for constructing an *argparse* command line argument parser, and parsing the determined command line arguments. It also contains classes which provide methods that return parsed CLI arguments, depending on where the *kentauros* module is used from.

class kentauros.init.cli.**CLIArgs**
Bases: object

This class represents the parsed command line arguments and switches used with a kentauros script. It stores the parsed CLI arguments between class instantiations (in a class variable), so the CLI will be parsed only once. It also provides simple method calls for getting settings from the parsed CLI.

args = None
This class variable contains the permanent (instance-independent) storage of parsed CLI arguments. It is initially set to *None* and generated at the time of the first class initialisation.

Type ArgumentParser

get_action() → kentauros.definitions.ActionType
This method returns which package action has been specified at the command line. If this is *None*, something went wrong and no action type will be returned.

Returns *ActionType* – type of action that will be executed

get_debug() → bool
This method simply returns whether the *–debug* or *-d* switch has been set at the command line.

Returns *bool* – debug *on* or *off*

get_force() → bool
This method simply returns whether the *–force* or *-f* switch has been set at the command line.

Returns *bool* – force package actions *on* or *off*

get_message() → str
This method returns the changelog message specified by command line argument.

Returns *str* – specified changelog message

get_packages() → list
This method returns all the package names which were collected from the CLI arguments.

Returns *list* – specified packages in a list

get_packages_all() → bool
This method simply returns whether the *–all* or *-a* switch has been set at the command line.

Returns *bool* – parse all packages *on* or *off*

get_remove() → bool
This method simply returns whether the *–remove* or *-r* switch has been set at the command line.

Returns *bool* – remove package from database as part of cleanup *on* or *off*

get_verby() → int
This method returns how often the *–verbose* or *-v* switch has been set at the command line, subtracted from 2. This results in a lowest verbosity level of *2* (comparable to *–quiet*), a medium verbosity level of *1* (normal verbosity) and a high verbosity level of *0* (very verbose).

Returns *int* – verbosity level (0 to 2)

parser = None

kentauros.init.cli.**get_cli_parser**(*cli_parser: argparse.ArgumentParser*) → argparse.ArgumentParser
This function constructs and returns a parser for command line arguments, which is used for "normal" execution (e.g. invoking the *ktr* script).

The arguments parsed by this parser include those from the basic parser, and, additionally:

- package name(s) to specify which packages to process explicitly
- *–all* (*-a*) switch to enable processing all packages in *CONFDIR*
- *–force* (*-f*) switch to force actions which would not be executed

Parameters **cli_parser** (*ArgumentParser*) – basic argument parser got from *get_cli_parser_base()*

Returns *ArgumentParser* – CLI argument parser for *ktr* script

kentauros.init.cli.**get_cli_parser_base**() → argparse.ArgumentParser
This function constructs and returns a basic parser for command line arguments, which will be further processed and added to by other functions.

The arguments parsed by this basic parser include:

- *–debug* (*-d*) switch to enable debug messages in kentauros

- *–verbose* (*-v*, *-vv*) switch to control how many informational messages will be printed (twice for extra verbosity)

Returns *ArgumentParser* – basic CLI argument parser

`kentauros.init.cli.`**`package_name_completer`**(*prefix*, ***kwargs*)
This function returns a list of package names (found in the package configs directory by glob) that start with the 'prefix' argument.

Parameters **`prefix`** (*str*) – string prefix

Returns *list* – list of matching package names

kentauros.init.env module

This module contains simple functions for parsing environment variables needed for basic functions and determining debug and verbosity level.

`kentauros.init.env.`**`get_env_debug`**() → bool
This function returns *True* if the "*KTR_DEBUG*" environment variable was set to anything parse-able to *True* by python. If not (or the variable has not been set), it returns *False*.

Returns *bool* – debug *on* or *off*

`kentauros.init.env.`**`get_env_home`**() → str
This function tries to get the home directory of the user running ktr. If the *HOME* variable is not set, the current directory is used.

Returns *str* – environment variable for *HOME* or the current directory

`kentauros.init.env.`**`get_env_verby`**() → int
This function returns the parsed value of the *KTR_VERBOSITY* environment variable (anything parse-able to an *int* by python. If it has not been set, it will return *2* (the lowest verbosity level).

Returns *int* – verbosity level

Module contents

This subpackage contains functions for parsing environment variables and the functions necessary for getting and parsing command line arguments, which are then held in the `CLIArgs` class.

1.1.3 kentauros.modules package

Subpackages

kentauros.modules.builder package

Submodules

kentauros.modules.builder.abstract module

This module contains the abstract *Builder* class, which is then inherited by actual builders.

class kentauros.modules.builder.abstract.**Builder**(*package*)
Bases: *kentauros.modules.module.PkgModule*

This class is the base class for all builders. It's only real function is to provide a unified API for builder classes and store the package to which the builder belongs.

Parameters **package** (Package) – package to which this builder belongs

Variables **bpkg** (Package) – stores the package argument given at initialisation

build()
This method executes the builder commands.

export()
This method exports the built packages (if any) to the directory specified for package exports.

status() → dict
This method is expected to return a dictionary of statistics about the respective builder.

kentauros.modules.builder.abstract.**LOG_PREFIX = 'ktr/builder'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

kentauros.modules.builder.mock module

This module contains the *MockBuilder* class, which can be used to build binary packages from src.rpm packages.

kentauros.modules.builder.mock.**LOG_PREFIX = 'ktr/builder/mock'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.builder.mock.**MockBuild**(*mock: str*, *path: str*, *dist: str = None*)
Bases: object

This helper class is used for the actual execution of mock.

Parameters

- **mock** (*str*) – path of the used mock binary
- **path** (*str*) – path of the SRPM package that will be built
- **dist** (*str*) – chroot that the package will be built in

Variables

- ***mock*** (*str*) – stores the path of the used mock binary
- **path** (*str*) – stores the path of the SRPM package that will be built
- **dist** (*str*) – stores the chroot that the package will be built in

build() → int
This method starts the mock build (and waits for already running builds with the same chroot to finish before that).

Returns *int* – return code of the subprocess call

get_command() → list
This method returns the argument list needed by the subprocess method call, assembled from dist and path.

Returns *list* – argument list for consumption by subprocess methods

class kentauros.modules.builder.mock.**MockBuilder**(*package*)
Bases: *kentauros.modules.builder.abstract.Builder*

This `Builder` subclass is used to hold information and methods for executing a local package build using `mock`. At class instantiation, it checks for existence of the `mock` binary. If it is not found in `$PATH`, this instance is set to inactive.

Parameters **package** (*Package*) – package for which this mock/srpm builder is for

Variables **active** (*bool*) – determines if this instance is active

build() → bool
This method constructs the *MockBuilder* instances, which contain the commands for executing the builds, and executes them in turn. It also checks if the executing user is allowed to execute a mock build by checking if `$USER` is "root" or if the user is in the "mock" group.

If no source packages are found in the specified directory (`PACKDIR`), the build terminates without executing mock. If SRPM packages are found, only the most recent (biggest version number, determined just by sorting!) is built, for all specified chroots.

After the last mock invocation, a list of successful and unsuccessful builds is printed.

Returns *bool* – `True` if all builds succeeded, `False` if not

clean() → bool
This method is expected to clean up a sub-module's files and folders, if it creates any during its execution.

Returns *bool* – boolean indicating whether cleaning up was successful

execute() → bool
This method is expected to execute the package module and return a boolean, indicating whether the execution finished successfully or not.

Returns *bool* – boolean indicating whether the execution was successful

export() → bool
This method copies the build results (if any) from the mock result directory to the directory specified for binary package exports.

Returns *bool* – *True* if successful, *False* if not

get_active() → bool

Returns *bool* – boolean value indicating whether this builder should be active

get_dists() → list

Returns *list* – list of chroots that are going to be used for sequential builds

get_export() → bool

Returns *bool* – boolean value indicating whether this builder should export built packages

get_keep() → bool

Returns *bool* – boolean value indicating whether this builder should keep source packages

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method is expected to return a dictionary of statistics about the respective builder.

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

verify() → bool
This method runs several checks to ensure mock builds can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file
- checks if the *mock* binary is installed and can be found on the system
- checks if the current user is allowed to run builds with mock
- checks if the current user is root (building as root is strongly discouraged)

Returns *bool* – verification success

exception kentauros.modules.builder.mock.**MockError**(*value=''*)
Bases: Exception

This custom exception will be raised when errors occur during parsing of mock configuration files.

Parameters **value** (*str*) – informational string accompanying the exception

kentauros.modules.builder.mock.**get_default_mock_dist**() → str
This helper function tries to figure out which dist is the default one.

Returns *str* – dist string in default format

kentauros.modules.builder.mock.**get_dist_from_mock_config**(*dist: str*) → str
This helper function tries to read the "real" dist string from a custom dist config file.

Parameters **dist** (*str*) – name of the custom dist

Returns *str* – name of the underlying "standard" dist

kentauros.modules.builder.mock.**get_dist_result_path**(*dist: str*) → str
This helper function constructs the mock result path for a given dist string.

Parameters **dist** (*str*) – name of the standard dist

Returns *str* – path pointing to the mock result directory for this dist

kentauros.modules.builder.mock.**get_mock_cmd**() → str
This function tries to determine the correct mock binary path. If something is messing with the *$PATH* environment variable, it will try to account for that. If mock is not installed (or cannot be found within *$PATH*, this function will raise an Exception.

Raises subprocess.CalledProcessError

Returns *str* – path to the mock binary

Module contents

This subpackage contains the quasi-abstract Builder class and its MockBuilder subclass, which are used to hold information about the configured local builder for binary packages. This includes only MockBuilder right now, but should be extensible for other builders without need for architectural changes. Additionally, this file contains a dictionary which maps BuilderType enums to their respective class constructors.

kentauros.modules.builder.**BUILDER_TYPE_DICT = {<BuilderType.MOCK: 1>: <class 'kentauros.mo**
This dictionary maps *BuilderType* enum members to their respective *Builder* subclass constructors.

kentauros.modules.constructor package

Subpackages

kentauros.modules.constructor.rpm package

Submodules

kentauros.modules.constructor.rpm.spec_common module

This sub-module serves provides shared code for this subpackage. This includes:

- *RPMSpecError*: custom exception that is raised when errors occur during the parsing of .spec files
- *format_tag_line()*: simple function generating prettified spec tag lines

exception kentauros.modules.constructor.rpm.spec_common.**RPMSpecError**(*value=''*)
Bases: Exception

This custom exception will be raised when errors occur during parsing of an RPM spec file.

Parameters **value** (*str*) – informational string accompanying the exception

kentauros.modules.constructor.rpm.spec_common.**format_tag_line**(*tag: str*, *value: str*) → str
This function takes a tag and value as arguments and returns a nicely formatted tag line, aligning values after column 16 (second / fourth tab).

Parameters

- **tag** (*str*) – tag of tag line
- **value** (*str*) – tag value

Returns *str* – pretty tag line

kentauros.modules.constructor.rpm.spec_preamble_out module

This sub-module contains the functions that generate the necessary version strings for the .spec file, which depend on the type of source that is used.

kentauros.modules.constructor.rpm.spec_preamble_out.**SPEC_PREAMBLE_DICT = {<SourceType.LOCA**
This dictionary maps *SourceType* enum members to their respective RPM spec preamble generator functions.

kentauros.modules.constructor.rpm.spec_preamble_out.**spec_preamble_bzr**(*source: kentauros.modules.sources.bzr.BzrSo*) → str
This function returns the "%global" necessary for packages built from *bzr* repositories. This includes a definition of "rev" just now.

Parameters **source** (BzrSource) – source repository the revision will be determined from

Returns *str* – string containing the *%global rev $REV* line

kentauros.modules.constructor.rpm.spec_preamble_out.**spec_preamble_git**(*source: kentauros.modules.sources.git.GitSou*) → str

This function returns the "%globals" necessary for packages built from *git* repositories. This includes a definition of "commit" and "date" just now. The value of "commit" here are the first 8 characters of the corresponding git commit hash.

Parameters **source** (GitSource) – source repository the commit hash and date will be determined from

Returns *str* – string with the *%global commit COMMIT* and *%global date $DATE* lines

kentauros.modules.constructor.rpm.spec_preamble_out.**spec_preamble_local**(*source: kentauros.modules.sources.local.L*) → str

This function returns the "%global" necessary for packages built from tarballs specified by a *local path*.

Parameters **source** (LocalSource) – source the *%global* will be determined from

Returns *str* – empty string

kentauros.modules.constructor.rpm.spec_preamble_out.**spec_preamble_nosource**(*source: kentauros.modules.sources.no*) → str

This function returns an empty string.

Parameters **source** (NoSource) – source the *%global* will be determined from

Returns *str* – empty string

kentauros.modules.constructor.rpm.spec_preamble_out.**spec_preamble_url**(*source: kentauros.modules.sources.url.UrlSo*) → str

This function returns the "%global" necessary for packages built from tarballs specified by *url*.

Parameters **source** (UrlSource) – source the *%global* will be determined from

Returns *str* – empty string

kentauros.modules.constructor.rpm.spec_source_out module

This sub-module contains the functions that generate the necessary "Source0:" tags for the .spec file.

kentauros.modules.constructor.rpm.spec_source_out.**SPEC_SOURCE_DICT = {<SourceType.LOCAL: 3(**

This dictionary maps *SourceType* enum members to their respective RPM spec Source tag string generator functions.

kentauros.modules.constructor.rpm.spec_source_out.**spec_source_bzr**(*source: kentauros.modules.sources.bzr.BzrSource*) → str

This function returns the Source tag for packages built from *bzr* repositories.

Parameters **source** (BzrSource) – source repository a Source tag will be generated for

Returns *str* – Source tag with comments

kentauros.modules.constructor.rpm.spec_source_out.**spec_source_git**(*source: kentauros.modules.sources.git.GitSource*) → str

This function returns the Source string for packages built from *git* repositories.

Parameters **source** (GitSource) – source repository a Source tag will be generated for

Returns *str* – Source tag with comments

kentauros.modules.constructor.rpm.spec_source_out.**spec_source_local**(*source: kentauros.modules.sources.local.LocalS* → str

This function returns the Source string for packages built from tarballs specified by a *local path*.

Parameters **source** (LocalSource) – source a Source tag will be generated for

Returns *str* – Source tag in the format *Source0: $VERSION*

kentauros.modules.constructor.rpm.spec_source_out.**spec_source_nosource**(*source: kentauros.modules.sources.no_sour* → str

This function returns an empty string, as it should never be called.

Parameters **source** (NoSource) – source a Source tag will be generated for

Returns *str* – empty string

kentauros.modules.constructor.rpm.spec_source_out.**spec_source_url**(*source: kentauros.modules.sources.url.UrlSource*) → str

This function returns the Source string for packages built from tarballs specified by *url*.

Parameters **source** (UrlSource) – source a Source tag will be generated for

Returns *str* – Source tag in the format *Source0: $URL*

kentauros.modules.constructor.rpm.spec_version_out module

This sub-module contains the functions that generate the necessary "Version:" tags for the .spec file.

kentauros.modules.constructor.rpm.spec_version_out.**SPEC_VERSION_DICT = {<SourceType.LOCAL:**
This dictionary maps *SourceType* enum members to their respective RPM spec version string generator functions.

kentauros.modules.constructor.rpm.spec_version_out.**spec_version_bzr**(*source: kentauros.modules.sources.bzr.BzrSourc* → str

This function returns the version string for packages built from *bzr* repositories.

Parameters **source** (BzrSource) – source repository a version string will be generated for

Returns *str* – version string in the format *$VERSION+rev%{rev}*

kentauros.modules.constructor.rpm.spec_version_out.**spec_version_git**(*source: kentauros.modules.sources.git.GitSource*) → str

This function returns the version string for packages built from *git* repositories.

Parameters **source** (GitSource) – source repository a version string will be generated for

Returns *str* – version string in the format *$VERSION+git%{date}.%{commit}*

kentauros.modules.constructor.rpm.spec_version_out.**spec_version_local**(*source: kentauros.modules.sources.local.Loc*) → str

This function returns the version string for packages built from tarballs specified by a *local path*.

Parameters **source** (LocalSource) – source a version string will be generated for

Returns *str* – version string in the format *$VERSION*

kentauros.modules.constructor.rpm.spec_version_out.**spec_version_nosource**(*source: kentauros.modules.sources.no_s*) → str

This function returns the version string for packages built without sources.

Parameters **source** (NoSource) – source a version string will be generated for

Returns *str* – version string in the format *$VERSION*

kentauros.modules.constructor.rpm.spec_version_out.**spec_version_url**(*source: kentauros.modules.sources.url.UrlSourc*) → str

This function returns the version string for packages built from tarballs specified by *url*.

Parameters **source** (UrlSource) – source a version string will be generated for

Returns *str* – version string in the format *$VERSION*

Module contents

This subpackage serves the purpose of handling RPM spec files.

kentauros.modules.constructor.rpm.**LOG_PREFIX = 'ktr/constructor/rpm'**

This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.constructor.rpm.**RPMSpec**(*path: str, source: kentauros.modules.sources.abstract.Source*)

Bases: object

This class serves as the go-to swiss army knife for handling everything concerning RPM spec files from within kentauros.

The typical usage has 3 stages:

- reading from file (path that is passed at initialisation)
- manipulating .spec file contents

- writing to different file path

Variables

- **path** (*str*) – path pointing to the associated RPM spec file
- **source** (Source) – package sources that are needed to generate some information

build_preamble_string() → str
This method returns the appropriate spec preamble string, depending on the type of Source that has been set.

Returns *str* – preamble string containing necessary definitions

build_version_string() → str
This method returns the appropriate spec line containing the "Version" tag, depending on the type of Source that has been set.

Returns *str* – preamble string containing necessary definitions

do_release_reset()
This method resets the release number to 0suffix.

export_to_file(*path: str*)
This method exports the .spec file's (modified in memory) contents to another file, specified by the *path* argument.

Parameters **path** (*str*) – path to write the modified .spec contents to

get_lines() → list
This method splits the contents of the .spec file into lines and returns a list of them. It also removes the trailing newline at the end of files so they don't multiply like rabbits.

Returns *list* – list of lines

get_release() → str
This method reads and parses the RPM spec file's contents for its "Release" tag.

Returns *str* – release string found on the line containing the "Version:" tag

get_version() → str
This method reads and parses the RPM spec file's contents for its "Version" tag.

Returns *str* – version string found on the line containing the "Version:" tag

set_source()
This method writes the updated source tag to the rpm spec file.

set_version()
This method writes the updated version to the rpm spec file.

write_contents_to_file(*path: str*)
This method writes the .spec file's (modified in memory) contents to another file, specified by the *path* argument (BUT without prepending the preamble).

Parameters **path** (*str*) – path to write the modified .spec contents to

kentauros.modules.constructor.rpm.**do_release_bump**(*path: str*, *comment: str = None*) → bool
This function calls *rpmdev-bumpspec* with the specified arguments to bump the release number and create a changelog entry with a given comment.

Parameters **comment** (*str*) – comment to be added to the changelog entry

kentauros.modules.constructor.rpm.**parse_release**(*release: str) -> (<class 'str'>, <class 'str'>)*

This function splits the Release string into the leading numeric part and the trailing alphanumeric part.

Parameters **release** (*str*) – Release string

Returns *str, str* – numeric part, trailing part

Submodules

kentauros.modules.constructor.abstract module

This module contains the abstract *Constructor* class, which is then inherited by actual constructors.

class kentauros.modules.constructor.abstract.**Constructor**(*package*)

Bases: *kentauros.modules.module.PkgModule*

This class is the abstract base class for all constructors. It's only real function is to provide a unified API for builder classes and store the package to which the builder belongs.

Parameters **package** (Package) – package for which this constructor is for

Variables **cpkg** (Package) – stores parent package instance reference

build()

This method assembles the source package from files prepared previously.

cleanup()

This method cleans up all temporary files and directories.

export()

This method exports the assembled source packages to the specified package directory.

init()

This method creates the directory structure needed by other methods of this class.

prepare() → bool

This method prepares all files necessary for the actual assembly of the source package.

status() → dict

This method is expected to return a dictionary of statistics about the respective constructor.

kentauros.modules.constructor.abstract.**LOG_PREFIX = 'ktr/constructor'**

This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

kentauros.modules.constructor.srpm module

This module contains the *SrpmConstructor* class, which can be used to construct .src.rpm packages.

kentauros.modules.constructor.srpm.**LOG_PREFIX = 'ktr/constructor/srpm'**

This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.constructor.srpm.**SrpmConstructor**(*package*)

Bases: *kentauros.modules.constructor.abstract.Constructor*

This Constructor subclass implements methods for all stages of building and exporting source packages. At class instantiation, it checks for existence of *rpmbuild* and *rpmdev-bumpspec* binaries. If they are not found in $PATH, this instance is rendered inactive.

Parameters **package** (Package) – package for which this src.rpm constructor is for

Variables

- **active** (*bool*) – determines if this instance is active
- **dirs** (*str*) – dictionary containing some directory paths

_check_source_presence() → bool
This method checks if the Source's output directory is present.

_cleanup_sources()
This method cleans up the Source's output directory according to settings.

_copy_configuration()
This method copies the package configuration file to the *rpmbuild/SOURCES* directory in case it is included in the package build.

_copy_sources()
This method copies all files (not directories) in the sources directory to the *rpmbuild/SOURCES* directory.

_copy_specs_around() → bool
This method handles the package's .spec file.

Variables containing the old and new Version and Release strings are calculated (from all available sources of information, including the old spec file and the local state database).

A preamble to the spec file is generated based on which macros are expected to be set for different types of sources and is prepended to the output .spec file.

Based on the information about old and new Version and Release strings, whether a VCS update triggered this build or the build was forced, the new Version string is generated.

- The Release string might not change in case there are no changes to the package and a simple construct action has been triggered. No changelog entry is added.
- If the Version string has changed between builds (or it is the first package build), the Release is set to 1 and a changelog entry is added to the .spec file.
- If a package rebuild is forced (by the *–force* CLI argument, the Release is incremented by 1 and a changelog message is added to the .spec file.
- If a build has been triggered because of an updated VCS snapshot, the Release is reset to 1 and a changelog entry is added. This only works if the state database has all necessary information.

_do_initial_build_prep(*old_release: str*, *new_spec_path: str*)
This method prepares the .spec file for an initial package build.

Parameters

- **old_release** (*str*) – old release string
- **new_spec_path** (*str*) – path of new .spec file

static _do_packaging_only_build_prep(*new_spec_path: str*)
This method prepares the .spec file for a packaging-only change.

Parameters **new_spec_path** (*str*) – path of new .spec file

_do_snapshot_update_build_prep(*new_spec_path: str*)
This method prepares the .spec file for a snapshot update.

Parameters **new_spec_path** (*str*) – path of new .spec file

_do_version_update_build_prep(*new_spec_path: str*)
This method prepares the .spec file for a version update.

Parameters **new_spec_path** (*str*) – path of new .spec file

_get_last_release(*spec: kentauros.modules.constructor.rpm.RPMSpec*)
This method tries to read the latest state from the state database - if that is not available, the .spec file is parsed as a fallback.

Parameters **spec** (RPMSpec) – rpm spec object

Returns *str* – last known release

_get_last_version(*spec: kentauros.modules.constructor.rpm.RPMSpec*)
This method tries to read the latest state from the state database - if that is not available, the .spec file is parsed as a fallback.

Parameters **spec** (RPMSpec) – rpm spec object

Returns *str* – last known version

_get_old_status(*) -> (<class 'str'>, <class 'str'>)
This method tries to determine the old Version and Release strings from the best available source. If the local state database has not yet been updated with those values, the .spec file is parsed as a fallback.

Returns *tuple* – (version, release)

_get_spec_destination() → str
This method calculates the destination of the .spec file in the *rpmbuild/SPECS* dir.

_prepare_spec() → str
This method sets the *Version* and *Source0* tags in the template spec file and resets the *Release* tag to *0%{dist}* if the version has changed (to the best of the program's knowledge). A preamble to the .spec file containing all necessary macros / definitions is generated (and returned). Lastly, the spec is exported to the destination file.

Returns *str* – the contents of the .spec preamble

static _print_debug_info(*new_version*, *old_version*, *old_release*)
This method prints debug information about old and new Version and Release strings.

_restore_spec_template(*new_spec_path: str*, *preamble: str*)
This method restores the original .spec file template (without added preamble, but with possibly added changelog entries).

Parameters

- **new_spec_path** (*str*) – path of the new spec file
- **preamble** (*str*) – determined preamble string

build()
This method executes the actual SRPM package assembly. It sets *$HOME* to the created temporary directory and executes *rpmbuild -bs* with the copy of the package spec file in *rpmbuild/SPECS*. After that, *$HOME* is reset to the old value.

clean() → bool
This method is expected to clean up a sub-module's files and folders, if it creates any during its execution.

Returns *bool* – boolean indicating whether cleaning up was successful

cleanup()
This method cleans up all temporary files and directories.

execute() → bool
This method is expected to execute the package module and return a boolean, indicating whether the execution finished successfully or not.

Returns *bool* – boolean indicating whether the execution was successful

export()
This method copies the assembled source packages from *rpmbuild/SRPMS* to the directory for built packages as specified in the kentauros configuration. If multiple SRPM packages are found, they all are copied.

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

init()
This method creates a temporary directory (which is then set to *$HOME* in the *SrpmConstructor.build()* method) and other necessary sub-directories (here: *SOURCES*, *SRPMS*, *SPECS*).

prepare() → bool
This method prepares all files necessary for source package assembly.

Returns *bool* – returns *True* if the preparation was successful.

status() → dict
This method is expected to return a dictionary of statistics about the respective constructor.

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

verify() → bool
This method runs several checks to ensure srpm builds can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file
- checks if the *mock* binary is installed and can be found on the system
- checks if the current user is allowed to run builds with mock
- checks if the current user is root (building as root is strongly discouraged)
- checks if the .spec file is present at the expected location

Returns *bool* – verification success

Module contents

This subpackage contains the abstract `Constructor` base class and `DummyConstructor` and `SrpmConstructor` subclass definitions. They contain methods for building sources into buildable packages. Additionally, this file contains a dictionary which maps *kentauros.definitions.ConstructorType* enums to their respective class constructors.

kentauros.modules.constructor.**CONSTRUCTOR_TYPE_DICT = {<ConstructorType.SRPM: 1>: <class**
This dictionary maps *ConstructorType* enum members to their respective *Constructor* subclass constructors.

kentauros.modules.sources package

Submodules

kentauros.modules.sources.abstract module

This module contains the template / dummy *Source* class, which is then inherited by actual sources.

kentauros.modules.sources.abstract.**LOG_PREFIX = 'ktr/sources'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.sources.abstract.**Source**(*package*)
Bases: *kentauros.modules.module.PkgModule*

This class serves as an abstract base class for source handlers. They are expected to override this class's unimplemented methods. It also provides common infrastructure for all code sources in the form of generalised implementations of *get*, *refresh* and *formatver* methods.

Variables

- **updated** (*bool*) – indicates whether the source was updated since the state in the DB
- **sdir** (*str*) – source directory of the package this source belongs to
- **dest** (*str*) – destination path when downloading / copying sources
- **spkg** (Package) – stores the package argument given at initialisation
- **stype** (SourceType) – type of source

clean() → bool
This method cleans up all of a package's sources - excluding other files in the packages's source directory, which may include patches or other, additional files - they are preserved.

Returns *bool* – *True* if successful

execute() → bool
This method provides a generic way of preparing a package's sources. This will invoke the *Source.get()* method or the *Source.update()* method and the *Source.export()* method (as overridden by the subclass, respectively).

If sources can be downloaded / copied into place successfully, an update for them will not be attempted. Otherwise (sources are already present within the package directory), an update will be attempted before exporting.

Returns *bool* – success status of source getting or updating

export() → bool
It is expected that an appropriately named tarball is present within the package's source directory after this method has been executed.

formatver() → str
This method provides a generic way of getting a package's version as string. Subclasses are expected to override this method with their own version string generators, which then might include git commit hashes, git commit date and time, bzr revision, etc..

Returns *str* – formatted version string

get() → bool
It is expected that an appropriately named source file or directory is present within the package's source directory after this method has been executed.

get_keep() → bool
This method is expected to read and return the 'keep' value specified in the package configuration file in the source section.

get_orig() → str
This method is expected to read and return the 'orig' value specified in the package configuration file in the source section. It is also expected to replace variables with their corresponding values.

refresh() → bool
This method provides a generic way of refreshing a package's sources. This will invoke the generic *Source.clean()* method and the *Source.get()* method (as overridden by the subclass).

Returns *bool* – success status of source getting

status() → dict
This method is expected to return a dictionary of statistics about the respective source. This might include, for example, the current git commit hash, bzr revision number, etc.

update() → bool
It is expected that the source repository present within the package's source directory is up-to-date with upstream sources after this method has been executed, except when package configuration explicitely specifies something else.

kentauros.modules.sources.bzr module

This sub-module only contains the *BzrSource* class, which has methods for handling sources that have *source.type=bzr* specified and *source.orig* set to a bzr repository URL (or an *lp:* abbreviation) in the package's configuration file.

class kentauros.modules.sources.bzr.**BzrSource**(*package*)
Bases: *kentauros.modules.sources.abstract.Source*

This Source subclass holds information and methods for handling bzr sources.

- If the *bzr* command is not found on the system, *self.active* is automatically set to *False*
- For the purpose of checking connectivity to the remote server, the URL is stored in *self.remote*. If the specified repository is hosted on launchpad.net, *lp:* will be substituted with launchpad's URL automatically.

Parameters **package** (*Package*) – package instance this *Source* belongs to

export() → bool
This method executes the export from the package source repository to a tarball with pretty file name. It also respects the *bzr.keep=False* setting in the package configuration file - the bzr repository will be deleted from disk after the export if this flag is set.

Returns *bool* – *True* if successful, *False* if not or already exported

formatver() → str
This method returns a nicely formatted version string for bzr sources.

Returns *str* – nice version string (base version + "+bzr" + revision)

get() → bool
This method executes the bzr repository download to the package source directory. This respects the branch and revision set in the package configuration file.

Returns *bool* – *True* if successful, *False* if not or source already exists

get_branch() → str

Returns *str* – string containing the branch that is set in the package configuration

get_keep() → bool
This method is expected to read and return the 'keep' value specified in the package configuration file in the source section.

get_keep_repo() → bool

Returns *bool* – boolean value indicating whether the bzr repository should be kept

get_orig() → str

Returns *str* – string containing the upstream bzr repository URL (or *lp:* link)

get_revno() → str

Returns *str* – string containing the revision number that is set in the package configuration

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

rev() → str
This method determines which revision the bzr repository associated with this `BzrSource` currently is at and returns it as a string. Once run, it saves the last processed revision number in *self.saved_rev*, in case the revision needs to be determined when bzr repository might not be accessible anymore (e.g. if *bzr.keep=False* is set in configuration, so the repository is not kept after export to tarball).

Returns *str* – either revision string from repo, last stored rev string or *""* when unsuccessful

status() → dict
This method returns statistics describing this BzrSource object and its associated file(s). At the moment, this only includes the branch and revision specified in the configuration file.

Returns *dict* – key-value pairs (property: value)

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

update() → bool
This method executes a bzr repository update as specified in the package configuration file. If a specific revision has been set in the config file, this method will not attempt to execute an update.

Returns *bool* – *True* if update available and successful, *False* otherwise

verify() → bool
This method runs several checks to ensure bzr commands can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file
- checks if the *bzr* binary is installed and can be found on the system

Returns *bool* – verification success

kentauros.modules.sources.bzr.**LOG_PREFIX = 'ktr/sources/bzr'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

kentauros.modules.sources.git module

This sub-module contains only contains the *GitSource* class, which has methods for handling sources that have *source.type=git* specified and *source.orig* set to a git repository URL in the package's configuration file.

class kentauros.modules.sources.git.**GitSource**(*package*)
Bases: *kentauros.modules.sources.abstract.Source*

This Source subclass holds information and methods for handling git sources.

- If the *git* command is not found on the system, *self.active* is automatically set to *False*
- For the purpose of checking connectivity to the remote server, the URL is stored in *self.remote*.
- If neither *branch* nor *commit* hash has been set in the package configuration file, then the branch defaults to *master* (this is also written to the configuration file).
- If a specific commit hash has been specified in the package configuration file, *shallow* is automatically set to *False* (this is also written to the configuration file).

Parameters **package** (*Package*) – package instance this *GitSource* belongs to

commit() → str
This method provides an easy way of getting the commit hash of the requested commit. It also stores the latest commit hash between method invocations, if the source goes away and the hash is needed again.

Returns *str* – commit hash

date() → str
This method provides an easy way of getting the date and time of the requested commit in a standardised format (`YYMMDD.HHmmSS`). It also stores the latest parsed date between method invocations, if the source goes away and the commit datetime string is needed again.

The returned value represents the date and time of 'committing', not of 'authoring' the commit, and has been converted to UTC.

Returns *str* – commit date.time string (`YYMMDD.HHmmSS`)

export() → bool
This method executes the export from the package source repository to a tarball with pretty file name. It also respects the *git.keep=False* setting in the package configuration file - the git repository will be deleted from disk after the export if this flag is set.

Returns *bool* – *True* if successful or already done, *False* at failure

formatver() → str
This method assembles a standardised version string for git sources. This includes the package source base version, the git commit date and time and the first eight characters of the git commit hash, for example: `11.3.0+git160422.234950.39e9cf6c`

Returns *str* – nicely formatted version string

get() → bool
This method executes the git repository download to the package source directory. This respects the branch and commit set in the package configuration file.

Returns *bool* – *True* if successful, *False* if not or source pre-exists

get_branch() → str

Returns *str* – string containing the branch that is set in the package configuration

get_commit() → str

Returns *str* – string containing the commit hash that is set in the package configuration

get_keep() → bool
This method is expected to read and return the 'keep' value specified in the package configuration file in the source section.

get_keep_repo() → bool

Returns *bool* – boolean value indicating whether the git repository should be kept

get_orig() → str

Returns *str* – string containing the upstream git repository URL

get_shallow() → bool

Returns *bool* – boolean value indicating whether the git checkout depth should be 1 or not

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method returns statistics describing this BzrSource object and its associated file(s). At the moment, this only includes the branch and commit hash specified in the configuration file.

Returns *dict* – key-value pairs (property: value)

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

update() → bool
This method executes a git repository update as specified in the package configuration file. If a specific commit has been set in the config file, this method will not attempt to execute an update.

Returns *bool* – *True* if update available and successful, *False* if not

verify() → bool
This method runs several checks to ensure git commands can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file
- checks that the configuration file is consistent (i.e. shallow clone and commit checkout are not compatible)
- checks if the *git* binary is installed and can be found on the system

Returns *bool* – verification success

kentauros.modules.sources.git.**LOG_PREFIX = 'ktr/sources/git'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

kentauros.modules.sources.local module

This sub-module contains only contains the *LocalSource* class, which has methods for handling sources that have *source.type=local* specified and *source.orig* set to an absolute path of a local file in the package's configuration file.

kentauros.modules.sources.local.**LOG_PREFIX = 'ktr/sources/local'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.sources.local.**LocalSource**(*package*)
Bases: *kentauros.modules.sources.abstract.Source*

This Source subclass provides handling of local sources.

Parameters **package** (Package) – package instance this source belongs to

export() → bool
It is expected that an appropriately named tarball is present within the package's source directory after this method has been executed.

get() → bool
This method attempts to copy the specified source from the location specified in the package configuration file to the determined destination. If the destination file already exists, nothing will be done.

Returns *bool* – *True* if source was copied successfully, *False* if not

get_keep() → bool
This method is expected to read and return the 'keep' value specified in the package configuration file in the source section.

get_orig() → str

Returns *str* – string containing the source file path

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method is expected to return a dictionary of statistics about the respective source. This might include, for example, the current git commit hash, bzr revision number, etc.

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

update() → bool
It is expected that the source repository present within the package's source directory is up-to-date with upstream sources after this method has been executed, except when package configuration explicitely specifies something else.

verify() → bool
This method runs several checks to ensure local copying can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file

Returns *bool* – verification success

kentauros.modules.sources.no_source module

This sub-module only contains a dummy *NoSource* class which acts as a smart placeholder in case no source module is defined in the package configuration file.

kentauros.modules.sources.no_source.**LOG_PREFIX = 'ktr/sources/none'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.sources.no_source.**NoSource**(*package*)
Bases: *kentauros.modules.sources.abstract.Source*

This Source subclass does nothing except be a smart placeholder for packages that don't define a source module in their package configuration.

Parameters **package** (Package) – package instance this *NoSource* belongs to

export() → bool
It is expected that an appropriately named tarball is present within the package's source directory after this method has been executed.

formatver() → str
This method provides a generic way of getting a package's version as string. Subclasses are expected to override this method with their own version string generators, which then might include git commit hashes, git commit date and time, bzr revision, etc..

Returns *str* – formatted version string

get() → bool
It is expected that an appropriately named source file or directory is present within the package's source directory after this method has been executed.

get_keep() → bool
This method is expected to read and return the 'keep' value specified in the package configuration file in the source section.

get_orig() → str
This method is expected to read and return the 'orig' value specified in the package configuration file in the source section. It is also expected to replace variables with their corresponding values.

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method is expected to return a dictionary of statistics about the respective source. This might include, for example, the current git commit hash, bzr revision number, etc.

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

update() → bool
It is expected that the source repository present within the package's source directory is up-to-date with upstream sources after this method has been executed, except when package configuration explicitely specifies something else.

verify() → bool
This method checks if all configuration values needed for this module are present and valid.

Returns

bool –

boolean indicating whether the configuration / system verification completed
successfully

kentauros.modules.sources.source_error module

This sub-module only includes a custom Exception that is raised if unrecoverable errors occur during the execution of a Source module code.

exception kentauros.modules.sources.source_error.**SourceError**(*value=''*)
Bases: Exception

This custom exception will be raised when unrecoverable errors occur during execution of a Source module.

Parameters **value** (*str*) – informational string accompanying the exception

kentauros.modules.sources.url module

This sub-module contains only contains the *UrlSource* class, which has methods for handling sources that have *source.type=url* specified and *source.orig* set to a URL of a tarball in the package's configuration file.

kentauros.modules.sources.url.**LOG_PREFIX = 'ktr/sources/url'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class kentauros.modules.sources.url.**UrlSource**(*package*)
Bases: *kentauros.modules.sources.abstract.Source*

This Source subclass holds information and methods for handling URL sources.

- If the *wget* command is not found on the system, *self.active* is automatically set to *False*.
- For the purpose of checking connectivity to the remote server, the URL is stored in *self.remote*.

Parameters **package** (Package) – package instance this *UrlSource* belongs to

export() → bool
It is expected that an appropriately named tarball is present within the package's source directory after this method has been executed.

get() → bool
This method executes the download of the file specified by the URL to the package source directory.

Returns *bool* – *True* if successful, *False* if not or source already exists

get_keep() → bool
This method is expected to read and return the 'keep' value specified in the package configuration file in the source section.

get_keep_repo() → bool

Returns *bool* – boolean value indicating whether the downloaded file should be kept

get_orig() → str

Returns *str* – string containing the upstream file URL

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method returns a dictionary containing the package version of the tarball that was last downloaded (as the status is only updated after successful actions).

Returns *dict* – key-value pairs (property: value)

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

update() → bool
It is expected that the source repository present within the package's source directory is up-to-date with upstream sources after this method has been executed, except when package configuration explicitely specifies something else.

verify() → bool
This method runs several checks to ensure wget commands can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file
- checks if the *wget* binary is installed and can be found on the system

Returns *bool* – verification success

Module contents

This subpackage contains the `Source` base class an `BzrSource`, `GitSource`, `LocalSource` and `UrlSource` subclasses, which are used for holding information about a package's sources and methods for manipulating them. Additionally, this file contains a dictionary which maps `SourceType` enums to their respective class constructors.

kentauros.modules.sources.**SOURCE_TYPE_DICT = {<SourceType.LOCAL: 30>: <class 'kentauros.mo**
This dictionary maps `SourceType` enum members to their respective `Source` subclass constructors.

kentauros.modules.uploader package

Submodules

kentauros.modules.uploader.abstract module

This module contains the abstract *Uploader* class, which is then inherited by actual upload classes.

class kentauros.modules.uploader.abstract.**Uploader**(*package*)
Bases: *kentauros.modules.module.PkgModule*

This class serves as a quasi-abstract base class for source package uploader classes. They are expected to override this class's methods as necessary.

Parameters **package** (Package) – package for which this constructor is for

Variables **upkg** (Package) – stores parent package instance reference

status() → dict
This method is expected to return a dictionary of statistics about the respective uploader.

upload()
This method executes the source package upload with the settings specified in the package configuration.

kentauros.modules.uploader.copr module

This module contains the *CoprUploader* class, which can be used to upload .src.rpm packages to copr.

class kentauros.modules.uploader.copr.**CoprUploader**(*package*)
Bases: *kentauros.modules.uploader.abstract.Uploader*

This Uploader subclass implements methods for all stages of uploading source packages. At class instantiation, it checks for existence of the *copr-cli* binary. If it is not found in *$PATH*, this instance is set to inactive.

Parameters **package** (Package) – package for which this src.rpm uploader is for

Variables **active** (*bool*) – determines if this instance is active

clean() → bool
This method is expected to clean up a sub-module's files and folders, if it creates any during its execution.

Returns *bool* – boolean indicating whether cleaning up was successful

execute() → bool
This method is expected to execute the package module and return a boolean, indicating whether the execution finished successfully or not.

Returns *bool* – boolean indicating whether the execution was successful

get_active()

Returns *bool* – boolean value indicating whether this builder should be active

get_dists()

Returns *list* – list of chroots that are going to be used for sequential builds

get_keep()

Returns *bool* – boolean value indicating whether this builder should keep source packages

get_repo()

Returns *str* – name of the repository to upload to

get_wait()

Returns *bool* – boolean value indicating whether this builder should wait for remote builds

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method is expected to return a dictionary of statistics about the respective uploader.

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

upload() → bool
This method executes the upload of the newest SRPM package found in the package directory. The invocation of *copr-cli* also includes the chroot settings set in the package configuration file.

Returns *bool* – returns *False* if anything goes wrong, *True* otherwise

verify() → bool
This method runs several checks to ensure copr uploads can proceed. It is automatically executed at package initialisation. This includes:

- checks if all expected keys are present in the configuration file
- checks if the *copr-cli* binary is installed and can be found on the system

Returns *bool* – verification success

kentauros.modules.uploader.copr.**LOG_PREFIX = 'ktr/uploader/copr'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

Module contents

This subpackage contains the `Uploader` base class and `CoprUploader`, which is used for holding information about a package's upload location or method (if defined in package configuration). Additionally, this file contains a dictionary which maps `UploaderType` enums to their respective class constructors.

kentauros.modules.uploader.**UPLOADER_TYPE_DICT = {<UploaderType.COPR: 1>: <class 'kentauros**
This dictionary maps `UploaderType` enum members to their respective `Uploader` subclass constructors.

Submodules

kentauros.modules.module module

This module contains the abstract *PkgModule* class, defines the methods all package modules must provide.

class kentauros.modules.module.**PkgModule**
Bases: object

This abstract class defines the properties that all package modules must have.

__str__() → str
This method is expected to produce a nice string describing the sub-module.

Returns *str* – string containing a description of the sub-module

clean() → bool
This method is expected to clean up a sub-module's files and folders, if it creates any during its execution.

Returns *bool* – boolean indicating whether cleaning up was successful

execute() → bool
This method is expected to execute the package module and return a boolean, indicating whether the execution finished successfully or not.

Returns *bool* – boolean indicating whether the execution was successful

imports() → dict
This method is expected to return a dictionary of statistics about a module that has not yet been imported into the package database.

Returns *dict* – dictionary containing the sub-module's imported stats

status() → dict
This method is expected to return a dictionary of statistics about this module.

Returns *dict* – dictionary containing the sub-module's exported stats

status_string() → str
This method is expected to return a string describing the status of this module.

Returns *str* – string containing module statistics

verify() → bool
This method checks if all configuration values needed for this module are present and valid.

Returns

bool –

boolean indicating whether the configuration / system verification completed successfully

Module contents

This subpackage contains all plug-able kentauros modules.

1.2 Submodules

1.3 kentauros.bootstrap module

This module contains functions that cover the bare necessity of setting up the directories kentauros expects to exist. This happens after CLI arguments and environment variables have been parsed to determine which directories those should be.

kentauros.bootstrap.**LOG_PREFIX = 'ktr/bootstrap'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

kentauros.bootstrap.**ktr_bootstrap**() → bool
This function has to be called before any other actions are attempted on packages. It ensures that the required directory structure is present. If it fails, kentauros execution will be aborted.

Returns *bool* – success (or not)

kentauros.bootstrap.**ktr_mkdirp**(*path: str*) → bool
This function checks for directory existence and the ability to write to it. If the directory does not exist, it will be created.

Parameters **path** (*str*) – path of directory to check and create

Returns *bool* – success (or not)

1.4 kentauros.conntest module

This module contains a simple function which is used for basic connectivity checks before actions that require internet access / access to a specific URL.

kentauros.conntest.**is_connected**(*host_url: str*) → bool
This function tries to create a connection to the hostname specified by the URL argument. If any error occurs during connecting, *False* is returned.

Parameters **host_url** (*str*) – URL of the host connectivity will checked to

Returns *bool* – *True* if connection setup successful, *False* if not

kentauros.conntest.**trial**(*address: str*) → bool
This helper function attempts to connect to a remote host exactly once.

Parameters **address** (*str*) – URL string

Returns *bool* – *True* if successful, *False* if not

1.5 kentauros.definitions module

This module contains all definitions of types and global variables that are used by nearly every other module / subpackage.

It includes these globally used variables:

- KTR_SYSTEM_DATADIR: directory where kentauros data is installed to resides when installed
- KTR_VERSION: this version string is used when installing and packaging kentauros

The following Enums are defined here:

- KtrConfType: types of kentauros configuration source
- *ActionType*: types of actions that can be executed
- *BuilderType*: types of supported binary package builders
- *ConstructorType*: types of supported source package constructors
- *SourceType*: types of supported package sources
- *UploaderType*: types of supported package upload modules

class kentauros.definitions.**ActionType**
Bases: enum.Enum

This Enum defines the different types of actions that are supported by kentauros. This includes a default *NONE* type.

BUILD = 22

CHAIN = 40

CLEAN = 30

CONSTRUCT = 21

IMPORT = 10

NONE = 0

PREPARE = 20

STATUS = 11

UPLOAD = 23

VERIFY = 12

class kentauros.definitions.**BuilderType**
Bases: enum.Enum

This Enum defines the different types of binary package builders that are supported by kentauros. It also includes a default *NONE* type.

MOCK = 1

NONE = 0

class kentauros.definitions.**ConstructorType**
Bases: enum.Enum

This Enum defines the different types of source package builders that are supported by kentauros. It also includes a default *NONE* type.

NONE = 0

SRPM = 1

kentauros.definitions.**KTR_SYSTEM_DATADIR = '/usr/share/kentauros/'**
This string represents the absolute path which kentauros data files are copied to at installation and where they are expected to be when running ktr.

kentauros.definitions.**KTR_VERSION = '1.0.8'**
This string represents the version string used by `setup.py` at install time and `make-srpm.sh` when building an srpm package. The version in `kentauros.spec` has to be set manually yet.

class kentauros.definitions.**PkgModuleType**
Bases: enum.Enum

This Enum defines the types of package submodules there can be.

BUILDER = 3

CONSTRUCTOR = 2

SOURCE = 1

UPLOADER = 4

class kentauros.definitions.**SourceType**
Bases: enum.Enum

This Enum defines the different types of package source sources that are supported by kentauros. It also includes a default *NONE* type.

BZR = 21

GIT = 20

LOCAL = 30

NONE = 0

URL = 10

class kentauros.definitions.**UploaderType**
Bases: enum.Enum

This Enum defines the different types of (source) package uploader modules that are supported by kentauros. It also includes a default *NONE* type.

COPR = 1

NONE = 0

1.6 kentauros.instance module

This sub-module contains the *Kentauros* class, which holds configuration values parsed from CLI arguments, environment variables and configuration files. The implementation makes sure that command line arguments, environment

variables and configuration files are parsed only once per program run. Additionally, this subpackage holds logging and error printing functions.

class kentauros.instance.**Kentauros**
Bases: object

This class stores settings and variables that must be the same during the execution of code from the "kentauros" package. This is accomplished by storing the critical data in a class variable, which is initialised only once per execution.

add_package(*conf_name: str*, *package*)
This method adds a package to the list of packages known to the kentauros instance.

Parameters

- **conf_name** – package configuration name
- **package** – Package object

Returns *bool* – successful addition to package list

cli = None

conf = None

debug = None

get_basedir() → str
This method tries to parse the kentauros configuration file for a specified base directory. If it is not specified, it returns the current directory as a fallback value.

Returns *str* – kentauros base directory

get_confdir() → str
Returns the string "configs" appended to the base directory.

get_datadir() → str
Returns the string "sources" appended to the base directory.

get_expodir() → str
Returns the string "exports" appended to the base directory.

get_package(*conf_name: str*)
This method gets a package from the list of packages known to the kentauros instance.

Parameters **conf_name** – package configuration name

Returns *Package* – package object

get_package_names()
This method gets the list of known package configuration names.

Returns *list* – list of package configurations active in this instance

get_packdir() → str
Returns the string "packages" appended to the base directory.

get_specdir() → str
Returns the string "specs" appended to the base directory.

initialised = False

packages = None

state_delete(*conf_name: str*) → int
This method deletes the entries for the given package from the database and returns its ID.

Parameters **conf_name** (*str*) – package configuration name

Returns *int* – ID of the removed entry

state_read(*conf_name: str*) → dict
This method reads the entries for the given package from the database and returns them as an ordered dictionary.

Parameters **conf_name** (*str*) – package configuration name

Returns *dict* – result of the query

state_write(*conf_name: str*, *entries: dict*) → int
This method inserts or updates a package's entry in the state database with the dictionary entries given.

Parameters

- **package** (*str*) – package configuration name
- **entries** (*dict*) – dict containing the key-value pairs to insert or update in the db

Returns *int* – ID of the package in the database

verby = None

workdir = '/home/docs/checkouts/readthedocs.org/user_builds/kentauros/checkouts/stable

kentauros.instance.**LOG_PREFIX = 'ktr/instance'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

1.7 kentauros.logger module

This sub-module contains the KtrLogger class, which provides methods for logging messages to file or standard outputs.

class kentauros.logger.**KtrLogger**(*log_prefix: str = None*)
Bases: object

This class provides methods for printing messages to standard outputs.

Parameters **log_prefix** – specifies a custom prefix for log messages

Variables **log_prefix** – stores a custom prefix for log messages

dbg(*msg: str*, *prefix: str = None*)
This method prints messages with a "DEBUG: " prefix to stdout, but only if the *KTR_DEBUG* environment variable has been set or the *–debug* or *-d* flag has been supplied at the command line.

Parameters **msg** (*str*) – debug message to be printed

err(*msg: str*, *prefix: str = None*)
This method prints messages with an "ERROR: " prefix to standard error output, regardless of environment variables and CLI settings supplied.

Parameters **msg** (*str*) – error message to be printed

log(*msg: str*, *pri: int = 2*, *prefix: str = None*, *outfile=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *sep: str = ':'*)
This method prints messages to standard output, depending on the priority argument and the verbosity level determined from environment variables and CLI switches.

Invocation with

- *pri=2* (which is the default) will always print the attached message

- *pri=1* will print messages when verbosity is set to 1
- *pri=0* will print messages when verbosity is set to 0 or when debugging is enabled

Parameters

- **msg** (*str*) – message that will be printed
- **pri** (*int*) – message priority (0-2, where 0 is lowest and 2 is highest)

log_command(*cmd_list: list*, *pri: int = 2*, *prefix: str = None*)
This method prints commands that are then executed by use of the `subprocess.call()` or `subprocess.check_output()` functions. Its priority behaviour is the same as the `Kentauros.log()` function's.

Parameters

- **cmd_list** (*list*) – list of strings, as passed to `subprocess` functions
- **pri** (*int*) – message priority (0-2, where 0 is lowest and 2 is highest)
- **prefix** (*str*) – custom module-wide prefix string

log_list(*header: str*, *lst: list*, *pri: int = 2*, *prefix: str = None*)
This method prints lists, with one element on each line. Its priority behaviour is the same as the `Kentauros.log()` function's.

Parameters

- **header** (*str*) – header for the list
- **lst** (*list*) – list of objects that are parse-able to str()s by python
- **pri** (*int*) – message priority (0-2, where 0 is lowest and 2 is highest)
- **prefix** (*str*) – custom module-wide prefix string

`kentauros.logger.`**LOG_PREFIX = 'ktr/logger'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

`kentauros.logger.`**print_flush**(**args*, ***kwargs*)
This function serves as a wrapper around the built-in print function. Calling it instead of the standard *print* ensures that output buffers are flushed after every call.

1.8 kentauros.package module

This sub-package contains the *Package* class, which holds package configuration parsed from the corresponding *package.conf* file. After parsing the package configuration, package sub-modules are added according to configuration.

`kentauros.package.`**LOG_PREFIX = 'ktr/package'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

class `kentauros.package.`**Package**(*conf_name: str*)
Bases: `object`

This class envelops all things necessary to perform actions on a specific "package" of software.

Parameters **conf_name** (*str*) – name of the configuration file, without the ".conf" suffix

Variables

- **file** (*str*) – configuration file path

- **conf** (*ConfigParser*) – parser for package.conf file

Raises *PackageError* – error if package.conf file is invalid

get_conf_name() → str

Returns *str* – package configuration name

get_module(*module_type: str*)
This method gets a specific package module from the module dictionary.

Parameters **module_type** (*str*) – module type string (abstract class name, lower-case)

Returns *PkgModule* – corresponding package module, if it is found

get_modules()
This method gets all a package's modules.

Returns *list* – package module list

get_name() → str

Returns *str* – package name

get_version() → str

Returns *str* – package version string

replace_vars(*input_str: str*) → str
This method replaces variables in configuration file values with the appropriate values set elsewhere. For example, this can be used to specify the name and version inside a URL.

Parameters **input_str** (*str*) – string where variables should be replaced

Returns *str* – string where variables have been replaced

status() → dict
This method returns statistics describing this Package object and its associated source.

Returns *dict* – key-value pairs (property: value)

status_string() → str
This method returns a string containing statistics describing this Package object and its associated source.

Returns *str* – package information

verify() → bool
This method verifies that the absolute minimum for proceeding with package initialisation is set. This also ensures the validity of some entries.

Returns *bool* – *True* if configuration is minimally valid, *False* if entries are missing

exception kentauros.package.**PackageError**(*value: str = ''*)
Bases: Exception

This custom exception will be raised when errors occur during parsing of a package .conf file.

Parameters **value** (*str*) – informational string accompanying the exception

1.9 kentauros.run module

This module contains the *run()* function that is called as entry point from the *ktr* script.

kentauros.run.**LOG_PREFIX = 'ktr'**
This string specifies the prefix for log and error messages printed to stdout or stderr from inside this subpackage.

kentauros.run.**do_import_action**(*name: str*)
This function executes the Import action for the given package configuration.

Parameters **name** (*str*) – package configuration name

kentauros.run.**do_process_packages**() -> (<*class 'list'*>, <*class 'list'*>)
This function processes all packages and executes the specified action on them.

Returns *list, list* – list of successful package actions, list of unsuccessful package actions

kentauros.run.**do_verify_action**(*name: str*) → bool
This function executes the Verify action for the given package configuration.

Parameters **name** (*str*) – package configuration name

kentauros.run.**get_packages**() → list
This function returns the set of all packages that are specified as command line arguments or all packages that have configuration files present, depending on whether the *–all* CLI flag has been set.

Returns *list* – list of package configuration names

kentauros.run.**get_packages_all**() → list
This function parses the content of the package configuration files directory and returns the package configuration names.

Returns *list* – list of strings of package configuration names

kentauros.run.**get_packages_cli**() → list
This function parses the package configuration names supplied via CLI arguments, removes any that do not match with a present configuration file in the respective directory, and returns the checked list.

Returns *list* – list of strings of package configuration names

kentauros.run.**init_package_objects**(*packages: list*)
This function parses the list of package configuration names and initialises the *Package* objects. If no errors occur during initialisation, the *Package* instance is added to the kentauros instance's list of packages.

kentauros.run.**print_no_package_error**()
This function prints an error and helpful information if no action is specified.

kentauros.run.**print_package_header**(*name: str*)
This function prints a small banner to indicate which package actions are executed for next.

Parameters **name** (*str*) – package configuration name

kentauros.run.**print_parameters**()
This function prints the kentauros program parameters.

kentauros.run.**run**() → int
This function is corresponding to (one of) the "main" function of the *kentauros* package and is the entry point used by the *ktr.py* script from git and the script installed at installation.

1.10 Module contents

kentauros is an automated build system.

at the moment, the following actions are supported:

- getting source code from:
  - local tarball
  - tarball at URL

  - git repository
  - bzr repository
- building source packages:
  - RPM format (.spec necessary, but in theory an unmodified spec should work)
- building binary packages:
  - RPM format (mock necessary)
- uploading source packages to cloud build service:
  - copr for fedora, RHEL/EPEL and Mageia packages

CHAPTER 2

kentauros

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

k

Index

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

P

R

S

T

U

V

W