
Krrez

Release 10.0.1291

Author name not set

Jul 27, 2025

CONTENTS

1	License	3
2	Up-To-Date?	5
3	Dependencies	7
4	User Manual	9
4.1	Getting Started	9
4.2	Some Basic Terms	12
4.3	Advanced	13
5	Command Line Interface Reference	17
6	API Reference	19
6.1	krrez namespace	19
	Python Module Index	85
	Index	87

Krrez provides a foundation for putting the installation and configuration of computer systems into code. This allows to install those systems in an automatic, repeatable and testable way.

There are some jargon terms around, like ‘Infrastructure-as-code’, ‘Configuration management’ and even ‘DevOps’. Krrez addresses all those topics to some extent, but in a much different, lighter way than prevailing tools, as it does not really target the same use cases. In particular, it is designed with personal computer systems in mind instead of cloud systems.

Automation code is just plain Python code that eventually gets executed on a target system, and does things like package installations, modifications of configuration files, execution of command lines, and more. This code must be provided by you, at least for everything beyond some basic stuff that is included in Krrez.

The usual process for a Krrez-based system installation begins with transferring the Krrez installer to an external medium like a USB stick. This sounds harder than it is; in fact it is a guided and very simple step. Next to core operating system installation routines, this image also carries all your automation code. The next step is to boot the target machine from that medium. This will install an operating system (Debian Linux is the one that Krrez supports out of the box) and execute the automation code.

Everything after booting the target machine can be completely automatic, or can ask the user for some additional configuration infos on-the-fly, depending on how the installation image was set up.

Once that installation is finished, you will typically not use Krrez on that system for the rest of its lifetime. If you want to run an updated version of your automation code, with some modifications, or some added or removed behavior, Krrez’ philosophy is to just reinstall the affected system(s). This makes it much easier to write automation code on the hand, as you do not have to deal with bulky properties like idempotency or reversibility, but on the other hand makes it a bad choice for some kinds of use cases.

(At least it is possible with some extra work to enable an installed system to reinstall itself without a new installation medium or any other reason to physically get in touch with that machine.)

Automation code is structured in modules. It is possible to create installation media for different kinds of systems, each with an own subset of automation code modules, e.g. for your primary workstation, your notebook, network storage, home automation server, and so on.

Krrez also comes with a mechanism for testing. It runs the entire installation process on some internal Virtual Machines, then lets your test code make its checks on them. This can be as complex as the interaction between more than one machine (maybe based on different profiles).

LICENSE

Krrez is distributed under the terms of the AGPL 3 license. This also affects all included files without a license header (non-source files like images), unless they are explicitly mentioned as third-party content. Read the ‘Dependencies’ section for included third-party stuff.

UP-TO-DATE?

Are you currently reading from another source than the homepage? If you are in doubt whether your package is up-to-date, you should visit the project homepage and check that. You are currently reading the documentation for version 10.0.1291.

DEPENDENCIES

Krrez makes use of some third-party parts.



Required: **Python 3.13**



Required: **Python package fabric** `~= 3.0`



Required: **Python package hallyd** `~= 0.90`



Required: **Python package klovve** `~= 1.1`



Required: **Python package libvirt-python** `>= 9.0.0`



Required: **Python package paramiko** `~= 3.1`



Required: **Python package pygraphviz** `~= 1.10`



Required: **Python package watchdog** `~= 4.0`



Recommended: **GNU/Linux**



Included: **background image** (License: https://commons.wikimedia.org/wiki/Commons:GNU_Free_Documentation_License,_version_1.2; from here)



Included: **font ‘Inconsolata’** (for websites; by Raph Levien; License: OFL; [from here](#))



Included: **font ‘Khula’** (for websites; by Erin McLaughlin; License: OFL; [from here](#))



Included: **font ‘Symbola’** (for logo symbol; License: free for use; [from here](#))

USER MANUAL

4.1 Getting Started

4.1.1 The Krrez Executable

This section is about getting ready to start and use Krrez.

Note

Assuming that there is no severe software error (no guarantees for that; see license!) and you do not explicitly try to, none of the instructions from the documentation should do any harm to your system. Be cautious with own experiments before reading this documentation, though.

Install Krrez to any modern Linux workstation system in order to prepare it for writing your infrastructure automation code and for finally creating installation media:

```
pip install --upgrade krrez
```

If everything went fine, you are now able to call Krrez:

```
krrez --help
```

Krrez can be called in different ways. Some of them are available depending on the capabilities of your system.

For a graphical user interface, the following software requirements must be met:

- [PyGObject](#)
- [GTK 3](#) and its gir module (in Debian, just install ‘gir1.2-gtk-3.0’)
- [Evince](#) and its gir module, optionally, for displaying the documentation (in Debian, just install ‘gir1.2-evince-3.0’)

If you want to use a friendly but terminal based one instead, the following software requirements must be met:

- [TODO](#) curses?

If none of them are available, it is still possible to do all things, but with minimal convenience. This section assumes a graphical user interface, though. A later section describes in detail how to work with the command line interface.

4.1.2 Krrez Studio

The first thing we want to do now is to create some automation code and to include that in a Krrez Profile.

Start the Krrez Studio application by executing this command line:

```
krrez
```

You should then see Krrez' main user interface, which offers you a central place for all kinds of actions around using Krrez. For now, switch to the "Development" tab, which offers some starting points for the development of your automation code.

Create a new Bit and select it there.

We can see the path to the file that contains the scaffolding for your new automation code. Open that in a text editor in order to modify it now.

Note

The code that you are going to write here is to be executed when systems get freshly installed by Krrez. There is no danger for your workstation, as the code will not be executed directly there.

At first, find the place inside that file where you are advised to add your code. Add some code there. For example, you could add the following block:

```
with open("/foo", "w"): pass
```

This will create an empty file `foo` in the root directory of the system. You could do arbitrary complex things here, but this stupid example is just enough for the following steps. In reality, your code would probably install and configure a particular software, or adapt system configuration aspects. Also read more about `api` later, as it provides a lot of functionality that was not even touched here.

As the next step, switch to *Profiles* (see bar on the top) and create a new one. Add the bit that you created before to that Profile.

Now you have everything needed in order to turn a usb stick into a Krrez installer, which will install a fresh operating system on a target machine and run that automation code there. Without large amounts of customizing, this operating system will be a Debian Linux.

We will see later how to run a Krrez installation, and go on in Krrez Studio before.

Switch to *Tests* and create a new one. For most cases it is a good idea to use the same name as the bit that you are going to test (in our case the one from the first step). Add the Profile that you created before to that test. Then open the source code file for this test and find the place where you are advised to add your test code.

Warning

In contrast to the bit code from earlier steps, test code **will indeed be executed directly on your workstation**. So there is a potential danger of damaging it, which you should keep in mind. **Code here should not apply any changes to the system** (but just to the virtual test machines in a very particular way introduced later).

For our bit code example from above, we could add the following block to the test:

```
assert "foo" in foo.exec(["ls", "/"]) # TODO
```

As the final steps, switch to the *Test plans* and create one. Add your test as well as your Profile from the previous steps.

4.1.3 Testing

Let us now run our test. This will simulate the installation of Krrcz with your automation code to a fresh system and will run your test code. While the test code will be executed from your workstation directly, the target systems are a simulation based on virtual machines.

Switch to the “Testing” tab at first. Here you can choose your test and start it. If any kinds of errors come up, this will visibly fail, so you can solve the issue and run it again.

A successful run will take some time. The exact amount depends heavily on your machine. Consider that it installs a complete operating system, but in a virtual machine, and has to generate an installer medium for that before (which is, technically speaking, another complete operating system installation).

Once your test passes, you can be sure that your automation code does what you expect (as good as you were able to express your expectations in the test).

Also read about *Usage of System Resources*, if this is relevant for you.

Note

If you want to inspect the virtual machines that Krrcz uses to run the test, e.g. for problem analysis, you can find and open them with [virt-manager](#). During installation, you can log in as user `seed`, password `seed`. After the installation, it depends on what users and/or what login mechanisms your automation scripts have set up.

You could now go on implementing other bits, similar as before, and add them to one or more Profiles. This gives a way to modularize automation code, and to set up subsets of your bits of different use cases.

4.1.4 Seeding

Once you have something useful enough to deploy to a real machine, you can start with that by switching to the “Seeding” tab. Fill that form and create an installation medium. You can then use it in order to install a target machine. Follow the on-screen instructions for details.

That procedure of creating the installation medium (and to run the installation from that medium on a target machine) is called *seeding*.

4.1.5 Code in an IDE

Writing a few lines of code in a text editor is obviously possible. It is recommended to use more sophisticated tools, though, as it can improve your experience and productivity a lot.

Look on the web or in your application store for a development environment (‘IDE’) that supports the Python language. There are various free ones and commercial ones. Not all of the commercial ones cost money - nowadays there is a variety of subtle ways for users to ‘pay’ for things. Finding a great IDE is not trivial, though. If you are in doubt, check out ‘PyCharm’.

At first you need to find the root directory for custom Krrcz modules on your system. Take a look into the DevLab tool that you used in the [Getting Started](#). Select any element and find the source file path of it. The root directory we are looking for is everything before the `/krrcz/bits/...`

Any IDE should be able to open that directory (some will call it ‘import’) and handle it as a Python project. Try to do that in your one, and then open one of your bit files.

A good IDE will now display some problems. This is because your code makes use of Krrcz modules, but your IDE does not know about them. In order to solve that, you have to add the `src` subdirectory of your Krrcz directory to the list of Python package sources. That can be somewhere inside the IDE (the most simple way), but you could also append it to the `$PYTHONPATH` environment variable for that instead, or set up a Python virtualenv.

4.2 Some Basic Terms

After you got a first idea what Krrrez does, we will now take a deeper look into some technical terms.

4.2.1 Bits

One of the most important concepts in Krrrez are Bits. All your automation code is going to end up in Bits. Each Bit encapsulates code for a particular piece of automation you want to implement. It could be responsible for setting up a particular service, like a web or mail server, setting up a particular desktop application, or taking care of a particular system configuration aspect. There is no strict limit what a Bit can do, as it is arbitrary Python code. Each Bit has a name, describing what it does, like “apache”, “plasma_desktop” or “my_monitoring_tool”.

There are (practically speaking) no Bits included in Krrrez, but it is up to you to create and implement them, as you already did in the *Getting Started*. Bits can depend on each other and can even actually interact with each other in order to achieve complex things in a nice modular way.

Technically, a Bit is a Python module package. Its `__init__.py` contains the definition of a `Bit` class, as our initial sample Bit did, and resides somewhere inside the `krrrez.bits` Python package, so its full module name could be like `krrrez.bits.apache`. Your automation code is to be placed in the `__apply__` method of that class.

Beside the `__init__.py` file, there could be more things in that package: On the one hand, there is often a `-data` directory, which contains additional files used by the `apply` method. On the other hand, as Python packages are a tree structure, there can be more Bits. For example, there could be a `hardening` directory, building up the `krrrez.bits.apache.hardening` Bit package, just like it could be for any other Python module package. The way how you structure your Bits in a hierarchy is up to you and basically has to match your organizational preference. You can place Bits inside Bits, like mentioned above, in order to organize code in a way meaningful to you, but you can also structure them just in a simple and flat way.

Note

There is much more to say about Bits. TODO `__apply_foo__` and shit. If you want to understand inner pieces of Krrrez for some reason, also read `krrrez.bitling.Bitling`.

Dependencies

Bits can depend on each other. Imagine a Bit that sets up something which is based on something provided by another Bit. For example, a particular Bit might install some web application, into a web server that is installed by another Bit. Each Bit needs to specify what other Bits it depends on. Krrrez will automatically take care of those dependencies in the right order when it applies your Bits.

Dependencies can be specified like in this example:

```
class Bit(krrrez.api.Bit):  
  
    @krrrez.api._BeforehandDependency("some.other.bit")  
    def __apply__(self):  
        ...
```

Add one or more of those `_BeforehandDependency`-lines, one for each Bit that your Bit depend on. The Bit name is like the Python module name that contains the other Bit (i.e. what you would `import`), but without the prepending “`krrrez.bits.`”.

This will make sure that *some.other.bit* is installed before your one.

If you want to make sure that another Bit will be installed, but *after* your one, use `_AfterwardsDependency` instead of `_BeforehandDependency`. If you just want to control the relative order to another Bit, but without enforcing its inclusion, add `, is_optional=True`. You will not need those tricks very often, though.

Please note: There is another, completely different syntax for the same thing. Instead of the way from the last example, the dependency could instead be specified this way:

```
import krrrez.bits.some.other.bit

class Bit(krrrez.api.Bit):

    _some_other_bit: krrrez.bits.some.other.bit.Bit

    def __apply__(self):
        ...
```

Note the additional `import` line, the longer name on the right hand side of the colon, and that ‘new’ name on the left hand side. That latter name can be arbitrary, but by convention should start with “_” (you will see soon what it is for). There is also no way to use `_AfterwardsDependency` or `is_optional`.

This syntax allows to use `self._some_other_bit` in the body of your `__apply__` method, in order to call methods on that other Bit. This mechanism allows a Bit to provide (resp. use) additional methods for related subsequent steps, e.g. a web server Bit might provide an `install_web_application` method. If you do not want to access the other Bit this way, you should use the former syntax, as it is much more compact.

The former syntax is sometimes called “decoration-style” and the latter one “attribute-style”.

4.2.2 Profiles

A Profile is basically a subset of your Bits (it is actually slightly more, but we will see that later). Whenever you are going to install a new machine, you have to choose what Profile to install. You can have one Profile per machine, e.g. a particular selection of Bits for a storage server, and another one for a desktop machine.

4.2.3 Seeding

The procedure of installing a new machine (or make a fresh installation to an existing one) is called Seeding. It usually starts with generating a fresh installation medium. You can do that on any system that has Krrrez installed and your automation code (Bits, Profiles, ...) available. The medium will be prepared to contain the Profile that you have chosen to seed. Depending on architecture characteristics of your target machine, this is usually a bootable USB stick, but could also be an SD card for an IoT device. Following the instructions from Krrrez’ Seeding tool, you will then insert that medium into the target machine and start the machine from it. At the end of the procedure, your target machine is prepared with a freshly installed operating system and your automation code applied.

4.3 Advanced

4.3.1 Config

Sometimes there are things happening in Bits which need to be configurable. Or, in other words, you do not want to embed some pieces of data into your code, but allow it to be specified externally. Asking the user during installation for sensitive information like passwords is one of those scenarios. Another one is a Bit that you are going to use in more than one Profile, but in different ways (e.g. because the dev path to a particular hardware devices varies).

Your Bit can ask for such configuration values like in this example:

```
import krrrez.bits.sys.config

class Bit(krrrez.api.Bit):
```

(continues on next page)

(continued from previous page)

```
_con fig: krrrez.bits.sys.config.Bit

def __apply__(self):
    ...
    foo = self._config.ask_for("foo").input("What foo?")
    ...
```

Inside the `ask_for` call, a key for this configuration value is chosen (here "foo"). The variable name (i.e. the `foo` on the left hand side of the equal sign) is of course arbitrary. There is also a user prompt message inside the `input` call.

This line fetches a configuration value either by the specified key from an external configuration. That external configuration is usually the Profile. If the value is not specified there, it will ask the user.

TODO more than `.input` , confidentially

4.3.2 Builtin Automation Convenience Features

There are some builtin features for various kinds of automation tasks, like creating and controlling services, creating system users, convenient filesystem operations, and more. They are easy to be used in your Bit implementation. For example, you can just access `self._fs` and find filesystem operations there. Take a look into [krrrez.api.Bit](#) and see what features exist.

There is also `hallyd`, which contains lower level, generic additional functionality. It is not specific to infrastructure automation, but might be useful in some situations.

4.3.3 Technical Details

Usage of System Resources

Krrrez makes use of some kinds of operating system and machine resources during its operation. For instance, during testing, it temporarily creates virtual machines, loop devices, network bridges, and might consume a terrible amount of memory and disk space (depending on how many and what machines are part of your test). Other operations, like seeding a new installation medium, are usually less intense in that regard.

There will always remain a log of those actions, including time stamps and diagnostic output. For some of them (like test runs), you can browse their history in the Krrrez user interfaces and via command line.

Beyond those logs, Krrrez will take care of cleaning up everything it created temporarily. In some cases, e.g. if you unplug your machine while Krrrez is running tests, some of the temporary stuff will remain. However, it will get cleaned up next time you start Krrrez.

Note

There is one corner case, which probably does not apply to you at all, but is mentioned here for the sake of completeness: Whenever you run a test that contains Landmarks (the author of the test would know if this is the case), and you reboot your machine while such a test is running (and has already passed at least one Landmark), there will be Landmark data remaining. They allow to resume the test run from there. You can remove them manually with the Krrrez user interface or command line, or implement your own script that cleans them up.

About the API Reference

There is an API Reference at the end of this document, which lists and explains all relevant modules, classes, functions, ... that are part of Krrrez. There is no need to read all that. The documentation links to some pieces of it sometimes, though.

The only parts relevant for Krrrez automation code developers are [krrrez.api](#), builtin Bits in `krrrez.bits`, and maybe Profiles in [krrrez.profiles](#) that you are going to subclass.

In line with Python guidelines, members starting with one `'_'` are considered as non-public (even if they are listed in the API reference). If they are members of a class, they are usually only relevant inside subclass implementations (with a few internal exceptions).

COMMAND LINE INTERFACE REFERENCE

API REFERENCE

6.1 krrez namespace

6.1.1 Subpackages

krrez.api package

Main programming interface for the implementation of bits, i.e. your custom automation logic to be executed by Krrez when it seeds a system.

class `krrez.api.Bit`

Bases: [*MayDeclareSecondaryBitUsages*](#)

Base class for bits.

Put your automation code (i.e. what is to be executed on a Krrez target machine) in a new subclass. This subclass must either be named *Bit* (the common case), or at least have a name that ends with *Bit* (you should not do that without a particular reason). It must override `__apply__()` with your actual automation code and be put into a Python submodule somewhere inside `krrez.bits.`.

In order to define dependencies, there are some options. The most common one is to add a line like this to the top of your Bit class body:

```
_foo: krrez.bits.foo.Bit # by convention, the name is the last part of the module,
↳with an underscore appended
```

This line specifies a dependency, i.e. whenever your Bit is going to be applied, the infrastructure will apply `krrez.bits.foo.Bit` before. You either have to `import krrez.bits.foo` for that (recommended at least if you use a real IDE), or put quotes around the type name.

Beyond the specification of a dependency, this line has another effect: Your `__apply__()` method can use this other Bit, e.g. in order to call some of its service methods, like here:

```
def __apply__(self):
    my_thingy = ...
    self._foo.add_bar_handler(my_thingy)
```

For optional dependencies, see [*IfPicked*](#). For some other advanced ways, see [*Beforehand*](#), [*Later*](#) and [*Eventually*](#).

In order to define a configuration value, see [*ConfigValue*](#).

In order to define a lock (you only need them in some advanced cases), see [*Lock*](#).

Read more in the Krrez documentation.

Do not construct Bits directly.

_helpers: `Bit`

_fs: `Bit`

_packages: `Bit`

_services: `Bit`

_system_users: `Bit`

property name: `str`

The Bit name.

You usually do not need it.

property _data_dir: `Path`

The path of this Bit's data directory.

This is the “-data” subdirectory of the directory that contains your Bit module file.

property _log: `Logger`

The logger.

property _internals: `_Internals`

Internal features that are usually not needed to be used.

_bit_for_type(*bit_type*)

class _Internals(*bit*)

Bases: `object`

Parameters

bit (`Bit`)

property origin_bit: `Bit` | `None`

property session: `Session` | `None`

property dialog_endpoint: `Endpoint` | `None`

property log_block: `LogBlock` | `None`

initialize_as_secondary(*origin_bit*)

Parameters

origin_bit (`Bit`)

Return type

`None`

prepare_apply(*session, log_block, dialog_endpoint*)

Parameters

• **session** (`Session`)

• **log_block** (`LogBlock`)

• **dialog_endpoint** (`Endpoint`)

Return type

`None`

class krrez.api.Profile(* (*Keyword-only parameters separator (PEP 3102)*), *hostname, disks, raid_partitions, network_interfaces, krrez_bits, arch, operating_system, seed_strategy, keyboard, locale, timezone, seed_user, config*)

Bases: `object`

Base class for Profiles.

A profile collects all kinds of setup, including system settings like disk partitioning, and a list of Krrrez Bits to install. Whenever to install a new Krrrez system, the seeding procedure, and so the target system, is defined by the Profile you choose.

Parameters

- **hostname** (*str*)
- **disks** (*list[krrrez.bits.seed.steps.disks.Disk]*)
- **raid_partitions** (*list[krrrez.bits.seed.steps.disks.RaidPartition]*)
- **network_interfaces** (*list[krrrez.bits.seed.steps.networking.NetworkInterface]*)
- **krrrez_bits** (*list[str]*)
- **arch** (*str*)
- **operating_system** (*krrrez.bits.seed.common.OperatingSystem*)
- **seed_strategy** (*krrrez.seeding.SeedStrategy*)
- **keyboard** (*krrrez.bits.seed.steps.keyboard.Keyboard*)
- **locale** (*str*)
- **timezone** (*str*)
- **seed_user** (*krrrez.bits.seed.steps.seed_user.SeedUser | None*)
- **config** (*dict[str, Any]*)

classmethod `get(data)`

Return type

`Profile`

to_flow_config_dict()

Return type

`dict[str, Any]`

is_hidden = `False`

krrrez.api.Eventually = `<krrrez.api.internal.GenericDependencyAnnotation object>`

Specifies a list of Bits as no-order dependencies, i.e. enforcing the specified Bits to be applied when this is one will, no matter if afterward or beforehand, like this: `__eventually: krrrez.api.Eventually["krrrez.bits.foo.Bit", "krrrez.bits.bar.Bit", ...]`.

krrrez.api.Later = `<krrrez.api.internal.GenericDependencyAnnotation object>`

Specifies a list of Bits as reverse-order dependencies, i.e. similar to usual dependencies, but applying the specified Bits *after* the own Bit was applied, like this: `__later: krrrez.api.Later["krrrez.bits.foo.Bit", "krrrez.bits.bar.Bit", ...]`.

krrrez.api.Beforehand = `<krrrez.api.internal.GenericDependencyAnnotation object>`

A simpler way to specify multiple dependencies if you do not need them as dedicated property in Bit. `__apply__()`. You can annotate one property with it and specify multiple Bit types like this: `__more_deps: krrrez.api.Beforehand["krrrez.bits.foo.Bit", "krrrez.bits.bar.Bit", ...]`.

`krrez.api.IfPicked = typing.Optional`

Modifier for a dependency specification.

A dependency specification can be marked as optional by putting `krrez.api.IfPicked[...]` around the type. Such a dependency specification will influence the order, but it will not enforce the other Bit to be applied. If it is not at all in the list of Bits to be applied, this dependency specification will not make it part of the list.

Inside your `Bit.__apply__()` code, this property is `None` if that Bit was optional and not applied.

It may also be used in *Beforehand* and *Later* specifications.

Submodules

`krrez.api.internal` module

`class krrez.api.internal.MayDeclareSecondaryBitUsages(*, used_by=<function MayDeclareSecondaryBitUsages.<lambda>>)`

Bases: `object`

Subclasses of this class may declare the usage of other Bits by an annotated class attribute (like for dataclasses), and use them in a convenient, straight-forward way.

This mechanism is used in `krrez.api.Bit`, but also in a few other places.

static `_resolve_optional(annotation_value)`

Parameters

`annotation_value` (*Any*)

Return type

`tuple[Any, bool]`

classmethod `_all_bit_usage_declarations(*, all_bits=None)`

Parameters

`all_bits` (*Iterable[type[Bit]]* | *None*)

Return type

`dict[str, tuple[type[Bit] | None, str, Any]]`

static `_try_resolve_bit_type(bit_type, *, all_bits=None)`

Translate any way of attribute-style dependency to a Bit subclass.

Parameters

- `bit_type` (*str* | *type[Bit]*) – The Bit specification.
- `all_bits` (*Iterable[type[Bit]]*)

Return type

`type[Bit] | None`

`_bit_for_type(bit_type)`

`class krrez.api.internal.ProfileMeta`

Bases: `type`

Meta class for `krrez.api.Profile` and subclasses. It provides some metadata and useful information as class properties.

```
class _ProfileParameter(name: str, type: type[Any])
```

Bases: object

Parameters

- **name** (*str*)
- **type** (*type[Any]*)

name: *str*

type: *type[Any]*

property name: *str*

property open_parameters: *list[_ProfileParameter] | None*

property available_target_devices: *list[tuple[Path, str]]*

property is_browsable: *bool*

property is_hidden: *bool*

__get_block_dev_info()

Parameters

name (*str*)

Return type

str

```
class krrrez.api.internal.Dependency
```

Bases: object

A dependency is assigned to a Bit and can pull in other Bits and control the order of Bits when it gets applied.

In order to assign a dependency to a Bit, instantiate a subclass and decorate your apply method with it. This is called the “decoration-style” syntax in the documentation.

additional_needed_bits (*cooling_down, plan*)

A list of Bits that are pulled in by this dependency.

Parameters

- **cooling_down** (*bool*) – If the process is in cooling down mode.
- **plan** (*Plan*) – The resolution plan.

Return type

Iterable[str]

relative_order (*own_bit, other_bit*)

The relative order between two Bits, the ‘own’ one, and the other one.

Parameters

- **own_bit** (*type[Bit]*) – The own Bit.
- **other_bit** (*type[Bit]*) – The other Bit.

Returns

-1 if the own one has to come earlier, 1 if it has to come later, and 0 if it does not matter.

Return type

int

manipulate_resolution_plan(*owning_bit*, *plan*)

Apply custom manipulations to a resolution plan. Return True if a change was made.

This is only used for very particular internal tricks.

Parameters

- **owning_bit** (*type*[[Bit](#)]) – The Bit that this dependency is associated to.
- **plan** ([Plan](#)) – The resolution plan.

Return type

bool

class `krrez.api.internal.BaseForSimpleBehaviorDependency`(*bits*, *, *afterwards*)

Bases: [Dependency](#)

Base class for a simple (afterwards or beforehand) dependency.

Parameters

- **bits** (*Iterable*[*str*]) – The Bits to depend on.
- **afterwards** (*bool* | *None*) – Whether this is an afterwards-dependency.

__resolve_optional(*bit*)

__dependency_bits()

property *_bits*: *list*[*str*]

property *_optional_bits*: *list*[*str*]

additional_needed_bits(*cooling_down*, *plan*)

A list of Bits that are pulled in by this dependency.

Parameters

- **cooling_down** – If the process is in cooling down mode.
- **plan** – The resolution plan.

relative_order(*own_bit*, *other_bit*)

The relative order between two Bits, the ‘own’ one, and the other one.

Parameters

- **own_bit** – The own Bit.
- **other_bit** – The other Bit.

Returns

-1 if the own one has to come earlier, 1 if it has to come later, and 0 if it does not matter.

class `krrez.api.internal.SimpleDependency`(*bits*, *, *afterwards*=*False*)

Bases: [BaseForSimpleBehaviorDependency](#)

A simple dependency.

Parameters

- **bits** (*Iterable*[*str*]) – The Bits to depend on.
- **afterwards** (*bool* | *None*) – Whether this is an afterwards-dependency.

property bit_names: list[str]

The names of non-optional Bits to depend on.

property optional_bit_names: list[str]

The names of optional Bits to depend on.

property all_bit_names: list[str]

The names of optional and non-optional Bits to depend on.

property afterwards: bool

class krrrez.api.internal.ConnectableToBit

Bases: ABC

abstractmethod _connected_to_bit(*bit*, *key*)

Implement this method with logic to connect this object to the owning Bit.

Automatically called by the infrastructure when a ConnectableToBit attribute from *krrrez.api.Bit* is accessed.

Parameters

- **bit** (*Bit*) – The Bit to connect to.
- **key** (*str*) – The attribute key.

Return type

Self

_abc_impl = <_abc._abc_data object>

class krrrez.api.internal.Lock

Bases: *ConnectableToBit*

A multithreading/multiprocessing lock.

Can be used whenever code could be executed in parallel from multiple places, in order to synchronize access. Works beyond process barrier and is reentrant. Define a lock in the body of your Bit class like this:

```
_lock = krrrez.api.Lock()
```

You can use it inside your methods like this:

```
with self._lock:
    ...
```

Note: If you define multiple Bits in one module, and they define a lock with the same name, it will refer to the same lock.

_connected_to_bit(*bit*, *key*)

Implement this method with logic to connect this object to the owning Bit.

Automatically called by the infrastructure when a ConnectableToBit attribute from *krrrez.api.Bit* is accessed.

Parameters

- **bit** – The Bit to connect to.
- **key** – The attribute key.

```
_abc_impl = <_abc._abc_data object>

class krrez.api.internal.AdHocLogger
    Bases: Logger

    class _MessageMode
        Bases: LoggerMode

        _log(message, severity, aux_name)

        _abc_impl = <_abc._abc_data object>

    class _BlockMode
        Bases: _MessageMode

        _log(message, severity, aux_name)

        _abc_impl = <_abc._abc_data object>

    property block

    property message

    _abc_impl = <_abc._abc_data object>

class krrez.api.internal.BareConfigValue(*, default=<object object>, is_confidential=False,
                                         type=<class 'object'>, short_name=None,
                                         module_name=None, full_name=None, question=None)

    Bases: Generic[_TConfigValueType], ConnectableToBit

    Parameters

        • default (_TConfigValueType)

        • is_confidential (bool)

        • type (type[_TConfigValueType])

        • short_name (str | None)

        • module_name (str | None)

        • full_name (str | None)

        • question (_AskForDialog | None)

    _NoConfigValueDefault = <object object>

class _AskForDialog(method_name: str, args: Iterable, kwargs: dict)
    Bases: object

    Parameters

        • method_name (str)

        • args (Iterable)

        • kwargs (dict)

    method_name: str

    args: Iterable
```

```

    kwargs: dict
property type: type[_TConfigValueType]
property short_name: str | None
property module_name: str | None
property full_name: str | None
property default: _TConfigValueType | object
property is_confidential: bool
property question: \_AskForDialog | None
actual_full_name(bit, item)

```

Parameters

- **bit** ([Bit](#))
- **item** (*str*)

Return type

str

```
__ctrl()
```

Return type

[Controller](#)

```
_connected_to_bit(bit, key)
```

Implement this method with logic to connect this object to the owning Bit.

Automatically called by the infrastructure when a ConnectableToBit attribute from [krrcz.api.Bit](#) is accessed.

Parameters

- **bit** – The Bit to connect to.
- **key** – The attribute key.

```
class _Setter(value)
```

Bases: [Generic\[_TConfigValueType\]](#)

Parameters

value ([_TConfigValueType](#))

```
property __value
```

```
property value: _TConfigValueType
```

```
_abc_impl = <_abc._abc_data object>
```

```
class krrcz.api.internal._ConfigValueWithoutAskFor(*, default=<object object>,
                                                    is_confidential=False, type=<class 'object'>,
                                                    short_name=None, module_name=None,
                                                    full_name=None, question=None)
```

Bases: [BareConfigValue](#)[\[_TConfigValueType\]](#), [Generic\[_TConfigValueType\]](#)

Parameters

```
    • default (_TConfigValueType)
    • is_confidential (bool)
    • type (type[_TConfigValueType])
    • short_name (str | None)
    • module_name (str | None)
    • full_name (str | None)
    • question (_AskForDialog | None)
    _abc_impl = <_abc._abc_data object>

class krrcz.api.internal.ConfigValueAskForPrepared(*, default=<object object>,
                                                    is_confidential=False, type=<class 'object'>,
                                                    short_name=None, module_name=None,
                                                    full_name=None, question=None)

Bases: Generic[_TConfigValueType], BareConfigValue[_TConfigValueType]

Parameters
    • default (_TConfigValueType)
    • is_confidential (bool)
    • type (type[_TConfigValueType])
    • short_name (str | None)
    • module_name (str | None)
    • full_name (str | None)
    • question (_AskForDialog | None)

class _AskFor(config_value)
    Bases: Generic[_TConfigValueType], Processor[_ConfigValueWithoutAskFor[_TConfigValueType],
    \_ConfigValueWithoutAskFor[_TConfigValueType], \_ConfigValueWithoutAskFor[_TConfigValueType]],
    AllAbstractMethodsProvidedByTrick[Processor]

    Parameters
        config_value (ConfigValueAskForPrepared)

    has_default = None

    _ask_for()

    Return type
        \_AskFor[_TConfigValueType]

    _abc_impl = <_abc._abc_data object>

class krrcz.api.internal.ConfigValue(*, default=<object object>, is_confidential=False, type=<class
    'object'>)

Bases: ConfigValueAskForPrepared[_TConfigValueType], Generic[_TConfigValueType]

A configuration value.

Configuration values allow to make parts of your automation logic configurable. The actual value at runtime
can either be defined in the seed profile, or if not, is entered by the user during the seed procedure. Note that all
values must have a primitive data type (or be serializable by hallyd).
```


The way to specify a configuration value is to add a line like this to the top of your Bit class body:

```
_my_config = krrrez.api.ConfigValue(default="hello", type=str)
```

You can access the actual value inside your Bit.__apply__() method in some ways. It is readable:

```
my_config = self._my_config.value
```

And also writable:

```
with self._my_config as my_config:
    my_config.value = "ola"
```

When you read it, at there was no value specified e.g. in the seed profile, the behavior depends on whether you have specified a default. If yes, you will read this default value. If not, the user will be asked to enter it during installation.

For more control about how the user is asked to enter a value at installation time, see [ask_for](#), e.g. used like this:

```
_name = krrrez.api.ConfigValue(type=str).ask_for.input("Please enter the foo name.")
```

In general, with usual configuration, the internal key for your configuration value will be “[A].[B]” where [A] is the short name of the Bit where it is defined, but without the last part (i.e. “foo” for `krrrez.bits.foo.Bit`), and [B] is the name of the attribute, but with underscores stripped away on the left hand side (i.e. `my_config` for `_my_config`).

Parameters

- **default** (*TConfigValueType*) – The default value.
- **is_confidential** (*bool*) – Whether this value contains confidential information that must not be persisted.
- **type** (*type[TConfigValueType]*) – The value type. Mostly used for IDE guidance.

ask_for = <krrrez.api.internal.ConfigValueAskForPrepared._AskFor object>

_abc_impl = <_abc._abc_data object>

class `krrrez.api.internal.GenericDependencyAnnotation(**dependency_config)`

Bases: `object`

Special objects for some ways to define dependencies.

`krrrez.api.internal.usage_does_not_imply_a_dependency(bit_type)`

Mark a Bit type as not implying a dependency when used in an attribute-style dependency specification.

You usually do not need it.

Parameters

- **bit_type** (*type*) – The Bit type to mark.

Return type

type

krrez.asset package

`krrez.asset.bit(bit_type, *, used_by=None, skip_installed_check=False)`

Parameters

- **bit_type** (`type[_TBit]`)
- **used_by** (`Bit` | `None`)
- **skip_installed_check** (`bool`)

Return type

`_TBit`

Submodules

krrez.asset._pip module

krrez.asset.data module

`krrez.asset.data.readme_pdf(culture)`

Parameters

culture (`str`)

Return type

`Path`

krrez.asset.deploy_info module

krrez.asset.project_info module

krrez.coding package

Understand and modify some Krrrez specific code patterns, like `krrez.api.Bit` implementations.

This is used by applications like the Development Lab.

class `krrez.coding.Bits`

Bases: `object`

Code actions on `krrez.api.Bit` implementations.

APPLY_METHOD_NAME = `'__apply__'`

The name of the default apply method.

static `apply_method_name_to_name(apply_method_name)`

Parameters

apply_method_name (`str`)

Return type

`str` | `None`

static `is_bit_name_for_normal_bit(bit_name)`

Parameters

bit_name (`str`)

Return type

`bool`

static `is_special_apply_method_name(name)`

Parameters

`name` (*str*)

Return type

`bool`

static `editor_for_bit(bit)`

Parameters

`bit` (*Bit*)

Return type

`_Editor`

static `editor_for_new_bit(bit_name, module_base_directory)`

Parameters

`module_base_directory` (*Path*)

Return type

`_Editor`

class `_Editor(bit_name, module_base_directory)`

Bases: `Editor`

Parameters

- `bit_name` (*str*)
- `module_base_directory` (*Path*)

create()

Return type

`None`

class `krrez.coding.Profiles`

Bases: `object`

Code actions on [krrez.api.Profile](#) implementations.

static `editor_for_profile(profile)`

Parameters

`profile` (*type* [*Profile*])

Return type

`_Editor`

static `editor_for_new_profile(profile_name, module_base_directory)`

Parameters

- `profile_name` (*str*)
- `module_base_directory` (*Path*)

Return type

`_Editor`

```
class _Editor(profile_name, module_base_directory)
    Bases: Editor

    Parameters
        • profile_name (str)
        • module_base_directory (Path)

    _KRREZ_BITS_PATTERN = re.compile('krrrez_bits\\s*=\\s*(\\[[^]]*)')

    create()
        Return type
        None

    property krrrez_bits: Iterable[str] | None

    add_krrrez_bit(bit_name)
        Parameters
            bit_name (str)
        Return type
        None

    remove_krrrez_bit(bit_name)
        Parameters
            bit_name (str)
        Return type
        None

class krrrez.coding.ProfileTests
    Bases: object

    Code actions on Profile Tests (krrrez.api.Bit implementations with a special name).

    PROFILE_TEST_NAME_TO_BIT_NAME_PREFIX = 'zz_test.zz_profiles.'
    PROFILE_TEST_NAME_TO_SEED_BIT_NAME_POSTFIX = '.seed'

    static is_bit_name_for_profile_test(bit_name)
        Parameters
            bit_name (str)
        Return type
        bool

    static is_bit_name_for_profile_test_seed(bit_name)
        Parameters
            bit_name (str)
        Return type
        bool

    static bit_name_to_profile_test_name(bit_name)
        Parameters
            bit_name (str)
        Return type
        str
```

```

static profile_test_name_to_bit_name(profile_test_name)

    Parameters
        profile_test_name (str)

    Return type
        str

static bit_name_to_profile_test_seed_name(bit_name)

    Parameters
        bit_name (str)

    Return type
        str

static profile_test_seed_name_to_bit_name(profile_test_seed_name)

    Parameters
        profile_test_seed_name (str)

    Return type
        str

static editor_for_new_profile_test(profile_test_name, module_base_directory)

    Parameters
        • profile_test_name (str)
        • module_base_directory (Path)

    Return type
        _Editor

static editor_for_profile_test(profile_test)

    Parameters
        profile_test (Bit)

    Return type
        _Editor

class _Editor(profile_name, module_base_directory)
    Bases: Editor

    Parameters
        • profile_name (str)
        • module_base_directory (Path)

    create()

    Return type
        None

class krrrez.coding.Tests
    Bases: object

    Code actions on Tests (krrrez.api.Bit implementations with a special name).

    TEST_NAME_TO_BIT_NAME_PREFIX = 'zz_test.'

```

```
PROFILE_BIT_NAME_PREFIX = 'In'

static is_bit_name_for_test(bit_name)

    Parameters
        bit_name (str)

    Return type
        bool

static bit_name_to_test_name(bit_name)

    Parameters
        bit_name (str)

    Return type
        str

static test_name_to_bit_name(test_name)

    Parameters
        test_name (str)

    Return type
        str

static editor_for_test(test)

    Parameters
        test (Bit)

    Return type
        _Editor

static editor_for_new_test(test_name, module_base_directory)

    Parameters
        • test_name (str)
        • module_base_directory (Path)

    Return type
        _Editor

class _Editor(test_name, module_base_directory)
    Bases: Editor

    Parameters
        • test_name (str)
        • module_base_directory (Path)

    create()

    Return type
        None

    property profile_names: Iterable[str] | None

    add_profile(profile_name)

        Parameters
            profile_name (str)
```

Return type

None

remove_profile(*profile_name*)**Parameters****profile_name** (*str*)**Return type**

None

class krrrez.coding.TestPlans

Bases: object

Code actions on Test Plans ([krrrez.api.Bit](#) implementations with a special name).**TEST_PLAN_NAME_TO_BIT_NAME_PREFIX** = 'zz_test.zz_plans.'**static** **is_bit_name_for_test_plan**(*bit_name*)**Parameters****bit_name** (*str*)**Return type**

bool

static **bit_name_to_test_plan_name**(*bit_name*)**Parameters****bit_name** (*str*)**Return type**

str

static **test_plan_name_to_bit_name**(*test_plan_name*)**Parameters****test_plan_name** (*str*)**Return type**

str

static **editor_for_test_plan**(*test_plan*)**Parameters****test_plan** ([Bit](#))**Return type**[_Editor](#)**static** **editor_for_new_test_plan**(*test_plan_name*, *module_base_directory*)**Parameters****module_base_directory** ([Path](#))**Return type**[_Editor](#)**class** **_Editor**(*test_plan_name*, *module_base_directory*)Bases: [Editor](#)**Parameters**• **test_plan_name** (*str*)

- **module_base_directory** (*Path*)

create()

Return type
None

property tests: *Iterable[str]*

property test_plans: *Iterable[str]*

property profile_tests: *Iterable[str]*

add_test(*test_name*)

Parameters
test_name (*str*)

Return type
None

remove_test(*test_name*)

Parameters
test_name (*str*)

Return type
None

add_test_plan(*test_plan_name*)

Parameters
test_plan_name (*str*)

Return type
None

remove_test_plan(*test_plan_name*)

Parameters
test_plan_name (*str*)

Return type
None

add_profile_test(*profile_name*)

Parameters
profile_name (*str*)

Return type
None

remove_profile_test(*profile_name*)

Parameters
profile_name (*str*)

Return type
None

krrez.coding._module_root_directory(*path*)

Parameters
path (*Path*)

Return type
Path

krrez.coding._bit_name_to_specific_name(*bit_name*, *, *error_message*, *bit_to_specific_prefix*)

Parameters

- `bit_name(str)`
- `error_message(str)`
- `bit_to_specific_prefix(str)`

Return type

str

krrez.flow package

The entire mechanism involved in applying bits (like `krrez bits apply ...` commands) are implemented in this module and its submodules.

This module contains some foundation parts, but not the engine itself.

`krrez.flow.BITS_NAMESPACE = 'krrez.bits'`

The Python module namespace that contains `krrez.api.Bit` implementations.

`krrez.flow.KRREZ_ETC_DIR = Path('/etc/krrez')`

The base directory for Krrez etc data (system configuration files).

Parameters

`paths(str | Path)`

Return type

Path

`krrez.flow.KRREZ_USR_DIR = Path('/usr/local/krrez')`

The base directory for Krrez usr data (binary data, system resources).

Parameters

`paths(str | Path)`

Return type

Path

`krrez.flow.KRREZ_VAR_DIR = Path('/var/lib/krrez')`

The base directory for Krrez var data (variable data files).

Parameters

`paths(str | Path)`

Return type

Path

`krrez.flow._IS_KRREZ_MACHINE_FLAG_PATH = Path('/etc/krrez/is_krrez_machine')`

The flag file that indicates whether a system is a Krrez machine (i.e. was seeded and installed by Krrez).

Parameters

`paths(str | Path)`

Return type

Path

`krrez.flow.KRREZ_SRC_ROOT_DIR = Path('/usr/local/share/krrez')`

The root directory of Krrez sources, i.e. the 'krrez' Python package. Its name is `krrez`.

Parameters

`paths(str | Path)`

Return type

Path

class `krrez.flow.Context`(*path=None*)

Bases: `object`

A context is the place where configuration data, logs, and some other things are stored that will be used by Krrrez (associated with a context is a path, where all that data is stored).

For usual cases, like usual `krrez bits apply` runs, there is no choice for the end user, but it will always use its default location. However, internally, there will be contexts with other paths involved for some operations (like seeding and testing).

Parameters

path (*str* | *Path* | *None*)

DEFAULT_ROOT_PATH = `Path('/var/lib/krrrez/ctx')`

The default context path.

Parameters

paths (*str* | *Path*)

Return type

Path

_mark_bit_installed(*bit*)

Parameters

bit (*type*[`krrez.api.Bit`])

Return type

None

is_bit_installed(*bit*)

Parameters

bit (*type*[`krrez.api.Bit`])

Return type

bool

installed_bits()

Return type

list[*str*]

property path: *Path*

property config: `krrez.flow.config.Controller`

property parent_context: `Context` | *None*

_blank_contexts_path()

Return type

Path

_locals_path()

Return type

Path

__context_path_to_magic_file_path()

Parameters

context_path (*Path*)

Return type*Path***get_sessions()****Return type**list[[Session](#)]**get_blank_contexts()****Return type**list[[Context](#)]**lock**(*lock_name*, *, *ns=None*)**Parameters**

- **lock_name** (*str*)
- **ns** (*SupportsQualifiedName* | *None*)

Return type*Lock***_interaction_request_fetcher_for_session**(*session*, *provider*)

krrez.flow.create_blank_context(*in_context=None*, *, *inherit_config_values=False*,
blank_context_path=None)

Parameters

- **in_context** ([Context](#) | *None*)
- **inherit_config_values** (*bool*)
- **blank_context_path** (*Path* | *None*)

Return type[Context](#)**class** **krrez.flow.Session**Bases: [ABC](#)

Each bit apply run, either trigger by usual “krrez bits apply ...” or internally by some other procedure, is associated to one separate session.

A session itself is associated to a context, where it retrieves configuration values from, stores logging, and more.

static **create**(*, *context*)**Parameters****context** ([Context](#))**Return type**[Session](#)**static** **by_name**(*name*, *, *context*)**Parameters**

- **name** (*str*)
- **context** ([Context](#))

Return type*Session***abstract property context:** *Context***abstract property name:** *str***abstract property path:** *Path***property exists:** *bool***static name_is_valid**(*name*)**Parameters****name** (*str*)**Return type***bool***static _sessions_path**(*context*)**Parameters****context** (*Context*)**Return type***Path***static _all_sessions_path**(*context*)**Parameters****context** (*Context*)**Return type***Path***_abc_impl** = <_abc._abc_data object>**class** *krrez.flow._Session*(*context, name*)Bases: *Session*

Session implementation.

Parameters

- **context** (*Context*)
- **name** (*str*)

property context**property name****property path****property path2****_abc_impl** = <_abc._abc_data object>*krrez.flow.is_krrrez_machine*()

Whether this machine is enabled to be a Krrrez machine.

This is a safety feature which helps preventing unintended execution of bits.

See also *mark_as_krrrez_machine*()

Return type

bool

`krrez.flow.mark_as_krrrez_machine(system_root_path)`

Enables a system to be a Krrrez machine, allowing things like applying bits.

See also [`is_krrrez_machine\(\)`](#).

Parameters

system_root_path (*Path*) – The root path of the system to enable (assuming that it is mounted for preparation somewhere inside the host filesystem).

Return type

None

Subpackages**krrez.flow.graph package**

Bit graph implementation, primarily used with the [`krrez.flow.graph.resolver`](#).

class `krrez.flow.graph.Node`(*bit*)

Bases: object

One node in a bit graph, holding one bit, and pointing to all nodes that are considered as prerequisite of this node.

There is no separate class for an entire graph, but you consider its root node as the graph.

Such a graph is usually the result of dependency resolution for a list of bits to be installed. You usually do not create it manually, but get the graph from [`krrez.flow.graph.resolver`](#).

The root node is the final node regarding the execution order, while the leaves are the ones that execution can start with.

Parameters

bit (*type*[[`krrez.api.Bit`](#)] | *None*) – The Bit to hold.

property after: `list`[[`Node`](#)]

The (editable) list of nodes that need to applied before this one can get applied.

property bit: `type`[[`Bit`](#)] | *None*

The bit associated to this node. *None* for the root node.

nodes_reachable_from(*nodes*)

All nodes in this graph that are directly reachable, assuming that some nodes are already reached.

Parameters

nodes (*Iterable*[[`Node`](#)]) – The nodes that are already reached.

Return type*Iterable*[[`Node`](#)]

all_descendants(***, *starting_from_node=True*, *including_self=False*)

All descendants of this node.

Parameters

- **starting_from_node** (*bool*) – Whether to start iterating from this node (instead of iterating in earliest-reachable-first order).
- **including_self** (*bool*) – Whether to include this node as well (instead of, actually, just descendants).

Return type*Iterable*[*Node*]**condense**(**, exclude_if*)

Removes some nodes from the graph, connecting its after-nodes to the predecessors.

Parameters

exclude_if (*Callable*[[*Node*], *bool*]) – Function that decides if a node is to be removed.

Return type*None***node_for_bit**(*bit*)

The node in this graph that is associated to a given Bit (or *None* if not found).

Parameters

bit (*type*[*Bit*]) – The bit to find.

Return type*Node* | *None***__all_descendants__helper**()**Return type***list*[*tuple*[*Node*, *Node*]]**__remove_node**(*node*)**Parameters**

node (*Node*)

Return type*None***exception** *krrez.flow.graph.GraphError*

Bases: *RuntimeError*

An error in graph processing occurred.

Submodules

krrez.flow.graph.resolver module

Dependency resolution for bits.

krrez.flow.graph.resolver.graph_for_bits(*bits*)

A graph of bits, containing the given bits and their dependencies, connected in a valid way.

Parameters

bits (*Iterable*[*type*[*Bit*]]) – The bits that the graph at least have to contain.

Return type*Node**krrez.flow.graph.resolver._expand_by_dependencies_and_list_manipulations*(*plan*)

Expand a plan by dynamical mechanisms, in order to refine the plan.

Parameters

plan (*Plan*) – The plan to adapt.

Return type

None

```
krrez.flow.graph.resolver._fill_dependencies_to_nodes(root_node, plan)
```

Transfer the dependencies from a plan into a graph.

Parameters

- **root_node** ([Node](#)) – The graph to adapt.
- **plan** ([Plan](#)) – The plan to take the dependencies from.

Return type

None

```
class krrez.flow.graph.resolver.Plan
```

Bases: [ABC](#)

Data structure for dependency resolution, keeping all required bits and their dependencies.

This is turned into a graph after dependency resolution has finished.

```
abstractmethod add_bit(bit)
```

Adds a bit, so make it part of the later graph.

Parameters

bit (*type*[[Bit](#)]) – The Bit to add.

Return type

None

```
abstractmethod bit_by_name(bit_name)
```

The bit by name (or None if not found).

Parameters

bit_name (*str*) – The bit name.

Return type

type[[Bit](#)] | None

```
abstractmethod all_bits()
```

All bits currently listed to become part of the later graph.

Return type

Iterable[*type*[[Bit](#)]]

```
abstractmethod dependencies_for_bit(bit)
```

The (editable) list of dependencies for a bit.

Parameters

bit (*type*[[Bit](#)]) – The bit.

Return type

list[[Dependency](#)]

```
_abc_impl = <_abc._abc_data object>
```

```
class krrez.flow.graph.resolver._Plan
```

Bases: [Plan](#)

[Plan](#) implementation.

add_bit(*bit*)

Adds a bit, so make it part of the later graph.

Parameters

bit – The Bit to add.

bit_by_name(*bit_name*)

The bit by name (or None if not found).

Parameters

bit_name – The bit name.

all_bits()

All bits currently listed to become part of the later graph.

dependencies_for_bit(*bit*)

The (editable) list of dependencies for a bit.

Parameters

bit – The bit.

_abc_impl = <_abc._abc_data object>

krrez.flow.graph.resolver._may_be_installed_before(*bit, other_bit, plan*)

Whether a bit can be installed before another bit, according to a plan.

Parameters

- **bit** (*type*[[Bit](#)]) – The first bit.
- **other_bit** (*type*[[Bit](#)]) – The other bit.
- **plan** ([Plan](#)) – The plan to consider.

Return type

bool

krrez.flow.graph.resolver._cleanup_dependencies_from_nodes(*root_node*)

Removes all dependencies from a graph that are redundant.

Parameters

root_node ([Node](#)) – The graph to clean-up.

Return type

None

exception **krrez.flow.graph.resolver.DependenciesCannotBeMetError**(*details*)

Bases: [RuntimeError](#)

Error in realizing dependencies.

Parameters

details (*str*)

krrez.flow.graph.visualizer module

Visualizing for [krrez.flow.graph](#).

krrez.flow.graph.visualizer.try_dump_pygraphviz(*node, extra_data*)

An SVG image that visualizes the given graph (or None if pygraphviz is not available).

Parameters

- **node** ([Node](#)) – The graph to visualize.
- **extra_data** (*dict[str, str]*) – A dictionary that keeps the current stage for each node.

Return type

bytes | None

```
krrez.flow.graph.visualizer.__try_dump_pygraphviz__dive(pygraphviz_graph, node, seen, extra_data)
```

Parameters

- **node** ([Node](#))
- **seen** (*set[Node]*)
- **extra_data** (*dict[str, str]*)

```
krrez.flow.graph.visualizer.__try_dump_pygraphviz__add_pygraphviz_node(pygraphviz_graph,
                                                                        node, extra_data)
```

Parameters

- **node** ([Node](#))
- **extra_data** (*dict[str, str]*)

```
krrez.flow.graph.visualizer._label(node)
```

The label string for a node.

Parameters**node** ([Node](#)) – The node.**Return type**

str

```
krrez.flow.graph.visualizer._border_color(node, extra_data)
```

The border color for a node.

Parameters

- **node** ([Node](#)) – The node.
- **extra_data** (*dict[str, str]*) – Dictionary with state data per node.

Return type[Color](#)

```
krrez.flow.graph.visualizer._fill_color(node, extra_data)
```

The fill color for a node.

Parameters

- **node** ([Node](#)) – The node.
- **extra_data** (*dict[str, str]*) – Dictionary with state data per node.

Return type[Color](#)

```
class krrez.flow.graph.visualizer.Color(red, green, blue)
```

Bases: object

A color.

Parameters

- **red** (*float*) – The red component between 0 and 1.
- **green** (*float*) – The green component between 0 and 1.
- **blue** (*float*) – The blue component between 0 and 1.

__trim_value()

The original value trimmed to the interval 0..1.

Parameters

value (*float*) – The value to trim.

Return type

float

property red: float

The red component between 0 and 1.

property green: float

The green component between 0 and 1.

property blue: float

The blue component between 0 and 1.

property as_html: str

The html color representation of this color.

scaled_by(*factor*)

A variant of this color, scaled by a given factor.

Parameters

factor (*float*) – The factor. Lower than 1 makes it darker, higher than 1 makes it brighter.

Return type

[Color](#)

Submodules

[krrrez.flow.bit_loader module](#)

Finding and loading bits.

`krrrez.flow.bit_loader.all_bits(*, accept_cached=False)`

All bits.

See also [all_normal_bits\(\)](#).

Parameters

accept_cached (*bool*)

Return type

Iterable[*type*[[Bit](#)]]

`krrrez.flow.bit_loader.all_normal_bits()`

All normal bits.

Excludes stuff like tests. See also [all_bits\(\)](#).

Return type

Iterable[*type*[[Bit](#)]]

`krrez.flow.bit_loader.bit_by_name(name, all_bits=None)`

A bit by name. Raises `BitNotFoundError` if not found.

If the name contains `‘.SPECIAL.’`, it returns a special no-op bit. You usually do not need that; it is solely used internally.

Parameters

- **name** (*str*) – The bit name.
- **all_bits** (*Iterable[type[Bit]] | None*)

Return type

`type[Bit]`

`krrez.flow.bit_loader.bit_name(bit)`

The (short) name of a bit.

It is a substring of the type’s full name: The prefix `“krrez.bits.”` is removed, and if the class name equals to `“Bit”`, the `“.Bit”` postfix is removed as well.

See also `bit_full_name()`.

Parameters

bit (*str | Bit | type[Bit]*) – The bit. Can be a short or long name, a type or an instance.

Return type

`str`

`krrez.flow.bit_loader.bit_full_name(bit)`

The full name of a bit (equivalent to the type’s full name incl. module name).

See also `bit_name()`.

Parameters

bit (*str | Bit | type[Bit]*) – The bit. Can be a short or long name, a type or an instance.

Return type

`str`

`krrez.flow.bit_loader.bit_module_path(bit)`

The path of the Python module that contains this bit.

Parameters

bit (*Bit | type[Bit]*)

Return type

Path

`krrez.flow.bit_loader.bit_documentation(bit)`

The documentation text of this bit.

Parameters

bit (*Bit | type[Bit]*)

Return type

`str`

`krrez.flow.bit_loader.bit_for_secondary_usage(bit_type, *, used_by)`

A bit by bit type, for secondary usage, i.e. not for calling its apply method.

Parameters

- **bit_type** (*type[Bit]*) – The bit type to instantiate.

- **used_by** ([Bit](#) / *None*) – The bit that currently runs its apply method (or *None*). See `krrrez.api.Bit._internals.origin_bit`.

Return type[Bit](#)`krrrez.flow.bit_loader._krrrez_module_directories__helper(with_builtin)`**Parameters****with_builtin** (*bool*)**Return type**`list[Path]``krrrez.flow.bit_loader.krrrez_module_directories(*, with_builtin)`

All Krrrez module root directories.

Parameters**with_builtin** (*bool*) – Whether to include the builtin one as well.**Return type**`list[Path]``krrrez.flow.bit_loader._join_package_name(*name_segments)`

Join package names, e.g. "foo" and "bar.baz" to "foo.bar.baz".

Parameters**name_segments** (*str* / *None*) – The name segments to join.**Return type***str*`krrrez.flow.bit_loader._is_valid_bit_name_segment(name_segment)`

Whether a string is a valid name segment.

Parameters**name_segment** (*str*) – The name segment to check.**Return type***bool*`krrrez.flow.bit_loader._all_bits(*, accept_cached=False)`**Parameters****accept_cached** (*bool*)**Return type***Iterable*[`type`[[Bit](#)]]`krrrez.flow.bit_loader._all_bits__helper(sub_package_name)`**Parameters****sub_package_name** (*str* / *None*)**Return type***Iterable*[`type`[[Bit](#)]]`krrrez.flow.bit_loader._all_bits__helper__deep(module_name, sub_package_name)`**Return type***Iterable*[`type`[[Bit](#)]]

exception `krrez.flow.bit_loader.BitNotFoundError(bit_name)`

Bases: `RuntimeError`

A bit was tried to access that does not seem to exist.

Parameters

`bit_name` (*str*)

krrez.flow.config module

`krrez.flow.config._config_dir_path_for_context_path(context_path)`

Parameters

`context_path` (*Path*)

Return type

Path

class `krrez.flow.config.Controller`

Bases: `ABC`

Read and modify configuration values.

This is a low-level mechanism and not what a Bit would use (see `krrez.api.ConfigValue`). It does not include user dialogs.

abstract property context: *Context*

abstractmethod `get(key, default=None)`

Get the configuration value for a key.

Parameters

- **`key`** (*str*) – The key.
- **`default`** (*Any* | *None*) – The default value, if the key does not exist.

Return type

Any | *None*

abstractmethod `set(key, value, *, confidentially=False)`

Set the configuration value for a key.

Parameters

- **`key`** (*str*) – The key.
- **`value`** (*Any* | *None*) – The new value.
- **`confidentially`** (*bool*)

Return type

None

abstractmethod `lock(key)`

Return a context manager that locks a particular key.

Parameters

`key` (*str*) – The key.

Return type

ContextManager[*None*, *bool* | *None*]

abstractmethod available_keys(*, *with_confidential=False*)

All keys that are stored.

Parameters

with_confidential (*bool*)

Return type

list[str]

static key_is_valid(*key*)

Parameters

key (*str*)

Return type

bool

static _verify_key_is_valid(*key*)

Parameters

key (*str*)

Return type

None

_abc_impl = <_abc._abc_data object>

class krrrez.flow.config._Controller(*context*)

Bases: [Controller](#)

Parameters

context ([krrrez.flow.Context](#))

property context

lock(*key*)

Return a context manager that locks a particular key.

Parameters

key – The key.

get(*key, default=None*)

Get the configuration value for a key.

Parameters

- **key** – The key.
- **default** – The default value, if the key does not exist.

set(*key, value, *, confidentially=False*)

Set the configuration value for a key.

Parameters

- **key** – The key.
- **value** – The new value.

available_keys(*, *with_confidential=False*)

All keys that are stored.

_abc_impl = <_abc._abc_data object>

```
class krrez.flow.config._InteractiveController(original_controller, dialog_endpoint)
```

```
    Bases: Controller
```

Parameters

- **original_controller** ([Controller](#))
- **dialog_endpoint** ([krrez.flow.dialog.Endpoint](#))

```
class _AskFor(controller, dialog_endpoint, key, confidentially)
```

```
    Bases: Endpoint, AllAbstractMethodsProvidedByTrick
```

Parameters

- **controller** ([_InteractiveController](#))
- **dialog_endpoint** ([krrez.flow.dialog.Endpoint](#))
- **key** (*str*)

```
class _None
```

```
    Bases: object
```

```
    has_default = None
```

```
ask_for(key, *, confidentially=False)
```

Parameters

- **key** (*str*)
- **confidentially** (*bool*)

Return type

[Endpoint](#)

property context

```
lock(key, default=None)
```

Return a context manager that locks a particular key.

Parameters

key – The key.

```
get(key, default=None)
```

Get the configuration value for a key.

Parameters

- **key** – The key.
- **default** – The default value, if the key does not exist.

```
set(key, value, *, confidentially=False)
```

Set the configuration value for a key.

Parameters

- **key** – The key.
- **value** – The new value.

```
available_keys(*, with_confidential=False)
```

All keys that are stored.

```
_abc_impl = <_abc._abc_data object>
```

```
krrez.flow.config.controller(*, context)
```

A (non-interactive) config controller for a context.

Parameters

context ([Context](#)) – The context to access.

Return type

[Controller](#)

```
krrez.flow.config.interactive_controller(*, original_controller, dialog_endpoint)
```

An interactive config controller, wrapping a non-interactive one with a dialog endpoint.

Parameters

- **original_controller** ([Controller](#)) – The original controller to wrap.
- **dialog_endpoint** ([Endpoint](#)) – The dialog endpoint.

Return type

[_InteractiveController](#)

```
krrez.flow.config.new_config_server_for_session(fresh_session_path)
```

An IPC object-ref to a config controller for a fresh session.

Parameters

fresh_session_path ([Path](#)) – The session path.

Return type

[Server](#)

```
krrez.flow.config.config_client_for_existing_session(session_path)
```

An IPC object-ref to a config controller for an existing session.

Parameters

session_path ([Path](#)) – The session path.

Return type

[Controller](#)

```
krrez.flow.config._ipc_config_object_path(session_path)
```

Parameters

session_path ([Path](#))

Return type

[Path](#)

krrez.flow.dialog module

User interaction facilities for usage inside [krrez.api.Bit](#) apply methods.

```
class krrez.flow.dialog.Style(*values)
```

Bases: Enum

NORMAL = 1

CAUTION = 2

CRITICAL = 3

class krrrez.flow.dialog.Processor

Bases: Generic[_TInputResult, _TSinglePickResult, _TMultiPickResult], ABC

Abstract data type for a type that has a method for each kind of user interaction, with any return types.

abstractmethod `input(question, *, suggestion="", multi_line=False, is_valid_regexp=None, invisible=False, additional_text="", style=Style.NORMAL)`

Parameters

- `question` (*str*)
- `suggestion` (*str*)
- `multi_line` (*bool*)
- `is_valid_regexp` (*str* | *None*)
- `invisible` (*bool*)
- `additional_text` (*str*)
- `style` (*Style*)

Return type

_TInputResult

abstractmethod `choose(question, *, choices, additional_text="", style=Style.NORMAL)`

Parameters

- `question` (*str*)
- `choices` (*dict*[*str*, *object* | *None*] | *list*[*object* | *None*])
- `additional_text` (*str*)
- `style` (*Style*)

Return type

_TSinglePickResult

abstractmethod `pick_single(question, choices, *, additional_text="", style=Style.NORMAL)`

Parameters

- `question` (*str*)
- `choices` (*dict*[*str*, *object* | *None*] | *list*[*object* | *None*])
- `additional_text` (*str*)
- `style` (*Style*)

Return type

_TSinglePickResult

abstractmethod `pick_multi(question, choices, *, additional_text="", style=Style.NORMAL)`

Parameters

- `question` (*str*)
- `choices` (*dict*[*str*, *object* | *None*] | *list*[*object* | *None*])
- `additional_text` (*str*)
- `style` (*Style*)

Return type*_TMultiPickResult***_abc_impl** = <_abc._abc_data object>**class** krrrez.flow.dialog.**Endpoint**Bases: *Processor*[str | None, int | None, list[int] | None], ABC

Abstract data type for a type that has a method for each kind of user interaction, with their usual return types.

Subclasses of this type are typically used by client code that wants to actually show a dialog to the user and to get back the user's answer.

_abc_impl = <_abc._abc_data object>**class** krrrez.flow.dialog.**Provider**Bases: *Processor*[Future[str | None], Future[int | None], Future[list[int] | None]], ABCAbstract data type for a type that has a method for each kind of user interaction, with an *asyncio.Future* of their usual return types as return types.

Subclasses of this type typically implement the frontend side of dialogs, i.e. the actual user interface.

Its method returns Future objects that hold a result once the user has answered the dialog, or it was cancelled.

_abc_impl = <_abc._abc_data object>**class** krrrez.flow.dialog.**_DialogRequest**(*method_name, args, kwargs*)

Bases: object

property *method_name***property** *args***property** *kwargs***class** krrrez.flow.dialog.**Hub**(*path*)Bases: Hub[_*DialogRequest*, Any | None]

Abstract base class for an IPC hub of dialog requests.

Parameters**path** (*Path*) – The path where to start the server. It must not exist yet. The filesystem must support sockets. This path will be accessible by everyone. Make sure to restrict access properly by the permissions of its super-directories!**class** krrrez.flow.dialog.**HubEndpoint**(*dialog_hub*)Bases: *Endpoint*, AllAbstractMethodsProvidedByTrick[*Endpoint*]Dialog endpoint implementation that passes interaction requests through a *Hub*.**Parameters****dialog_hub** (*Hub*)**has_default** = None**class** krrrez.flow.dialog.**_InteractionRequestFetcher**(**args, provider, **kwargs*)Bases: HubWorker[_*DialogRequest*, Any | None]**Parameters****provider** (*Provider*)

```
request_arrived(request)
request_disappeared(request)
```

krrez.flow.logging module

Logging interface.

```
class krrez.flow.logging.LoggerMode
```

Bases: ABC, Generic[_TLoggerModeResult]

```
debug(message, *, aux_name="")
```

Parameters

- **message** (*str*)
- **aux_name** (*str*)

Return type

_TLoggerModeResult

```
info(message, *, aux_name="")
```

Parameters

- **message** (*str*)
- **aux_name** (*str*)

Return type

_TLoggerModeResult

```
warning(message, *, aux_name="")
```

Parameters

- **message** (*str*)
- **aux_name** (*str*)

Return type

_TLoggerModeResult

```
fatal(message, *, aux_name="")
```

Parameters

- **message** (*str*)
- **aux_name** (*str*)

Return type

_TLoggerModeResult

```
abstractmethod _log(message, severity, aux_name)
```

Parameters

- **message** (*str*)
- **severity** (*Severity*)
- **aux_name** (*str*)

Return type*_TLoggerModeResult*`_abc_impl = <_abc._abc_data object>``class krrrez.flow.logging.Logger`

Bases: ABC

`abstract property block: LoggerMode[ContextManager[Logger, bool | None]]``abstract property message: LoggerMode[None]``_abc_impl = <_abc._abc_data object>``class krrrez.flow.logging.Severity(*values)`

Bases: Enum

`DEBUG = 1``INFO = 2``WARNING = 3``FATAL = 4`**krrrez.flow.runner module**

The engine.

`class krrrez.flow.runner.Engine`

Bases: object

Mechanism that applies (i.e. “installs”) some Bits to the current system.

It handles dependency resolution internally. It holds an internal `:py:class`krrrez.flow.writer.Writer`` for each run that feeds a listen and control interface with data, which can be consumed by a *krrrez.flow.watch.Watch*. It can be customized for special cases by subclassing.

`_worker_pool_size = 1``start(*, context, bit_names, log_block=None)`

Starts the applying procedure for a given list of Bits.

Parameters

- **context** (*Context*) – The context.
- **bit_names** (*Iterable[str]*) – The list of Bits to install.
- **log_block** (*LogBlock* | *None*) – The optional log block (from another session) to use as root log block.

Return type*Watch*`static _start_helper_s(*, pickled_engine, session_name, context_root_path, bit_names, pickled_log_block)`

_start_helper(**context*, *bit_names*, *session_name*, *log_block*)

Parameters

- **context** (*Context*)
- **bit_names** (*List[str]*)
- **log_block** (*LogBlock* | *None*)

Return type

None

_do_finish(*session*, *runner_writer*, *success*)

Parameters

- **session** (*Session*)
- **runner_writer** (*Writer*)
- **success** (*bool*)

Return type

None

_do_prepare(*session*, *install_bits*)

Parameters

session (*Session*)

Return type

None

_apply_message(*bit_name*)

Parameters

bit_name (*str*)

Return type

str

__engine_lock(*context*, *runner_writer*)

Parameters

runner_writer (*Writer*)

__compute_bit_graph(*context*, *runner_writer*, *chosen_bits*)

Parameters

- **runner_writer** (*Writer*)
- **chosen_bits** (*Iterable[type[Bit]]*)

class _WorkerLoop(*engine*, *runner_writer*, *bit_graph*, *install_bits*, *context*, *session*)

Bases: *object*

Parameters

- **engine** (*Engine*)
- **runner_writer** (*krrez.flow.writer.Writer*)
- **install_bits** (*list[type[krrez.api.Bit]]*)

```
property failed_bits: list[type[Bit]]
property is_running: bool
property was_successful: bool | None

run()
    Return type
    None

_WorkerLoop__finish_worker(worker)
    Parameters
    worker (_ApplyTask)
    Return type
    None

_WorkerLoop__installable_bits(bit_graph, installed_bits)
    Parameters
    • bit_graph (Node)
    • installed_bits (list[type[Bit]])
    Return type
    list[type[Bit]]

_WorkerLoop__refresh_bit_graph()

_WorkerLoop__start_worker(bit)
    Parameters
    bit (type[Bit])
    Return type
    _ApplyTask

_do_install_bit(bit, log_block, session, runner_writer)
    Parameters
    bit (type[Bit])
    Return type
    _ApplyTask

static _do_install_bit_helper(*, context_root_path, session_name, bit_name, picked_log_block,
                             dialog_hub_server_ipc_path)
    Parameters
    • context_root_path (Path)
    • session_name (str)
    • bit_name (str)
    • picked_log_block (bytes)
    • dialog_hub_server_ipc_path (Path)

class krrcz.flow.runner._ApplyTask(bit, log_block, process)
    Bases: object
    Parameters
    • bit (type[krrcz.api.Bit])
    • process (Popen)
```

property bit: type[*Bit*]

property log_block

property _process

class PipeReaderThread(*log_block, process*)

Bases: Thread

This constructor should always be called with keyword arguments. Arguments are:

group should be None; reserved for future extension when a ThreadGroup class is implemented.

target is the callable object to be invoked by the run() method. Defaults to None, meaning nothing is called.

name is the thread name. By default, a unique name is constructed of the form “Thread-N” where N is a small decimal number.

args is a list or tuple of arguments for the target invocation. Defaults to ().

kwargs is a dictionary of keyword arguments for the target invocation. Defaults to {}.

If a subclass overrides the constructor, it must make sure to invoke the base class constructor (Thread.__init__()) before doing anything else to the thread.

Parameters

process (*Popen*)

run()

Method representing the thread’s activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object’s constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

property error: bool

property is_running: bool

kill()

Return type

None

terminate()

Return type

None

wait_terminated(*timeout=None*)

Return type

bool

exception krrcz.flow.runner.WasUnsuccessfulError

Bases: Exception

exception krrcz.flow.runner.WasInterruptedError

Bases: *WasUnsuccessfulError*

krrez.flow.ui module

krrez.flow.watch module

Watching an engine session.

```
class krrez.flow.watch.Watch(session, *, log_block_arrived_handler=None,  
                             log_block_changed_handler=None, status_changed_handler=None,  
                             bit_graph_image_changed_handler=None)
```

Bases: object

Watches a *krrez.flow.Session*, e.g. in order to make it visible to the user.

Each session is associated to an execution of a *krrez.flow.runner.Engine*. The data that it reads is written by a *krrez.flow.writer.Writer*.

The actual watching occurs only in activated state, which is inside a `with` block. You can access all properties even without ever having the watch activated, but refresh only takes place while activated, and handlers will be called only then as well (this even means: you might never receive any log messages or similar before activating, as they come via a handler).

Parameters

session (*Session*) – The session to watch.

abort()

wait()

Block until the session has ended.

Return type

None

ensure_successful()

Block until the session has ended and raise an exception if the session has not finished successfully.

Return type

None

property session: *Session*

The session to watch.

property chosen_bits: `list[str]`

The Bits chosen by the caller to install.

property all_install_bits: `list[str] | None`

All Bits that are to be installed.

This data might be unavailable at the beginning until the dependency graph computation is finished.

property began_at: `datetime`

This session's start time.

property ended_at: `datetime | None`

This session's stop time (or None if currently running).

property was_successful: `bool`

Whether this session has finished successfully.

property _was_killed: `bool`

Whether this session was killed (e.g. due to system shutdown).


```

property installing_bits: list[str]
    The Bits that currently get applied.

property installed_bits: list[str]
    The Bits that were already applied during this run.

property progress: float
    The current progress, as value between 0 and 1.

__check_base_directory(*, init_phase=False)

    Parameters
        init_phase (bool)

__check_all_install_bits_directory(*, init_phase=False)

    Parameters
        init_phase (bool)

__check_alive(*, init_phase=False)

    Parameters
        init_phase (bool)

__check_log_directory()

__add_log_block(block_id, message, began_at, only_single_time, severity)

__log_block_changed(block_id, ended_at)

__set_bits_status(installing_bits, installed_bits, *, init_phase)

    Parameters
        init_phase (bool)

__set_all_install_bits_status(all_install_bits, *, init_phase)

    Parameters
        init_phase (bool)

__set_ended_status(failed_info, ended_at, *, init_phase)

    Parameters
        init_phase (bool)

__execute_handler(handler, *args)

__stop()

__refresh_bit_graph_image()

__refresh_bit_graph_image__process()

class __Monitor(path, on_changed_func)
    Bases: FilesystemMonitor

    Parameters
        • path (Path)
        • on_changed_func (Callable[[], None])

```

```
    _changed()

    _abc_impl = <_abc._abc_data object>

class __LogFile(path, add_log_block_func, log_block_changed_func)
    Bases: object

        Parameters
            path (Path)

        property is_finished

        update()

exception krrrez.flow.watch.RunFailedError
    Bases: RuntimeError

    Errors during the session runtime.

exception krrrez.flow.watch.InvalidStateError
    Bases: Exception

    Watch is in an invalid state.
```

krrez.flow.writer module

Engine session writer.

```
krrez.flow.writer._session_path_to_log_path(path)
```

```
    Parameters
        path (Path)

    Return type
        Path
```

```
class krrrez.flow.writer.Writer(*, session, log_block, chosen_bits, apply_message_func)
    Bases: object
```

Write state and logging data for a [krrez.flow.Session](#).

Each session is associated to an execution of a [krrez.flow.runner.Engine](#). The data that it writes is read by a [krrez.flow.watch.Watch](#).

The session is considered to be active within the activation period of this writer, i.e. in its with block.

```
    Parameters

        • session (krrez.flow.Session)

        • log_block (Writer.LogBlock | None)

        • chosen_bits (list[type[krrrez.api.Bit]])

        • apply_message_func (Callable[[str], str])
```

```
ISO_FORMAT_LENGTH = 26
```

```
PREFIX_PATTERN = 'prefix: {prefix}'
```

```
ENDED_AT_PATTERN = 'ended at {ended_at}: {block_id}'
```

```
BIT_GRAPH_FILE_NAME = 'bit_graph'
```

```

ALL_INSTALL_BITS_DIR_NAME = 'all_install_bits'

FAILED_FILE_NAME = 'failed'

ENDED_AT_FILE_NAME = 'ended_at'

class LogBlock(*, message, path, aux_name, severity, only_single_time=False, is_root, session)
    Bases: Logger
    Parameters
        • message (str)
        • path (str)
        • aux_name (str)
        • severity (Severity)
        • only_single_time (bool)
        • is_root (bool)
        • session (krrez.flow.Session)

class BlockLoggerMode(log_block)
    Bases: LoggerMode[ContextManager[Logger, bool | None]]
    _log(message, severity, aux_name)

    _abc_impl = <_abc._abc_data object>

class MessageLoggerMode(log_block)
    Bases: LoggerMode[None]
    _log(message, severity, aux_name)

    _abc_impl = <_abc._abc_data object>

property path: str
property aux_name: str
property session
property block
property message

_abc_impl = <_abc._abc_data object>

class _ApplyBitLogBlock(*args, bit_name, orig_session, **kwargs)
    Bases: LogBlock
    _abc_impl = <_abc._abc_data object>

property dialog_hub_path: Path
property dialog: Endpoint

```

set_install_bits(*install_bits*)

Set the Bits that are to be applied in this session.

Parameters

install_bits (*Iterable*[*type*[[Bit](#)]]) – The Bits.

Return type

None

refresh_bit_graph(*installing_bits*, *installed_bits*, *failed_bits*, *bit_graph*)

Refresh the Bit graph.

Parameters

- **installing_bits** (*list*[*type*[[Bit](#)]]) – The Bits that get applied at the moment.
- **installed_bits** (*list*[*type*[[Bit](#)]]) – The Bits that got applied.
- **failed_bits** (*list*[*type*[[Bit](#)]]) – The Bits that failed.
- **bit_graph** ([Node](#)) – The Bit graph.

Return type

None

finish(*, *success*)

Mark this run as finished (successfully or not).

Parameters

success (*bool*) – Whether it was successful.

Return type

None

__stop()

module_log(*bit_name*)

A new log block for a Bit.

Parameters

bit_name (*str*) – The Bit name.

Return type

[LogBlock](#)

property root_log: [LogBlock](#)

This session's root log block.

`krrez.flow.writer._ipc_dialog_hub_path`(*session_path*)

Parameters

session_path (*Path*)

Return type

Path

`krrez.flow.writer._ipc_dialog_hub_object_path`(*session_path*)

Parameters

session_path (*Path*)

Return type

Path

krrez.profiles namespace

Submodules

krrez.profiles.cloud module

class krrez.profiles.cloud.Profile(*, hostname, arch, krrrez_bits, config)

Bases: [Profile](#)

Parameters

- **hostname** (*str*)
- **arch** (*str*)
- **krrrez_bits** (*list[str]*)
- **config** (*dict[str, Any]*)

is_hidden = True

krrez.profiles.iot_box module

class krrez.profiles.iot_box.Profile(*, hostname, arch, krrrez_bits, config)

Bases: [Profile](#)

Parameters

- **hostname** (*str*)
- **arch** (*str*)
- **krrrez_bits** (*list[str]*)
- **config** (*dict[str, Any]*)

is_hidden = True

krrez.profiles.null module

class krrez.profiles.null.Profile(*, hostname)

Bases: [Profile](#)

Parameters

hostname (*str*)

is_hidden = True

krrez.profiles.workstation module

class krrez.profiles.workstation.Profile(*, hostname, arch, krrrez_bits, config)

Bases: [Profile](#)

Parameters

- **hostname** (*str*)
- **arch** (*str*)
- **krrrez_bits** (*list[str]*)
- **config** (*dict[str, Any]*)

```
is_hidden = True
```

krrez.seeding package

Krrcz seeding.

This is about mechanisms to apply bits to a target machine, usually by an installation medium that installs a fresh operation system and all selected bits, according to a chosen profile.

class krrez.seeding.SeedStrategy

Bases: ABC

Base class for seed strategies. Each seed strategy implements one particular way to install Krrcz to a target machine.

This includes the entire procedure, often starting with the creation of an installation medium, and ending with a fully installed Krrcz system on the target machine. See also the subclasses for a deeper understanding.

Seeding happens for a chosen `Profile`, which specifies what Bits should be installed and how things should be set up.

seed(*profile*, *, *target=None*, *log_block=None*)

Executes the seed strategy.

Note that a seed strategy can (indeed usually does) finish before the target machine is actually installed. It usually finishes with the creation of an installation medium, which you have to use to finish the installation on the target machine. See also [next_step_message](#).

Parameters

- **profile** ([Profile](#)) – The original profile to seed.
- **log_block** ([LogBlock](#) | *None*) – The engine scope to run in.
- **target** (*Path* | *None*) – The target device.

Return type

ContextManager[[Watch](#), bool | *None*]

abstract property next_step_message: str

Message that describes to the user how to proceed with the installation after [seed\(\)](#) finished.

abstractmethod _actual_profile(*profile*, *target*)

Returns a context manager that transforms the original given profile to an internal one. It is a more or less direct representation of the original one, depending on the seed strategy, but could be modified in some ways that carry the actual implementation for this seed strategy.

Parameters

- **profile** ([Profile](#)) – The original profile to seed.
- **target** (*Path*) – The target device.

Return type

ContextManager[[Profile](#), bool | *None*]

```
_abc_impl = <_abc._abc_data object>
```

class krrez.seeding._IndirectSeedStrategy

Bases: [SeedStrategy](#)

Abstract implementation for indirect seed strategies.

Those strategies internally consist of two seeds. At first, an installer is seeded to a particular disk. The next step is to boot the system from this disk. This will start another seeding to the actual target. The lifetime of the installer system ends once the entire installation is done.

The 1st stage could be seeded to an internal or an external disk, depending on the actual strategy.

abstractmethod `_stage1_profile(profile, target)`

Returns a context manager that prepares and provides the 1st stage profile.

Parameters

- **profile** ([Profile](#)) – The original profile to seed.
- **target** ([Path](#)) – The target device.

Return type

[ContextManager](#)[[Profile](#), bool | None]

static `_retry()`

_actual_profile(profile, target)

Returns a context manager that transforms the original given profile to an internal one. It is a more or less direct representation of the original one, depending on the seed strategy, but could be modified in some ways that carry the actual implementation for this seed strategy.

Parameters

- **profile** – The original profile to seed.
- **target** – The target device.

static `_first_boot_action(profile)`

`_abc_impl = <_abc._abc_data object>`

class `krrez.seeding._SeedThisMachineDirectly`

Bases: [SeedStrategy](#)

Seeds Krrrez on this machine in a direct way.

This is a non-abstract seed strategy, but there is barely a reason to use it directly. It is used internally.

next_step_message = ''

_actual_profile(profile, target)

Returns a context manager that transforms the original given profile to an internal one. It is a more or less direct representation of the original one, depending on the seed strategy, but could be modified in some ways that carry the actual implementation for this seed strategy.

Parameters

- **profile** – The original profile to seed.
- **target** – The target device.

`_abc_impl = <_abc._abc_data object>`

class `krrez.seeding.SeedRemovableSystemDiskDirectly`

Bases: [SeedStrategy](#)

Seeds Krrrez by creating a new system disk (SD card for IoT device, ...). You can insert it into the target machine and leave it there. It will execute the actual installation on first boot.

```
next_step_message = 'The next step is to take your new installation medium, insert it into the target machine, and boot from it. This medium will turn into the system medium during installation, so leave it attached afterwards.'
```

```
_actual_profile(profile, target)
```

Returns a context manager that transforms the original given profile to an internal one. It is a more or less direct representation of the original one, depending on the seed strategy, but could be modified in some ways that carry the actual implementation for this seed strategy.

Parameters

- **profile** – The original profile to seed.
- **target** – The target device.

```
_abc_impl = <_abc._abc_data object>
```

```
class krrcz.seeding.SeedIndirectlyViaRemovable
```

Bases: [`_IndirectSeedStrategy`](#)

Seeds Krrcz via a removable boot medium (usb stick, ...). It will be an indirect procedure: On your seed machine, an installation medium will be created. You can then insert it into the target machine and boot it from that medium. This will execute the actual installation there. Unplug/eject the installation medium afterward.

```
next_step_message = 'The next step is to take your new installation medium, insert it into the target machine, and boot from it. The installation procedure will void the installation medium, in order to avoid the danger of accidentally boot again from it. Remove it after the installation is finished.'
```

```
_stage1_profile(profile, target)
```

Returns a context manager that prepares and provides the 1st stage profile.

Parameters

- **profile** – The original profile to seed.
- **target** – The target device.

```
_abc_impl = <_abc._abc_data object>
```

```
class krrcz.seeding.ReseedThisMachine(*, profile_type, profile_data, kept_config, profile_patchers=())
```

Bases: [`_IndirectSeedStrategy`](#)

Reseeds Krrcz on this machine in an indirect way (only supported in some circumstances).

This strategy is somewhat special AND POTENTIALLY DANGEROUS!

You can use it on machines that are already seeded Krrcz machines. It will install this machine (i.e. your local one!) again with the same profile as before.

This can be useful after a backup, in order to reinstall your system (e.g. to a higher Krrcz version).

Parameters

profile_patchers (`Iterable[Callable[[krrcz.api.Profile], None]]`)

```
next_step_message = ''
```

```
_actual_profile(profile, target)
```

Returns a context manager that transforms the original given profile to an internal one. It is a more or less direct representation of the original one, depending on the seed strategy, but could be modified in some ways that carry the actual implementation for this seed strategy.

Parameters

- **profile** – The original profile to seed.
- **target** – The target device.

_stage1_profile(*profile, target*)

Returns a context manager that prepares and provides the 1st stage profile.

Parameters

- **profile** ([Profile](#)) – The original profile to seed.
- **target** – The target device.

class _Stage1ProfileGenerator(*profile*)

Bases: [object](#)

Parameters

profile ([Profile](#))

generate()

Return type

[Profile](#)

class _PartitionInfo(*stage1_disk_setup: hallyd.disk.DiskSetup, stage1_partition_setup: hallyd.disk.PartitionSetup, final_disk_setup: hallyd.disk.DiskSetup, final_partition_setup: hallyd.disk.PartitionSetup, disk_setup_index: int, partition_setup_index: int, host_disk: hallyd.disk.Disk, host_partition: hallyd.disk.Partition*)

Bases: [object](#)

Parameters

- **stage1_disk_setup** ([DiskSetup](#))
- **stage1_partition_setup** ([PartitionSetup](#))
- **final_disk_setup** ([DiskSetup](#))
- **final_partition_setup** ([PartitionSetup](#))
- **disk_setup_index** ([int](#))
- **partition_setup_index** ([int](#))
- **host_disk** ([Disk](#))
- **host_partition** ([Partition](#))

stage1_disk_setup: [DiskSetup](#)

stage1_partition_setup: [PartitionSetup](#)

final_disk_setup: [DiskSetup](#)

final_partition_setup: [PartitionSetup](#)

disk_setup_index: [int](#)

partition_setup_index: [int](#)

host_disk: [Disk](#)

host_partition: [Partition](#)

_Stage1ProfileGenerator__find_efi_partitions()

Return type

[Iterable](#)[[_PartitionInfo](#)]

```
_Stage1ProfileGenerator__find_partitions(filter_func)
    Parameters
        filter_func (Callable[[DiskSetup, PartitionSetup], bool])
    Return type
        Generator[_PartitionInfo, None, None]

_Stage1ProfileGenerator__find_working_partition()
    Return type
        _PartitionInfo

_Stage1ProfileGenerator__partition_by_indexes(disk_setup_index, partition_setup_index)
    Parameters
        • disk_setup_index (int)
        • partition_setup_index (int)
    Return type
        _PartitionInfo

static _Stage1ProfileGenerator__patch_efi_partitions(efi_partitions)
    Parameters
        efi_partitions (Iterable[_PartitionInfo])
    Return type
        None

static _Stage1ProfileGenerator__patch_final_profile(profile)
    Parameters
        profile (Profile)
    Return type
        None

static _Stage1ProfileGenerator__patch_stage1_profile(stage1_profile)
    Parameters
        stage1_profile (Profile)
    Return type
        None

static _Stage1ProfileGenerator__patch_working_partition(working_partition)
    Parameters
        working_partition (_PartitionInfo)
    Return type
        None

_abc_impl = <_abc._abc_data object>

class krrrez.seeding.SeedAct(*, profile, target_device)
    Bases: ABC
    Parameters
        • profile (Profile)
        • target_device (Path)
    property _target_device
    abstractmethod _begin()
```

`abstractmethod _start_creating_installation_medium(watch)`

Parameters

`watch` ([Watch](#))

`abstractmethod _installation_medium_created(seed_user, seed_strategy)`

`abstractmethod _start_finishing()`

`abstractmethod _watch_finishing(watch)`

Parameters

`watch` ([Watch](#))

`abstractmethod _done(*, finish_from_here, unknown)`

Parameters

- `finish_from_here` (*bool*)
- `unknown` (*bool*)

`seed()`

Return type

None

`_abc_impl = <_abc._abc_data object>`

`class krrrez.seeding._RemoteContext(connection, root_path)`

Bases: [Context](#)

Parameters

- `connection` ([Connection](#))
- `root_path` ([Path](#))

`_interaction_request_fetcher_for_session(session, provider)`

Subpackages

[krrrez.seeding.reseed package](#)

`krrrez.seeding.reseed.run(*, backup_connection_name=None, _for_zz_test=None, _skip_updating=False)`

Reseed the current machine with an updated version of Krrrez and its auxiliary packages.

Parameters

- `backup_connection_name` (*str | bool | None*) – The backup connection name for the final backup.
- `_for_zz_test` (*dict | None*) – Do not use.
- `_skip_updating` (*bool*) – Do not use.

Return type

None

Submodules

krrez.seeding.reseed.__main__ module

Submodules

krrez.seeding.api module

Main programming interface for the implementation of seeding bits.

class `krrez.seeding.api.Bit`

Bases: `Bit`

Do not construct Bits directly.

bit_names_for_stage(*stage*)

Parameters

stage (`Stage`)

Return type

list[str]

`_krrez_do_not_derive_a_dependency = True`

class `krrez.seeding.api.Stage(*values)`

Bases: Enum

`PREPARE = 'in_host_prepare'`

`PREPARE_RAW = 'in_host_prepare_raw'`

`BUILD_RAW = 'in_host_build_raw'`

`PREPARE_SYSTEM = 'in_host_prepare_system'`

`BUILD_SYSTEM = 'in_host_build_system'`

`PREPARE_CHROOT = 'in_host_prepare_chroot'`

`CHROOT = 'in_host_chroot'`

`IN_TARGET = 'in_target'`

`IN_TARGET_LATE = 'in_target_late'`

`ON_EXIT = 'in_host_on_exit'`

class `krrez.seeding.api.ConfigValue(*, default=None, type=<class 'object'>)`

Bases: `BareConfigValue[_TConfigValueType]`, `Generic[_TConfigValueType]`

Parameters

type (`type[_TConfigValueType]`)

`_abc_impl = <_abc._abc_data object>`

`krrez.seeding.api._key_name_for_stage(stage)`

Parameters

stage (`Stage`)

Return type

str

krrez.seeding.profile_loader module

Finding and loading profiles.

`krrez.seeding.profile_loader.profile_module_path(profile)`

The path of the Python module that contains this profile.

Parameters

profile (`Profile` / `type[Profile]`)

Return type

Path

`krrez.seeding.profile_loader.profile_name(profile)`

The (short) name of a profile.

See also [`profile_full_name\(\)`](#).

Parameters

profile (`str` / `Profile` / `type[Profile]`) – The profile. Can be a short or long name, a type or an instance.

Return type

`str`

`krrez.seeding.profile_loader.profile_full_name(profile)`

The full name of a profile (equivalent to the type's full name incl. module name).

See also [`profile_name\(\)`](#).

Parameters

profile (`str` / `Profile` / `type[Profile]`) – The profile. Can be a short or long name, a type or an instance.

Return type

`str`

`krrez.seeding.profile_loader.all_profiles()`

All profiles.

This also includes internal hidden profiles. See also [`browsable_profiles\(\)`](#).

Return type

`list[type[Profile]]`

`krrez.seeding.profile_loader.browsable_profiles()`

All browsable profiles.

See also [`all_profiles\(\)`](#).

Return type

`list[type[Profile]]`

`krrez.seeding.profile_loader._modules(modname='krrez.profiles')`

Return type

`list[Any]`

krrez.seeding.system module

System level helpers.

`krrez.seeding.system.turn_into_krrrez_system(system_root_path, *, install_krrrez_packages=True)`

Turn a target system into a Krrrez system.

This includes copying all Krrrez modules to a particular location in the target system, and some additional wiring in order to make the 'krrrez' command available and enabled (i.e. the target system is marked as Krrrez machine).

Parameters

- **system_root_path** (*Path*) – The root directory of the target system to turn into a Krrrez system.
- **install_krrrez_packages** (*bool*) – If to include copying Krrrez packages. If not, you can manually do so by picking up /__.

Return type

None

krrez.testing package

Krrrez testing.

`krrez.testing.start_tests(bit_names, *, context=None)`

Start a test run.

Parameters

- **bit_names** (*Iterable[str]*) – The test Bit names to apply. Those Bits are usually test plans. See [all_available_test_plans\(\)](#). There is usually just one.
- **context** (*Context* / *None*) – The context. Note: The actual test run will happen in a sub-context of it.

Return type

[Watch](#)

`krrez.testing.all_available_test_plans()`

The available test plans.

You usually call [all_available_test_plans\(\)](#) with one of them.

Return type

`list[str]`

`krrez.testing.all_test_sessions(context=None)`

All test sessions from the past.

Parameters

context (*Context* / *None*) – The context.

Return type

`list[Session]`

`krrez.testing.test_plan_for_test_session(session)`

Parameters

session (*Session*)

Return type

`str`

`krrez.testing.clean_up_old_test_sessions(context=None)`

Remove old test sessions.

Parameters

context (`Context` / `None`) – The context.

Return type

`None`

`class krrez.testing._TestingEngine`

Bases: `Engine`

`_worker_pool_size = 20`

`_apply_message(bit_name)`

`_do_finish(session, runner_writer, success)`

`krrez.testing._testing_context(context)`

Parameters

context (`Context` / `None`)

Return type

`Context`

Submodules

krrez.testing.api module

Main programming interface for the implementation of testing bits.

`class krrez.testing.api._InitDep`

Bases: `Dependency`

`manipulate_resolution_plan(owning_bit, plan)`

Apply custom manipulations to a resolution plan. Return True if a change was made.

This is only used for very particular internal tricks.

Parameters

- **owning_bit** – The Bit that this dependency is associated to.
- **plan** – The resolution plan.

`class krrez.testing.api.BundledTestBit`

Bases: `Bit`

Base class for simple implementation of test bundle oriented tests.

Use it this way (all in your test module): - Implement a subclass of it, named “*Bit*”. It usually contains only the `._prepare()` and `._test()` methods. - For each test bundle that your test is relevant for, implement a subclass of your *Bit*, named e.g.

“*InDonkeyTestBundle1Bit*”. In its `krrez.api.Bit.__apply__()` methods, call `self._prepare()` and `self._test()` how it makes sense.

Do not construct Bits directly.

_run: `Bit`

`__init_dep:` <krrcz.testing.api._InitDep object at 0x7f7973021220>

`class krrcz.testing.api.LandmarkBit`

Bases: *Bit*

Base class for landmarks.

Do not construct Bits directly.

`_run:` *Bit*

`_after_reboot:` *Bit*

`class krrcz.testing.api._PartOf(bit_name)`

Bases: *BaseForSimpleBehaviorDependency*

Parameters

- **bits** – The Bits to depend on.
- **afterwards** – Whether this is an afterwards-dependency.
- **bit_name** (*str*)

`property bit_name:` *str*

`manipulate_resolution_plan(owning_bit, plan)`

Apply custom manipulations to a resolution plan. Return True if a change was made.

This is only used for very particular internal tricks.

Parameters

- **owning_bit** – The Bit that this dependency is associated to.
- **plan** – The resolution plan.

`_PartOf__late_fake_bit(owning_bit, part_of_bit, plan)`

`class krrcz.testing.api._InstallBeforehandForTestBundle(bit_name, profile_name, testbundle_name)`

Bases: *SimpleDependency*

Parameters

- **bits** – The Bits to depend on.
- **afterwards** – Whether this is an afterwards-dependency.
- **bit_name** (*str*)
- **profile_name** (*str*)
- **testbundle_name** (*str*)

`property testbundle_name`

`property profile_name`

`additional_needed_bits(cooling_down, plan)`

A list of Bits that are pulled in by this dependency.

Parameters

- **cooling_down** – If the process is in cooling down mode.
- **plan** – The resolution plan.

krrez.testing.landmark module

Platform level support for landmarks in testing.

`krrez.testing.landmark.set_landmark(run, after_reboot, name)`

Set a landmark.

`start_resume_tests_from_landmark()` can resume a terminated session from the past from the current situation. Note that all machines will shutdown during that time, and that no other Bits may execute in parallel.

Parameters

- **run** (*Bit*) – The run Bit.
- **after_reboot** (*Bit*) – The after_reboot Bit.
- **name** (*str*) – The landmark name.

Return type

None

`krrez.testing.landmark.forget_landmark(session)`

Remove landmark information for a session.

Parameters

session (*Session*) – The session to clean.

Return type

None

`krrez.testing.landmark.clean_up_old_landmarks(context=None)`

Remove old landmarks.

Parameters

context (*Context* / *None*) – The context.

`krrez.testing.landmark.has_resumable_landmark(session)`

Return whether a session can be resumed from a landmark.

Parameters

session (*Session*) – The session to check.

Return type

bool

`krrez.testing.landmark.landmark_size_on_disk(session)`

Return the disk usage of landmark data for a session (in bytes).

Parameters

session (*Session*) – The session to check.

Return type

int

`krrez.testing.landmark.landmark_name_for_session(session)`

Return the landmark name for a session (or None if there is no landmark data for this session).

Parameters

session (*Session*) – The session to check.

Return type

str | None

`krrez.testing.landmark.start_resume_tests_from_landmark(session)`

Start a test run that resumes a session from the past from its last landmark.

Parameters

session ([Session](#)) – The session to resume.

Return type

[Watch](#)

`class krrez.testing.landmark._LandmarkResumingTestingEngine(landmark_name, run_dir, xml_domains)`

Bases: [_TestingEngine](#)

`_do_prepare(session, install_bits)`

`_do_install_bit(bit, log_block, session, runner_writer)`

`krrez.testing.landmark._base_landmark_path(session)`

Parameters

session ([Session](#))

Return type

[Path](#)

`krrez.testing.landmark._last_landmark_path(session)`

Parameters

session ([Session](#))

Return type

[Path](#)

`krrez.testing.landmark._last_data_landmark_path(session)`

Parameters

session ([Session](#))

Return type

[Path](#)

`class krrez.testing.landmark._Boot(machine, after_reboot)`

Bases: `object`

Parameters

- **machine** (`krrez.bits.zz_test.zz_run.Machine`)
- **after_reboot** (`krrez.bits.zz_test.zz_after_reboot.Bit`)

krrez.ui package

Subpackages

krrez.ui.apps package

Subpackages

krrez.ui.apps.dev_lab package

Subpackages

krrez.ui.apps.dev_lab.models package

Submodules

krrez.ui.apps.dev_lab.models.bit_editor module

krrez.ui.apps.dev_lab.models.main module

krrez.ui.apps.dev_lab.models.profile_editor module

krrez.ui.apps.dev_lab.models.test_editor module

krrez.ui.apps.dev_lab.models.test_plan_editor module

krrez.ui.apps.dev_lab.views package

Submodules

krrez.ui.apps.dev_lab.views.bit_editor module

krrez.ui.apps.dev_lab.views.main module

krrez.ui.apps.dev_lab.views.profile_editor module

krrez.ui.apps.dev_lab.views.test_editor module

krrez.ui.apps.dev_lab.views.test_plan_editor module

krrez.ui.apps.log_browser package

Subpackages

krrez.ui.apps.log_browser.models package

Submodules

krrez.ui.apps.log_browser.models.main module

krrez.ui.apps.log_browser.views package

Submodules

krrez.ui.apps.log_browser.views.main module

krrez.ui.apps.runner package

Subpackages

krrez.ui.apps.runner.models package

Submodules

krrez.ui.apps.runner.models.main module

krrez.ui.apps.runner.views package

Submodules

krrez.ui.apps.runner.views.main module

krrez.ui.apps.seeding package

Subpackages

krrez.ui.apps.seeding.models package

Submodules

krrez.ui.apps.seeding.models.finishing module

krrez.ui.apps.seeding.models.main module

krrez.ui.apps.seeding.models.new_seeding module

krrez.ui.apps.seeding.models.session module

krrez.ui.apps.seeding.views package

Submodules

krrez.ui.apps.seeding.views.finishing module

krrez.ui.apps.seeding.views.main module

krrez.ui.apps.seeding.views.new_seeding module

krrez.ui.apps.seeding.views.session module

krrez.ui.apps.studio package

Subpackages

krrez.ui.apps.studio.models package

Submodules

krrez.ui.apps.studio.models.main module

krrez.ui.apps.studio.models.welcome module

krrez.ui.apps.studio.views package

Submodules

krrez.ui.apps.studio.views.help module

krrez.ui.apps.studio.views.main module

krrez.ui.apps.studio.views.welcome module

krrez.ui.apps.testing package

Subpackages

krrez.ui.apps.testing.models package

Submodules

`krrez.ui.apps.testing.models.main` module

`krrez.ui.apps.testing.models.new_test_run` module

`krrez.ui.apps.testing.views` package

Submodules

`krrez.ui.apps.testing.views.main` module

`krrez.ui.apps.testing.views.new_test_run` module

`krrez.ui.models` package

Submodules

`krrez.ui.models.list_panel` module

`krrez.ui.models.property_panel` module

`krrez.ui.models.session` module

`krrez.ui.models.session_interaction` module

`krrez.ui.views` package

Submodules

`krrez.ui.views.list_panel` module

`krrez.ui.views.property_panel` module

`krrez.ui.views.session` module

`krrez.ui.views.session_interaction` module

`krrez.ui.views.window` module

6.1.2 Submodules

6.1.3 `krrez.krrrez_cli` module

`krrez.krrrez_cli.get_parser`(*only_relevant=False*)

Parameters

`only_relevant` (*bool*)

Return type

ArgumentParser

class `krrez.krrrez_cli._Command`(*, *context_path*, **_)

Bases: `ABC`

property `context`

`ui_app`(*app_name*, *unavailable_message=None*, ***kwargs*)

Parameters

```
        • app_name (str)
        • unavailable_message (str | None)

main()

_abc_impl = <_abc._abc_data object>

krrrez.krrrez_cli._list_logs(sessions)

krrrez.krrrez_cli._apply_bits(bits, *, context_path=None, confirm_after_installation=True, engine=None)

krrrez.krrrez_cli._dump_session(session_name, verbose, context)

class krrrez.krrrez_cli.Commands
    Bases: object

    exception AppUnavailableError
        Bases: RuntimeError

    class Command_bits(*, context_path, **_)
        Bases: _Command

        main()

        sub_list(**_)

        sub_status(*, bits, **_)
            Parameters
                bits (list[str])

        sub_apply(*, bits, no_confirm_after_installation, **_)
            Parameters
                • bits (list[str])
                • no_confirm_after_installation (bool)

        _abc_impl = <_abc._abc_data object>

    class Command_logs(*, context_path, **_)
        Bases: _Command

        main()

        sub_list(**_)

        sub_show(*, session, verbose, **_)
            Parameters
                • session (str)
                • verbose (bool)

        _abc_impl = <_abc._abc_data object>

    class Command_seeding(*, context_path, **_)
        Bases: _Command

        class Command_profiles(*, context_path, **_)
            Bases: _Command

            sub_list(**_)
```

```

    sub_info(profile, **_)

    _abc_impl = <_abc._abc_data object>

main()

sub_sow(*, profile, arguments, target_device, **_)
    Parameters
        • profile (str)
        • arguments (str)
        • target_device (str)

    _abc_impl = <_abc._abc_data object>

class Command_dev_lab(*, context_path, **_)
    Bases: _Command
    main()

    _abc_impl = <_abc._abc_data object>

class Command_testing(*, context_path, **_)
    Bases: _Command

class Command_plans(*, context_path, **_)
    Bases: _Command
    sub_list(**_)

    _abc_impl = <_abc._abc_data object>

class Command_logs(*, context_path, **_)
    Bases: _Command
    sub_list(**_)

    sub_show(*, session, verbose, **_)
        Parameters
            • session (str)
            • verbose (bool)

    _abc_impl = <_abc._abc_data object>

class Command_landmarks(*, context_path, **_)
    Bases: _Command
    sub_list(**_)

    sub_resume(*, session, **_)
        Parameters
            session (str)

    sub_remove(*, session, **_)
        Parameters
            session (str)

    _abc_impl = <_abc._abc_data object>

main()

```

```
sub_run(*, plan, **_)
    Parameters
    plan(str)

_abc_impl = <_abc._abc_data object>

class Command_studio(*, context_path, **_)
    Bases: _Command
    main()

    _abc_impl = <_abc._abc_data object>

krrrez.krrrez_cli.main()
```


PYTHON MODULE INDEX

k

- `krrez`, 19
- `krrez.api`, 19
- `krrez.api.internal`, 22
- `krrez.asset`, 30
- `krrez.asset.data`, 30
- `krrez.asset.deploy_info`, 30
- `krrez.asset.project_info`, 30
- `krrez.coding`, 30
- `krrez.flow`, 37
- `krrez.flow.bit_loader`, 46
- `krrez.flow.config`, 49
- `krrez.flow.dialog`, 52
- `krrez.flow.graph`, 41
- `krrez.flow.graph.resolver`, 42
- `krrez.flow.graph.visualizer`, 44
- `krrez.flow.logging`, 55
- `krrez.flow.runner`, 56
- `krrez.flow.watch`, 60
- `krrez.flow.writer`, 62
- `krrez.krrez_cli`, 81
- `krrez.profiles`, 65
- `krrez.profiles.cloud`, 65
- `krrez.profiles.iot_box`, 65
- `krrez.profiles.null`, 65
- `krrez.profiles.workstation`, 65
- `krrez.seeding`, 66
- `krrez.seeding.api`, 72
- `krrez.seeding.profile_loader`, 73
- `krrez.seeding.reseed`, 71
- `krrez.seeding.system`, 74
- `krrez.testing`, 74
- `krrez.testing.api`, 75
- `krrez.testing.landmark`, 77

INDEX

Symbols

- `_ApplyTask` (class in `krrez.flow.runner`), 58
- `_ApplyTask.PipeReaderThread` (class in `krrez.flow.runner`), 59
- `_Boot` (class in `krrez.testing.landmark`), 78
- `_Command` (class in `krrez.krrez_cli`), 81
- `_ConfigValueWithoutAskFor` (class in `krrez.api.internal`), 27
- `_Controller` (class in `krrez.flow.config`), 50
- `_DialogRequest` (class in `krrez.flow.dialog`), 54
- `_IS_KRREZ_MACHINE_FLAG_PATH` (in module `krrez.flow`), 37
- `_IndirectSeedStrategy` (class in `krrez.seeding`), 66
- `_InitDep` (class in `krrez.testing.api`), 75
- `_InstallBeforehandForTestBundle` (class in `krrez.testing.api`), 76
- `_InteractionRequestFetcher` (class in `krrez.flow.dialog`), 54
- `_InteractiveController` (class in `krrez.flow.config`), 50
- `_InteractiveController._AskFor` (class in `krrez.flow.config`), 51
- `_InteractiveController._AskFor._None` (class in `krrez.flow.config`), 51
- `_KRREZ_BITS_PATTERN` (`krrez.coding.Profiles._Editor` attribute), 32
- `_LandmarkResumingTestingEngine` (class in `krrez.testing.landmark`), 78
- `_NoConfigValueDefault` (`krrez.api.internal.BareConfigValue` attribute), 26
- `_PartOf` (class in `krrez.testing.api`), 76
- `_PartOf__late_fake_bit()` (`krrez.testing.api._PartOf` method), 76
- `_Plan` (class in `krrez.flow.graph.resolver`), 43
- `_RemoteContext` (class in `krrez.seeding`), 71
- `_SeedThisMachineDirectly` (class in `krrez.seeding`), 67
- `_Session` (class in `krrez.flow`), 40
- `_Stage1ProfileGenerator__find_efi_partitions()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` method), 69
- `_Stage1ProfileGenerator__find_working_partition()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` method), 70
- `_Stage1ProfileGenerator__partition_by_indexes()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` method), 70
- `_Stage1ProfileGenerator__patch_efi_partitions()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` static method), 70
- `_Stage1ProfileGenerator__patch_final_profile()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` static method), 70
- `_Stage1ProfileGenerator__patch_stage1_profile()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` static method), 70
- `_Stage1ProfileGenerator__patch_working_partition()` (`krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator` static method), 70
- `_TestingEngine` (class in `krrez.testing`), 75
- `_WorkerLoop__finish_worker()` (`krrez.flow.runner.Engine._WorkerLoop` method), 58
- `_WorkerLoop__installable_bits()` (`krrez.flow.runner.Engine._WorkerLoop` method), 58
- `_WorkerLoop__refresh_bit_graph()` (`krrez.flow.runner.Engine._WorkerLoop` method), 58
- `_WorkerLoop__start_worker()` (`krrez.flow.runner.Engine._WorkerLoop` method), 58
- `__add_log_block()` (`krrez.flow.watch.Watch` method), 61
- `__all_descendants__helper()` (`krrez.flow.graph.Node` method), 42
- `__check_alive()` (`krrez.flow.watch.Watch` method), 61
- `__check_all_install_bits_directory()` (`krrez.flow.watch.Watch` method), 61
- `__check_base_directory()` (`krrez.flow.watch.Watch` method), 61

```

        method), 61
__check_log_directory() (krrrez.flow.watch.Watch
    method), 61
__compute_bit_graph() (krrrez.flow.runner.Engine
    method), 57
__context_path_to_magic_file_path() (krrrez.flow.Context method), 38
__ctrl() (krrrez.api.internal.BareConfigValue method),
    27
__dependency_bits() (krrrez.api.internal.BaseForSimpleBehaviorDependency
    method), 24
__engine_lock() (krrrez.flow.runner.Engine method),
    57
__execute_handler() (krrrez.flow.watch.Watch
    method), 61
__get_block_dev_info() (krrrez.api.internal.ProfileMeta method), 23
__init_dep (krrrez.testing.api.BundledTestBit attribute),
    75
__log_block_changed() (krrrez.flow.watch.Watch
    method), 61
__refresh_bit_graph_image() (krrrez.flow.watch.Watch method), 61
__refresh_bit_graph_image__process() (krrrez.flow.watch.Watch method), 61
__remove_node() (krrrez.flow.graph.Node method), 42
__resolve_optional() (krrrez.api.internal.BaseForSimpleBehaviorDependency
    method), 24
__set_all_install_bits_status() (krrrez.flow.watch.Watch method), 61
__set_bits_status() (krrrez.flow.watch.Watch
    method), 61
__set_ended_status() (krrrez.flow.watch.Watch
    method), 61
__stop() (krrrez.flow.watch.Watch method), 61
__stop() (krrrez.flow.writer.Writer method), 64
__trim_value() (krrrez.flow.graph.visualizer.Color
    method), 46
__try_dump_pygraphviz__add_pygraphviz_node()
    (in module krrrez.flow.graph.visualizer), 45
__try_dump_pygraphviz__dive() (in module krrrez.flow.graph.visualizer), 45
__value (krrrez.api.internal.BareConfigValue property),
    27
_abc_impl (krrrez.api.internal.AdHocLogger attribute),
    26
_abc_impl (krrrez.api.internal.AdHocLogger._BlockMode
    attribute), 26
_abc_impl (krrrez.api.internal.AdHocLogger._MessageMode
    attribute), 26
_abc_impl (krrrez.api.internal.BareConfigValue at-
    tribute), 27
_abc_impl (krrrez.api.internal.ConfigValue attribute), 29
_abc_impl (krrrez.api.internal.ConfigValueAskForPrepared
    attribute), 28
_abc_impl (krrrez.api.internal.ConnectableToBit at-
    tribute), 25
_abc_impl (krrrez.api.internal.Lock attribute), 25
_abc_impl (krrrez.api.internal._ConfigValueWithoutAskFor
    attribute), 28
_abc_impl (krrrez.flow.Session attribute), 40
_abc_impl (krrrez.flow._Session attribute), 40
_abc_impl (krrrez.flow.config.Controller attribute), 50
_abc_impl (krrrez.flow.config._Controller attribute), 50
_abc_impl (krrrez.flow.config._InteractiveController at-
    tribute), 51
_abc_impl (krrrez.flow.dialog.Endpoint attribute), 54
_abc_impl (krrrez.flow.dialog.Processor attribute), 54
_abc_impl (krrrez.flow.dialog.Provider attribute), 54
_abc_impl (krrrez.flow.graph.resolver.Plan attribute), 43
_abc_impl (krrrez.flow.graph.resolver._Plan attribute),
    44
_abc_impl (krrrez.flow.logging.Logger attribute), 56
_abc_impl (krrrez.flow.logging.LoggerMode attribute),
    56
_abc_impl (krrrez.flow.watch.Watch.__Monitor at-
    tribute), 62
_abc_impl (krrrez.flow.writer.Writer.LogBlock attribute),
    63
_abc_impl (krrrez.flow.writer.Writer.LogBlock.BlockLoggerMode
    attribute), 63
_abc_impl (krrrez.flow.writer.Writer.LogBlock.MessageLoggerMode
    attribute), 63
_abc_impl (krrrez.flow.writer.Writer._ApplyBitLogBlock
    attribute), 63
_abc_impl (krrrez.krrrez_cli.Commands.Command_bits
    attribute), 82
_abc_impl (krrrez.krrrez_cli.Commands.Command_dev_lab
    attribute), 83
_abc_impl (krrrez.krrrez_cli.Commands.Command_logs
    attribute), 82
_abc_impl (krrrez.krrrez_cli.Commands.Command_seeding
    attribute), 83
_abc_impl (krrrez.krrrez_cli.Commands.Command_seeding.Command_prof
    attribute), 83
_abc_impl (krrrez.krrrez_cli.Commands.Command_studio
    attribute), 84
_abc_impl (krrrez.krrrez_cli.Commands.Command_testing
    attribute), 84
_abc_impl (krrrez.krrrez_cli.Commands.Command_testing.Command_land
    attribute), 83
_abc_impl (krrrez.krrrez_cli.Commands.Command_testing.Command_logs
    attribute), 83
_abc_impl (krrrez.krrrez_cli.Commands.Command_testing.Command_plans
    attribute), 83
_abc_impl (krrrez.krrrez_cli._Command attribute), 82

```

`_abc_impl` (`krrcz.seeding.ReseedThisMachine` attribute), 70
`_abc_impl` (`krrcz.seeding.SeedAct` attribute), 71
`_abc_impl` (`krrcz.seeding.SeedIndirectlyViaRemovable` attribute), 68
`_abc_impl` (`krrcz.seeding.SeedRemovableSystemDiskDirectly` attribute), 68
`_abc_impl` (`krrcz.seeding.SeedStrategy` attribute), 66
`_abc_impl` (`krrcz.seeding._IndirectSeedStrategy` attribute), 67
`_abc_impl` (`krrcz.seeding._SeedThisMachineDirectly` attribute), 67
`_abc_impl` (`krrcz.seeding.api.ConfigValue` attribute), 72
`_actual_profile()` (`krrcz.seeding.ReseedThisMachine` method), 68
`_actual_profile()` (`krrcz.seeding.SeedRemovableSystemDiskDirectly` method), 68
`_actual_profile()` (`krrcz.seeding.SeedStrategy` method), 66
`_actual_profile()` (`krrcz.seeding._IndirectSeedStrategy` method), 67
`_actual_profile()` (`krrcz.seeding._SeedThisMachineDirectly` method), 67
`_after_reboot` (`krrcz.testing.api.LandmarkBit` attribute), 76
`_all_bit_usage_declarations()` (`krrcz.api.internal.MayDeclareSecondaryBitUsages` class method), 22
`_all_bits()` (in module `krrcz.flow.bit_loader`), 48
`_all_bits_helper()` (in module `krrcz.flow.bit_loader`), 48
`_all_bits_helper_deep()` (in module `krrcz.flow.bit_loader`), 48
`_all_sessions_path()` (`krrcz.flow.Session` static method), 40
`_apply_bits()` (in module `krrcz.krrcz_cli`), 82
`_apply_message()` (`krrcz.flow.runner.Engine` method), 57
`_apply_message()` (`krrcz.testing._TestingEngine` method), 75
`_ask_for()` (`krrcz.api.internal.ConfigValueAskForPrepared` method), 28
`_base_landmark_path()` (in module `krrcz.testing.landmark`), 78
`_begin()` (`krrcz.seeding.SeedAct` method), 70
`_bit_for_type()` (`krrcz.api.Bit` method), 20
`_bit_for_type()` (`krrcz.api.internal.MayDeclareSecondaryBitUsages` method), 22
`_bit_name_to_specific_name()` (in module `krrcz.coding`), 36
`_bits` (`krrcz.api.internal.BaseForSimpleBehaviorDependency` property), 24
`_blank_contexts_path()` (`krrcz.flow.Context` method), 38
`_border_color()` (in module `krrcz.flow.graph.visualizer`), 45
`_changed()` (`krrcz.flow.watch.Watch.__Monitor` method), 61
`_cleanup_dependencies_from_nodes()` (in module `krrcz.flow.graph.resolver`), 44
`_config_dir_path_for_context_path()` (in module `krrcz.flow.config`), 49
`_connected_to_bit()` (`krrcz.api.internal.BareConfigValue` method), 27
`_connected_to_bit()` (`krrcz.api.internal.ConnectableToBit` method), 25
`_connected_to_bit()` (`krrcz.api.internal.Lock` method), 25
`_data_dir` (`krrcz.api.Bit` property), 20
`_do_finish()` (`krrcz.flow.runner.Engine` method), 57
`_do_finish()` (`krrcz.testing._TestingEngine` method), 75
`_do_install_bit()` (`krrcz.flow.runner.Engine` method), 58
`_do_install_bit()` (`krrcz.testing.landmark._LandmarkResumingTestingEngine` method), 78
`_do_install_bit_helper()` (`krrcz.flow.runner.Engine` static method), 58
`_do_prepare()` (`krrcz.flow.runner.Engine` method), 57
`_do_prepare()` (`krrcz.testing.landmark._LandmarkResumingTestingEngine` method), 78
`_done()` (`krrcz.seeding.SeedAct` method), 71
`_dump_session()` (in module `krrcz.krrcz_cli`), 82
`_expand_by_dependencies_and_list_manipulations()` (in module `krrcz.flow.graph.resolver`), 42
`_fill_color()` (in module `krrcz.flow.graph.visualizer`), 45
`_fill_dependencies_to_nodes()` (in module `krrcz.flow.graph.resolver`), 43
`_first_boot_action()` (`krrcz.seeding._IndirectSeedStrategy` static method), 67
`_fs` (`krrcz.api.Bit` attribute), 20
`_helpers` (`krrcz.api.Bit` attribute), 19
`_installation_medium_created()` (`krrcz.seeding.SeedAct` method), 71
`_interaction_request_fetcher_for_session()` (`krrcz.flow.Context` method), 39
`_interaction_request_fetcher_for_session()` (`krrcz.seeding._RemoteContext` method), 71

_internals (krrrez.api.Bit property), 20
 _ipc_config_object_path() (in module krrrez.flow.config), 52
 _ipc_dialog_hub_object_path() (in module krrrez.flow.writer), 64
 _ipc_dialog_hub_path() (in module krrrez.flow.writer), 64
 _is_valid_bit_name_segment() (in module krrrez.flow.bit_loader), 48
 _join_package_name() (in module krrrez.flow.bit_loader), 48
 _key_name_for_stage() (in module krrrez.seeding.api), 72
 _krrrez_do_not_derive_a_dependency (krrrez.seeding.api.Bit attribute), 72
 _krrrez_module_directories_helper() (in module krrrez.flow.bit_loader), 48
 _label() (in module krrrez.flow.graph.visualizer), 45
 _last_data_landmark_path() (in module krrrez.testing.landmark), 78
 _last_landmark_path() (in module krrrez.testing.landmark), 78
 _list_logs() (in module krrrez.krrrez_cli), 82
 _locals_path() (krrrez.flow.Context method), 38
 _log (krrrez.api.Bit property), 20
 _log() (krrrez.api.internal.AdHocLogger._BlockMode method), 26
 _log() (krrrez.api.internal.AdHocLogger._MessageMode method), 26
 _log() (krrrez.flow.logging.LoggerMode method), 55
 _log() (krrrez.flow.writer.Writer.LogBlock.BlockLoggerMode method), 63
 _log() (krrrez.flow.writer.Writer.LogBlock.MessageLoggerMode method), 63
 _mark_bit_installed() (krrrez.flow.Context method), 38
 _may_be_installed_before() (in module krrrez.flow.graph.resolver), 44
 _module_root_directory() (in module krrrez.coding), 36
 _modules() (in module krrrez.seeding.profile_loader), 73
 _optional_bits (krrrez.api.internal.BaseForSimpleBehaviorDependency property), 24
 _packages (krrrez.api.Bit attribute), 20
 _process (krrrez.flow.runner._ApplyTask property), 59
 _resolve_optional() (krrrez.api.internal.MayDeclareSecondaryBitUsages static method), 22
 _retry() (krrrez.seeding._IndirectSeedStrategy static method), 67
 _run (krrrez.testing.api.BundledTestBit attribute), 75
 _run (krrrez.testing.api.LandmarkBit attribute), 76
 _services (krrrez.api.Bit attribute), 20
 _session_path_to_log_path() (in module krrrez.flow.writer), 62
 _sessions_path() (krrrez.flow.Session static method), 40
 _stage1_profile() (krrrez.seeding.ReseedThisMachine method), 69
 _stage1_profile() (krrrez.seeding.SeedIndirectlyViaRemovable method), 68
 _stage1_profile() (krrrez.seeding._IndirectSeedStrategy method), 67
 _start_creating_installation_medium() (krrrez.seeding.SeedAct method), 70
 _start_finishing() (krrrez.seeding.SeedAct method), 71
 _start_helper() (krrrez.flow.runner.Engine method), 56
 _start_helper_s() (krrrez.flow.runner.Engine static method), 56
 _system_users (krrrez.api.Bit attribute), 20
 _target_device (krrrez.seeding.SeedAct property), 70
 _testing_context() (in module krrrez.testing), 75
 _try_resolve_bit_type() (krrrez.api.internal.MayDeclareSecondaryBitUsages static method), 22
 _verify_key_is_valid() (krrrez.flow.config.Controller static method), 50
 _was_killed (krrrez.flow.watch.Watch property), 60
 _watch_finishing() (krrrez.seeding.SeedAct method), 71
 _worker_pool_size (krrrez.flow.runner.Engine attribute), 56
 _worker_pool_size (krrrez.testing._TestingEngine attribute), 75

A

abort() (krrrez.flow.watch.Watch method), 60
 actual_full_name() (krrrez.api.internal.BareConfigValue method), 24
 add_bit() (krrrez.flow.graph.resolver._Plan method), 43
 add_bit() (krrrez.flow.graph.resolver.Plan method), 43
 add_krrrez_bit() (krrrez.coding.Profiles._Editor method), 32
 add_profile() (krrrez.coding.Tests._Editor method), 34
 add_profile_test() (krrrez.coding.TestPlans._Editor method), 36
 add_test() (krrrez.coding.TestPlans._Editor method), 36
 add_test_plan() (krrrez.coding.TestPlans._Editor method), 36

additional_needed_bits() (krrrez.api.internal.BaseForSimpleBehaviorDependency method), 24
 additional_needed_bits() (krrrez.api.internal.Dependency method), 23
 additional_needed_bits() (krrrez.testing.api._InstallBeforehandForTestBundle method), 76
 AdHocLogger (class in krrrez.api.internal), 26
 AdHocLogger._BlockMode (class in krrrez.api.internal), 26
 AdHocLogger._MessageMode (class in krrrez.api.internal), 26
 after (krrrez.flow.graph.Node property), 41
 afterwards (krrrez.api.internal.SimpleDependency property), 25
 all_available_test_plans() (in module krrrez.testing), 74
 all_bit_names (krrrez.api.internal.SimpleDependency property), 25
 all_bits() (in module krrrez.flow.bit_loader), 46
 all_bits() (krrrez.flow.graph.resolver._Plan method), 44
 all_bits() (krrrez.flow.graph.resolver.Plan method), 43
 all_descendants() (krrrez.flow.graph.Node method), 41
 all_install_bits (krrrez.flow.watch.Watch property), 60
 ALL_INSTALL_BITS_DIR_NAME (krrrez.flow.writer.Writer attribute), 62
 all_normal_bits() (in module krrrez.flow.bit_loader), 46
 all_profiles() (in module krrrez.seeding.profile_loader), 73
 all_test_sessions() (in module krrrez.testing), 74
 APPLY_METHOD_NAME (krrrez.coding.Bits attribute), 30
 apply_method_name_to_name() (krrrez.coding.Bits static method), 30
 args (krrrez.api.internal.BareConfigValue._AskForDialog attribute), 26
 args (krrrez.flow.dialog._DialogRequest property), 54
 as_html (krrrez.flow.graph.visualizer.Color property), 46
 ask_for (krrrez.api.internal.ConfigValue attribute), 29
 ask_for() (krrrez.flow.config._InteractiveController method), 51
 aux_name (krrrez.flow.writer.Writer.LogBlock property), 63
 available_keys() (krrrez.flow.config._Controller method), 50
 available_keys() (krrrez.flow.config._InteractiveController method), 51
 available_keys() (krrrez.flow.config.Controller method), 49
 available_target_devices (krrrez.api.internal.ProfileMeta property), 23

B

BareConfigValue (class in krrrez.api.internal), 26
 BareConfigValue._AskForDialog (class in krrrez.api.internal), 26
 BareConfigValue._Setter (class in krrrez.api.internal), 27
 BaseForSimpleBehaviorDependency (class in krrrez.api.internal), 24
 Beforehand (in module krrrez.api), 21
 began_at (krrrez.flow.watch.Watch property), 60
 Bit (class in krrrez.api), 19
 Bit (class in krrrez.seeding.api), 72
 bit (krrrez.flow.graph.Node property), 41
 bit (krrrez.flow.runner._ApplyTask property), 58
 bit() (in module krrrez.asset), 30
 Bit._Internals (class in krrrez.api), 20
 bit_by_name() (in module krrrez.flow.bit_loader), 46
 bit_by_name() (krrrez.flow.graph.resolver._Plan method), 44
 bit_by_name() (krrrez.flow.graph.resolver.Plan method), 43
 bit_documentation() (in module krrrez.flow.bit_loader), 47
 bit_for_secondary_usage() (in module krrrez.flow.bit_loader), 47
 bit_full_name() (in module krrrez.flow.bit_loader), 47
 BIT_GRAPH_FILE_NAME (krrrez.flow.writer.Writer attribute), 62
 bit_module_path() (in module krrrez.flow.bit_loader), 47
 bit_name (krrrez.testing.api._PartOf property), 76
 bit_name() (in module krrrez.flow.bit_loader), 47
 bit_name_to_profile_test_name() (krrrez.coding.ProfileTests static method), 32
 bit_name_to_profile_test_seed_name() (krrrez.coding.ProfileTests static method), 33
 bit_name_to_test_name() (krrrez.coding.Tests static method), 34
 bit_name_to_test_plan_name() (krrrez.coding.TestPlans static method), 35
 bit_names (krrrez.api.internal.SimpleDependency property), 24
 bit_names_for_stage() (krrrez.seeding.api.Bit method), 72
 BitNotFoundError, 48
 Bits (class in krrrez.coding), 30
 Bits._Editor (class in krrrez.coding), 31
 BITS_NAMESPACE (in module krrrez.flow), 37
 block (krrrez.api.internal.AdHocLogger property), 26
 block (krrrez.flow.logging.Logger property), 56
 block (krrrez.flow.writer.Writer.LogBlock property), 63

blue (*krrrez.flow.graph.visualizer.Color* property), 46
 browsable_profiles() (in module *krrrez.seeding.profile_loader*), 73
 BUILD_RAW (*krrrez.seeding.api.Stage* attribute), 72
 BUILD_SYSTEM (*krrrez.seeding.api.Stage* attribute), 72
 BundledTestBit (class in *krrrez.testing.api*), 75
 by_name() (*krrrez.flow.Session* static method), 39

C

CAUTION (*krrrez.flow.dialog.Style* attribute), 52
 choose() (*krrrez.flow.dialog.Processor* method), 53
 chosen_bits (*krrrez.flow.watch.Watch* property), 60
 CHROOT (*krrrez.seeding.api.Stage* attribute), 72
 clean_up_old_landmarks() (in module *krrrez.testing.landmark*), 77
 clean_up_old_test_sessions() (in module *krrrez.testing*), 74
 Color (class in *krrrez.flow.graph.visualizer*), 45
 Commands (class in *krrrez.krrrez_cli*), 82
 Commands.AppUnavailableError, 82
 Commands.Command_bits (class in *krrrez.krrrez_cli*), 82
 Commands.Command_dev_lab (class in *krrrez.krrrez_cli*), 83
 Commands.Command_logs (class in *krrrez.krrrez_cli*), 82
 Commands.Command_seeding (class in *krrrez.krrrez_cli*), 82
 Commands.Command_seeding.Command_profiles (class in *krrrez.krrrez_cli*), 82
 Commands.Command_studio (class in *krrrez.krrrez_cli*), 84
 Commands.Command_testing (class in *krrrez.krrrez_cli*), 83
 Commands.Command_testing.Command_landmarks (class in *krrrez.krrrez_cli*), 83
 Commands.Command_testing.Command_logs (class in *krrrez.krrrez_cli*), 83
 Commands.Command_testing.Command_plans (class in *krrrez.krrrez_cli*), 83
 condense() (*krrrez.flow.graph.Node* method), 42
 config (*krrrez.flow.Context* property), 38
 config_client_for_existing_session() (in module *krrrez.flow.config*), 52
 ConfigValue (class in *krrrez.api.internal*), 28
 ConfigValue (class in *krrrez.seeding.api*), 72
 ConfigValueAskForPrepared (class in *krrrez.api.internal*), 28
 ConfigValueAskForPrepared._AskFor (class in *krrrez.api.internal*), 28
 ConnectableToBit (class in *krrrez.api.internal*), 25
 Context (class in *krrrez.flow*), 37
 context (*krrrez.flow._Session* property), 40
 context (*krrrez.flow.config._Controller* property), 50
 context (*krrrez.flow.config._InteractiveController* property), 51

context (*krrrez.flow.config.Controller* property), 49
 context (*krrrez.flow.Session* property), 40
 context (*krrrez.krrrez_cli._Command* property), 81
 Controller (class in *krrrez.flow.config*), 49
 controller() (in module *krrrez.flow.config*), 52
 create() (*krrrez.coding.Bits._Editor* method), 31
 create() (*krrrez.coding.Profiles._Editor* method), 32
 create() (*krrrez.coding.ProfileTests._Editor* method), 33
 create() (*krrrez.coding.TestPlans._Editor* method), 36
 create() (*krrrez.coding.Tests._Editor* method), 34
 create() (*krrrez.flow.Session* static method), 39
 create_blank_context() (in module *krrrez.flow*), 39
 CRITICAL (*krrrez.flow.dialog.Style* attribute), 52

D

DEBUG (*krrrez.flow.logging.Severity* attribute), 56
 debug() (*krrrez.flow.logging.LoggerMode* method), 55
 default (*krrrez.api.internal.BareConfigValue* property), 27
 DEFAULT_ROOT_PATH (*krrrez.flow.Context* attribute), 38
 dependencies_for_bit() (*krrrez.flow.graph.resolver._Plan* method), 44
 dependencies_for_bit() (*krrrez.flow.graph.resolver.Plan* method), 43
 DependenciesCannotBeMetError, 44
 Dependency (class in *krrrez.api.internal*), 23
 dialog (*krrrez.flow.writer.Writer* property), 63
 dialog_endpoint (*krrrez.api.Bit._Internals* property), 20
 dialog_hub_path (*krrrez.flow.writer.Writer* property), 63
 disk_setup_index (*krrrez.seeding.ReseedThisMachine._Stage1ProfileGenerator._Partitions* attribute), 69

E

editor_for_bit() (*krrrez.coding.Bits* static method), 31
 editor_for_new_bit() (*krrrez.coding.Bits* static method), 31
 editor_for_new_profile() (*krrrez.coding.Profiles* static method), 31
 editor_for_new_profile_test() (*krrrez.coding.ProfileTests* static method), 33
 editor_for_new_test() (*krrrez.coding.Tests* static method), 34
 editor_for_new_test_plan() (*krrrez.coding.TestPlans* static method), 35
 editor_for_profile() (*krrrez.coding.Profiles* static method), 31
 editor_for_profile_test() (*krrrez.coding.ProfileTests* static method), 33
 editor_for_test() (*krrrez.coding.Tests* static method), 34

editor_for_test_plan() (krrcz.coding.TestPlans static method), 35
 ended_at (krrcz.flow.watch.Watch property), 60
 ENDED_AT_FILE_NAME (krrcz.flow.writer.Writer attribute), 63
 ENDED_AT_PATTERN (krrcz.flow.writer.Writer attribute), 62
 Endpoint (class in krrcz.flow.dialog), 54
 Engine (class in krrcz.flow.runner), 56
 Engine._WorkerLoop (class in krrcz.flow.runner), 57
 ensure_successful() (krrcz.flow.watch.Watch method), 60
 error (krrcz.flow.runner._ApplyTask property), 59
 Eventually (in module krrcz.api), 21
 exists (krrcz.flow.Session property), 40

F

failed_bits (krrcz.flow.runner.Engine._WorkerLoop property), 57
 FAILED_FILE_NAME (krrcz.flow.writer.Writer attribute), 63
 FATAL (krrcz.flow.logging.Severity attribute), 56
 fatal() (krrcz.flow.logging.LoggerMode method), 55
 final_disk_setup (krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator attribute), 69
 final_partition_setup (krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator attribute), 69
 finish() (krrcz.flow.writer.Writer method), 64
 forget_landmark() (in module krrcz.testing.landmark), 77
 full_name (krrcz.api.internal.BareConfigValue property), 27

G

generate() (krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator method), 69
 GenericDependencyAnnotation (class in krrcz.api.internal), 29
 get() (krrcz.api.Profile class method), 21
 get() (krrcz.flow.config._Controller method), 50
 get() (krrcz.flow.config._InteractiveController method), 51
 get() (krrcz.flow.config.Controller method), 49
 get_blank_contexts() (krrcz.flow.Context method), 39
 get_parser() (in module krrcz.krrcz_cli), 81
 get_sessions() (krrcz.flow.Context method), 39
 graph_for_bits() (in module krrcz.flow.graph.resolver), 42
 GraphError, 42
 green (krrcz.flow.graph.visualizer.Color property), 46

H

has_default (krrcz.api.internal.ConfigValueAskForPrepared._AskFor attribute), 28
 has_default (krrcz.flow.config._InteractiveController._AskFor attribute), 51
 has_default (krrcz.flow.dialog.HubEndpoint attribute), 54
 has_resumable_landmark() (in module krrcz.testing.landmark), 77
 host_disk (krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator attribute), 69
 host_partition (krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator attribute), 69
 Hub (class in krrcz.flow.dialog), 54
 HubEndpoint (class in krrcz.flow.dialog), 54

I

IfPicked (in module krrcz.api), 21
 IN_TARGET (krrcz.seeding.api.Stage attribute), 72
 IN_TARGET_LATE (krrcz.seeding.api.Stage attribute), 72
 INFO (krrcz.flow.logging.Severity attribute), 56
 info() (krrcz.flow.logging.LoggerMode method), 55
 initialize_as_secondary() (krrcz.api.Bit._Internals method), 20
 input() (krrcz.flow.dialog.Processor method), 53
 installed_bits (krrcz.flow.watch.Watch property), 61
 installed_bits() (krrcz.flow.Context method), 38
 installing_bits (krrcz.flow.watch.Watch property), 60
 interactive_controller() (in module krrcz.flow.config), 52
 InvalidStateError, 62
 is_bit_installed() (krrcz.flow.Context method), 38
 is_bit_name_for_normal_bit() (krrcz.coding.Bits static method), 30
 is_bit_name_for_profile_test() (krrcz.coding.ProfileTests static method), 32
 is_bit_name_for_profile_test_seed() (krrcz.coding.ProfileTests static method), 32
 is_bit_name_for_test() (krrcz.coding.Tests static method), 34
 is_bit_name_for_test_plan() (krrcz.coding.TestPlans static method), 35
 is_browsable (krrcz.api.internal.ProfileMeta property), 23
 is_confidential (krrcz.api.internal.BareConfigValue property), 27
 is_finished (krrcz.flow.watch.Watch.__LogFile property), 62
 is_hidden (krrcz.api.internal.ProfileMeta property), 23
 is_hidden (krrcz.api.Profile attribute), 21
 is_hidden (krrcz.profiles.cloud.Profile attribute), 65
 is_hidden (krrcz.profiles.iot_box.Profile attribute), 65
 is_hidden (krrcz.profiles.null.Profile attribute), 65

is_hidden (*krrez.profiles.workstation.Profile* attribute),
65
is_krrez_machine() (*in module krrez.flow*), 40
is_running (*krrez.flow.runner._ApplyTask* property), 59
is_running (*krrez.flow.runner.Engine._WorkerLoop*
property), 58
is_special_apply_method_name() (*kr-*
rez.coding.Bits static method), 30
ISO_FORMAT_LENGTH (*krrez.flow.writer.Writer* attribute),
62

K

key_is_valid() (*krrez.flow.config.Controller* static
method), 50
kill() (*krrez.flow.runner._ApplyTask* method), 59
krrez
module, 19
krrez.api
module, 19
krrez.api.internal
module, 22
krrez.asset
module, 30
krrez.asset.data
module, 30
krrez.asset.deploy_info
module, 30
krrez.asset.project_info
module, 30
krrez.coding
module, 30
krrez.flow
module, 37
krrez.flow.bit_loader
module, 46
krrez.flow.config
module, 49
krrez.flow.dialog
module, 52
krrez.flow.graph
module, 41
krrez.flow.graph.resolver
module, 42
krrez.flow.graph.visualizer
module, 44
krrez.flow.logging
module, 55
krrez.flow.runner
module, 56
krrez.flow.watch
module, 60
krrez.flow.writer
module, 62
krrez.krrez_cli

module, 81
krrez.profiles
module, 65
krrez.profiles.cloud
module, 65
krrez.profiles.iot_box
module, 65
krrez.profiles.null
module, 65
krrez.profiles.workstation
module, 65
krrez.seeding
module, 66
krrez.seeding.api
module, 72
krrez.seeding.profile_loader
module, 73
krrez.seeding.reseed
module, 71
krrez.seeding.system
module, 74
krrez.testing
module, 74
krrez.testing.api
module, 75
krrez.testing.landmark
module, 77
krrez_bits (*krrez.coding.Profiles._Editor* property), 32
KRREZ_ETC_DIR (*in module krrez.flow*), 37
krrez_module_directories() (*in module kr-*
rez.flow.bit_loader), 48
KRREZ_SRC_ROOT_DIR (*in module krrez.flow*), 37
KRREZ_USR_DIR (*in module krrez.flow*), 37
KRREZ_VAR_DIR (*in module krrez.flow*), 37
kwargs (*krrez.api.internal.BareConfigValue._AskForDialog*
attribute), 26
kwargs (*krrez.flow.dialog._DialogRequest* property), 54

L

landmark_name_for_session() (*in module kr-*
rez.testing.landmark), 77
landmark_size_on_disk() (*in module kr-*
rez.testing.landmark), 77
LandmarkBit (*class in krrez.testing.api*), 76
Later (*in module krrez.api*), 21
Lock (*class in krrez.api.internal*), 25
lock() (*krrez.flow.config._Controller* method), 50
lock() (*krrez.flow.config._InteractiveController*
method), 51
lock() (*krrez.flow.config.Controller* method), 49
lock() (*krrez.flow.Context* method), 39
log_block (*krrez.api.Bit._Internals* property), 20
log_block (*krrez.flow.runner._ApplyTask* property), 59
Logger (*class in krrez.flow.logging*), 56

LoggerMode (class in *krrcz.flow.logging*), 55

M

main() (in module *krrcz.krrcz_cli*), 84
 main() (*krrcz.krrcz_cli._Command* method), 82
 main() (*krrcz.krrcz_cli.Commands.Command_bits* method), 82
 main() (*krrcz.krrcz_cli.Commands.Command_dev_lab* method), 83
 main() (*krrcz.krrcz_cli.Commands.Command_logs* method), 82
 main() (*krrcz.krrcz_cli.Commands.Command_seeding* method), 83
 main() (*krrcz.krrcz_cli.Commands.Command_studio* method), 84
 main() (*krrcz.krrcz_cli.Commands.Command_testing* method), 83
 manipulate_resolution_plan() (*krrcz.api.internal.Dependency* method), 23
 manipulate_resolution_plan() (*krrcz.testing.api._InitDep* method), 75
 manipulate_resolution_plan() (*krrcz.testing.api._PartOf* method), 76
 mark_as_krrcz_machine() (in module *krrcz.flow*), 41
 MayDeclareSecondaryBitUsages (class in *krrcz.api.internal*), 22
 message (*krrcz.api.internal.AdHocLogger* property), 26
 message (*krrcz.flow.logging.Logger* property), 56
 message (*krrcz.flow.writer.Writer.LogBlock* property), 63
 method_name (*krrcz.api.internal.BareConfigValue._AskForDialog* attribute), 26
 method_name (*krrcz.flow.dialog._DialogRequest* property), 54
 module
 krrcz, 19
 krrcz.api, 19
 krrcz.api.internal, 22
 krrcz.asset, 30
 krrcz.asset.data, 30
 krrcz.asset.deploy_info, 30
 krrcz.asset.project_info, 30
 krrcz.coding, 30
 krrcz.flow, 37
 krrcz.flow.bit_loader, 46
 krrcz.flow.config, 49
 krrcz.flow.dialog, 52
 krrcz.flow.graph, 41
 krrcz.flow.graph.resolver, 42
 krrcz.flow.graph.visualizer, 44
 krrcz.flow.logging, 55
 krrcz.flow.runner, 56
 krrcz.flow.watch, 60
 krrcz.flow.writer, 62
 krrcz.krrcz_cli, 81

krrcz.profiles, 65
krrcz.profiles.cloud, 65
krrcz.profiles.iot_box, 65
krrcz.profiles.null, 65
krrcz.profiles.workstation, 65
krrcz.seeding, 66
krrcz.seeding.api, 72
krrcz.seeding.profile_loader, 73
krrcz.seeding.reseed, 71
krrcz.seeding.system, 74
krrcz.testing, 74
krrcz.testing.api, 75
krrcz.testing.landmark, 77
 module_log() (*krrcz.flow.writer.Writer* method), 64
 module_name (*krrcz.api.internal.BareConfigValue* property), 27

N

name (*krrcz.api.Bit* property), 20
 name (*krrcz.api.internal.ProfileMeta* property), 23
 name (*krrcz.api.internal.ProfileMeta._ProfileParameter* attribute), 23
 name (*krrcz.flow._Session* property), 40
 name (*krrcz.flow.Session* property), 40
 name_is_valid() (*krrcz.flow.Session* static method), 40
 new_config_server_for_session() (in module *krrcz.flow.config*), 52
 next_step_message (*krrcz.seeding._SeedThisMachineDirectly* attribute), 67
 next_step_message (*krrcz.seeding.ReseedThisMachine* attribute), 68
 next_step_message (*krrcz.seeding.SeedIndirectlyViaRemovable* attribute), 68
 next_step_message (*krrcz.seeding.SeedRemovableSystemDiskDirectly* attribute), 67
 next_step_message (*krrcz.seeding.SeedStrategy* property), 66
 Node (class in *krrcz.flow.graph*), 41
 node_for_bit() (*krrcz.flow.graph.Node* method), 42
 nodes_reachable_from() (*krrcz.flow.graph.Node* method), 41
 NORMAL (*krrcz.flow.dialog.Style* attribute), 52

O

ON_EXIT (*krrcz.seeding.api.Stage* attribute), 72
 open_parameters (*krrcz.api.internal.ProfileMeta* property), 23
 optional_bit_names (*krrcz.api.internal.SimpleDependency* property), 25

origin_bit (*krrez.api.Bit._Internals* property), 20

P

parent_context (*krrez.flow.Context* property), 38

partition_setup_index (*krrez.seeding.ReseedThisMachine._Stage1ProfileGenerator._PartitionInfo* attribute), 69

path (*krrez.flow._Session* property), 40

path (*krrez.flow.Context* property), 38

path (*krrez.flow.Session* property), 40

path (*krrez.flow.writer.Writer.LogBlock* property), 63

path2 (*krrez.flow._Session* property), 40

pick_multi() (*krrez.flow.dialog.Processor* method), 53

pick_single() (*krrez.flow.dialog.Processor* method), 53

Plan (*class in krrez.flow.graph.resolver*), 43

PREFIX_PATTERN (*krrez.flow.writer.Writer* attribute), 62

PREPARE (*krrez.seeding.api.Stage* attribute), 72

prepare_apply() (*krrez.api.Bit._Internals* method), 20

PREPARE_CHROOT (*krrez.seeding.api.Stage* attribute), 72

PREPARE_RAW (*krrez.seeding.api.Stage* attribute), 72

PREPARE_SYSTEM (*krrez.seeding.api.Stage* attribute), 72

Processor (*class in krrez.flow.dialog*), 52

Profile (*class in krrez.api*), 20

Profile (*class in krrez.profiles.cloud*), 65

Profile (*class in krrez.profiles.iot_box*), 65

Profile (*class in krrez.profiles.null*), 65

Profile (*class in krrez.profiles.workstation*), 65

PROFILE_BIT_NAME_PREFIX (*krrez.coding.Tests* attribute), 33

profile_full_name() (*in module krrez.seeding.profile_loader*), 73

profile_module_path() (*in module krrez.seeding.profile_loader*), 73

profile_name (*krrez.testing.api._InstallBeforehandForTestBundle* property), 76

profile_name() (*in module krrez.seeding.profile_loader*), 73

profile_names (*krrez.coding.Tests._Editor* property), 34

profile_test_name_to_bit_name() (*krrez.coding.ProfileTests* static method), 32

PROFILE_TEST_NAME_TO_BIT_NAME_PREFIX (*krrez.coding.ProfileTests* attribute), 32

PROFILE_TEST_NAME_TO_SEED_BIT_NAME_POSTFIX (*krrez.coding.ProfileTests* attribute), 32

profile_test_seed_name_to_bit_name() (*krrez.coding.ProfileTests* static method), 33

profile_tests (*krrez.coding.TestPlans._Editor* property), 36

ProfileMeta (*class in krrez.api.internal*), 22

ProfileMeta._ProfileParameter (*class in krrez.api.internal*), 22

Profiles (*class in krrez.coding*), 31

Profiles._Editor (*class in krrez.coding*), 31

ProfileTests (*class in krrez.coding*), 32

ProfileTests._Editor (*class in krrez.coding*), 33

progress (*krrez.flow.watch.Watch* property), 61

Provider (*class in krrez.flow.dialog*), 54

Q

question (*krrez.api.internal.BareConfigValue* property), 27

R

readme_pdf() (*in module krrez.asset.data*), 30

red (*krrez.flow.graph.visualizer.Color* property), 46

refresh_bit_graph() (*krrez.flow.writer.Writer* method), 64

relative_order() (*krrez.api.internal.BaseForSimpleBehaviorDependency* method), 24

relative_order() (*krrez.api.internal.Dependency* method), 23

remove_krrez_bit() (*krrez.coding.Profiles._Editor* method), 32

remove_profile() (*krrez.coding.Tests._Editor* method), 35

remove_profile_test() (*krrez.coding.TestPlans._Editor* method), 36

remove_test() (*krrez.coding.TestPlans._Editor* method), 36

remove_test_plan() (*krrez.coding.TestPlans._Editor* method), 36

request_arrived() (*krrez.flow.dialog._InteractionRequestFetcher* method), 54

request_disappeared() (*krrez.flow.dialog._InteractionRequestFetcher* method), 55

ReseedThisMachine (*class in krrez.seeding*), 68

ReseedThisMachine._Stage1ProfileGenerator (*class in krrez.seeding*), 69

ReseedThisMachine._Stage1ProfileGenerator._PartitionInfo (*class in krrez.seeding*), 69

root_log (*krrez.flow.writer.Writer* property), 64

run() (*in module krrez.seeding.reseed*), 71

run() (*krrez.flow.runner._ApplyTask.PipeReaderThread* method), 59

run() (*krrez.flow.runner.Engine._WorkerLoop* method), 58

RunFailedError, 62

S

scaled_by() (*krrez.flow.graph.visualizer.Color* method), 46

seed() (*krrez.seeding.SeedAct* method), 71

seed() (*krrez.seeding.SeedStrategy* method), 66

SeedAct (class in *krrcz.seeding*), 70
 SeedIndirectlyViaRemovable (class in *krrcz.seeding*), 68
 SeedRemovableSystemDiskDirectly (class in *krrcz.seeding*), 67
 SeedStrategy (class in *krrcz.seeding*), 66
 Session (class in *krrcz.flow*), 39
 session (*krrcz.api.Bit._Internals* property), 20
 session (*krrcz.flow.watch.Watch* property), 60
 session (*krrcz.flow.writer.Writer.LogBlock* property), 63
 set() (*krrcz.flow.config._Controller* method), 50
 set() (*krrcz.flow.config._InteractiveController* method), 51
 set() (*krrcz.flow.config.Controller* method), 49
 set_install_bits() (*krrcz.flow.writer.Writer* method), 63
 set_landmark() (in module *krrcz.testing.landmark*), 77
 Severity (class in *krrcz.flow.logging*), 56
 short_name (*krrcz.api.internal.BareConfigValue* property), 27
 SimpleDependency (class in *krrcz.api.internal*), 24
 Stage (class in *krrcz.seeding.api*), 72
 stage1_disk_setup (*krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator._PartitionInfo* attribute), 69
 stage1_partition_setup (*krrcz.seeding.ReseedThisMachine._Stage1ProfileGenerator._PartitionInfo* attribute), 69
 start() (*krrcz.flow.runner.Engine* method), 56
 start_resume_tests_from_landmark() (in module *krrcz.testing.landmark*), 77
 start_tests() (in module *krrcz.testing*), 74
 Style (class in *krrcz.flow.dialog*), 52
 sub_apply() (*krrcz.krrcz_cli.Commands.Command_bits* method), 82
 sub_info() (*krrcz.krrcz_cli.Commands.Command_seeding.Command_profiles* method), 82
 sub_list() (*krrcz.krrcz_cli.Commands.Command_bits* method), 82
 sub_list() (*krrcz.krrcz_cli.Commands.Command_logs* method), 82
 sub_list() (*krrcz.krrcz_cli.Commands.Command_seeding.Command_profiles* method), 82
 sub_list() (*krrcz.krrcz_cli.Commands.Command_testing.Command_landmarks* method), 83
 sub_list() (*krrcz.krrcz_cli.Commands.Command_testing.Command_logs* method), 83
 sub_list() (*krrcz.krrcz_cli.Commands.Command_testing.Command_plans* method), 83
 sub_remove() (*krrcz.krrcz_cli.Commands.Command_testing.Command_landmarks* method), 83
 sub_resume() (*krrcz.krrcz_cli.Commands.Command_testing.Command_landmarks* method), 83
 sub_run() (*krrcz.krrcz_cli.Commands.Command_testing* method), 83
 sub_show() (*krrcz.krrcz_cli.Commands.Command_logs* method), 82
 sub_show() (*krrcz.krrcz_cli.Commands.Command_testing.Command_logs* method), 83
 sub_sow() (*krrcz.krrcz_cli.Commands.Command_seeding* method), 83
 sub_status() (*krrcz.krrcz_cli.Commands.Command_bits* method), 82

T

 terminate() (*krrcz.flow.runner._ApplyTask* method), 59
 test_name_to_bit_name() (*krrcz.coding.Tests* static method), 34
 TEST_NAME_TO_BIT_NAME_PREFIX (*krrcz.coding.Tests* attribute), 33
 test_plan_for_test_session() (in module *krrcz.testing*), 74
 test_plan_name_to_bit_name() (*krrcz.coding.TestPlans* static method), 35
 TEST_PLAN_NAME_TO_BIT_NAME_PREFIX (*krrcz.coding.TestPlans* attribute), 35
 test_plans (*krrcz.coding.TestPlans._Editor* property), 36
 testbundle_name (*krrcz.testing.api._InstallBeforehandForTestBundle* property), 66
 TestPlans (class in *krrcz.coding*), 35
 TestPlans._Editor (class in *krrcz.coding*), 35
 Tests (class in *krrcz.coding*), 33
 tests (*krrcz.coding.TestPlans._Editor* property), 36
 Tests._Editor (class in *krrcz.coding*), 34
 to_flow_config_dict() (*krrcz.api.Profile* method), 21
 try_dump_pygraphviz() (in module *krrcz.flow.graph.visualizer*), 44
 turn_into_krrcz_system() (in module *krrcz.seeding.system*), 74
 type (*krrcz.api.internal.BareConfigValue* property), 27
 type (*krrcz.api.internal.ProfileMeta._ProfileParameter* attribute), 23

U

 ui_app() (*krrcz.krrcz_cli._Command* method), 81
 update() (*krrcz.flow.watch.Watch._LogFile* method), 62
 usage_does_not_imply_a_dependency() (in module *krrcz.api.internal*), 29

V

 value (*krrcz.api.internal.BareConfigValue* property), 27

W

 wait() (*krrcz.flow.watch.Watch* method), 60

`wait_terminated()` (*krrez.flow.runner._ApplyTask method*), [59](#)
`WARNING` (*krrez.flow.logging.Severity attribute*), [56](#)
`warning()` (*krrez.flow.logging.LoggerMode method*), [55](#)
`was_successful` (*krrez.flow.runner.Engine._WorkerLoop property*), [58](#)
`was_successful` (*krrez.flow.watch.Watch property*), [60](#)
`WasInterruptedError`, [59](#)
`WasUnsuccessfulError`, [59](#)
`Watch` (*class in krrez.flow.watch*), [60](#)
`Watch.__LogFile` (*class in krrez.flow.watch*), [62](#)
`Watch.__Monitor` (*class in krrez.flow.watch*), [61](#)
`Writer` (*class in krrez.flow.writer*), [62](#)
`Writer._ApplyBitLogBlock` (*class in krrez.flow.writer*), [63](#)
`Writer.LogBlock` (*class in krrez.flow.writer*), [63](#)
`Writer.LogBlock.BlockLoggerMode` (*class in krrez.flow.writer*), [63](#)
`Writer.LogBlock.MessageLoggerMode` (*class in krrez.flow.writer*), [63](#)