# Krrez

*Release 9.0.1256*

**unknown**

**Mar 15, 2025**

# CONTENTS

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).

# CHAPTER ONE

## 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.

# CHAPTER TWO

# 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 9.0.1256.

CHAPTER

THREE

# USER MANUAL

## 3.1 Getting Started

### 3.1.1 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.

### 3.1.2 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.

### 3.1.3 Testing

Let us now run our test. This will simulate the installation of Krrez 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 Krrez 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.

### 3.1.4 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*.

### 3.1.5 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 Krrez 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 `/krrez/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 Krrez modules, but your IDE does not know about them. In order to solve that, you have to add the `src` subdirectory of your Krrez 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.

## 3.2 Some Basic Terms

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

### 3.2.1 Bits

One of the most important concepts in Krrez 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 Krrez, 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 `krrez.bits` Python package, so its full module name could be like `krrez.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 `krrez.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 Krrez for some reason, also read `krrez.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. Krrez 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(krrez.api.Bit):

    @krrez.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 "`krrez.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 krrez.bits.some.other.bit


class Bit(krrez.api.Bit):

    _some_other_bit: krrez.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”.

### 3.2.2 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.

### 3.2.3 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 Krrez 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 Krrez’ 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.

## 3.3 Advanced

### 3.3.1 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 krrez.bits.sys.config


class Bit(krrez.api.Bit):

    _con fig: krrez.bits.sys.config.Bit

    def __apply__(self):
        ...
        foo = self._config.ask_for("foo").input("Choose a 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

### 3.3.2 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 *`krrez.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.

### 3.3.3 Technical Details

#### Usage of System Resources

Krrez 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 Krrez user interfaces and via command line.

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

**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 Krrez 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 Krrez. There is no need to read all that. The documentation links to some pieces of it sometimes, though.

The only parts relevant for Krrez automation code developers are `krrez.api`, builtin Bits in `krrez.bits`, and maybe Profiles in `krrez.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).

CHAPTER

FOUR

# COMMAND LINE INTERFACE REFERENCE

CHAPTER

FIVE

# API REFERENCE

## 5.1 krrez namespace

### 5.1.1 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.

**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.

**class** krrez.api.**Profile**(*, *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 Krrez Bits to install. Whenever to install a new Krrez system, the seeding procedure, and so the target system, is defined by the Profile you choose.

**Parameters**

- **hostname** (*str*) –
- **disks** (*list['krrez.bits.seed.steps.disks.Disk']*) –
- **raid_partitions** (*list['krrez.bits.seed.steps.disks.RaidPartition']*) –
- **network_interfaces** (*list['krrez.bits.seed.steps.networking.NetworkInterface']*) –
- **krrez_bits** (*list[str]*) –
- **arch** (*str*) –
- **operating_system** (*krrez.bits.seed.common.OperatingSystem*) –
- **seed_strategy** (`krrez.seeding.SeedStrategy`) –
- **keyboard** (*krrez.bits.seed.steps.keyboard.Keyboard*) –
- **locale** (*str*) –
- **timezone** (*str*) –
- **seed_user** (*Optional[krrez.bits.seed.steps.seed_user.SeedUser]*) –
- **config** (*dict[str, Any]*) –

krrez.api.**Eventually = <krrez.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: krrez.api.Eventually["krrez.bits.foo.Bit", "krrez.bits.bar.Bit", ...]`.

krrez.api.**Later = <krrez.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: krrez.api.Later["krrez.bits.foo.Bit", "krrez.bits.bar.Bit", ...]`.

krrez.api.**Beforehand = <krrez.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: krrez.api.Beforehand["krrez.bits.foo.Bit", "krrez.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 _try_resolve_bit_type**(*bit_type, *, all_bits=None*)

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

**Parameters**

- **bit_type** (*Union[str, type[krrez.api.Bit]]*) – The Bit specification.
- **all_bits** (*Iterable[type[krrez.api.Bit]]*) –

**Return type**

*Optional*[type[*krrez.api.Bit*]]

**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]*) –

class krrez.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[*`krrez.api.Bit`*]*) – The own Bit.
- **other_bit** (*type[*`krrez.api.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[*`krrez.api.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** (*Optional[bool]*) – Whether this is an afterwards-dependency.

**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** (*Optional[bool]*) – 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.

class krrez.api.internal.**ConnectableToBit**

Bases: ABC

**abstract _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 *krrez.api.Bit* is accessed.

**Parameters**

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

**Return type**

*Self*

class krrez.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 = krrez.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 `krrez.api.Bit` is accessed.

**Parameters**

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

class krrez.api.internal.**AdHocLogger**

Bases: `Logger`

class **_MessageMode**

Bases: `LoggerMode`

class **_BlockMode**

Bases: `_MessageMode`

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** (*Optional[str]*) –
- **module_name** (*Optional[str]*) –
- **full_name** (*Optional[str]*) –
- **question** (*Optional[_AskForDialog]*) –

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

Bases: object

**Parameters**

- **method_name** (*str*) –
- **args** (*Iterable*) –
- **kwargs** (*dict*) –

**_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 `krrez.api.Bit` is accessed.

**Parameters**

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

**class _Setter**(*value*)

Bases: Generic[_TConfigValueType]

**Parameters**

**value** (*_TConfigValueType*) –

**class** krrez.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** (*Optional[str]*) –
- **module_name** (*Optional[str]*) –
- **full_name** (*Optional[str]*) –
- **question** (*Optional[_AskForDialog]*) –

**class** krrez.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** (*Optional[str]*) –
- **module_name** (*Optional[str]*) –
- **full_name** (*Optional[str]*) –
- **question** (*Optional[_AskForDialog]*) –

**class _AskFor**(*config_value*)

Bases: Generic[_TConfigValueType], *Processor*[*_ConfigValueWithoutAskFor*[_TConfigValueType], *_ConfigValueWithoutAskFor*[_TConfigValueType], *_ConfigValueWithoutAskFor*[_TConfigValueType]], AllAbstractMethodsProvidedByTrick[*Processor*]

**Parameters**

**config_value** (ConfigValueAskForPrepared) –

class krrez.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 = krrez.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 = krrez.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 krrez.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.

class krrez.api.internal.**GenericDependencyAnnotation**(***dependency_config*)

Bases: object

Special objects for some ways to define dependencies.

krrez.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.apps package

class krrez.apps.**_ModelBase**(***kwargs*)

Bases: Model

**property app**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property context**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property all_bits**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

class krrez.apps.**_Model**(***kwargs*)

Bases: *_ModelBase*

class krrez.apps.**AppModel**(***kwargs*)

Bases: *_Model*

**property app**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

class krrez.apps.**MainModel**(***kwargs*)

Bases: *_Model*

### Subpackages

### krrez.apps.pieces package

### Subpackages

### krrez.apps.pieces.bits package

### Submodules

### krrez.apps.pieces.bits.app module

class krrez.apps.pieces.bits.app.**Model**(***kwargs*)

Bases: *AppModel*

class krrez.apps.pieces.bits.app.**View**(**args*, ***kwargs*)

Bases: ComposedView[*Model*]

### krrez.apps.pieces.bits.bit module

`class krrez.apps.pieces.bits.bit.`**`Model`**`(`*`**kwargs`*`)`

Bases: `Model`

**`property name`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property documentation`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property context`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property installing_session`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

`class krrez.apps.pieces.bits.bit.`**`View`**`(`*`*args`*`, `*`**kwargs`*`)`

Bases: `ComposedView[`*`Model`*`]`

### krrez.apps.pieces.bits.main module

`class krrez.apps.pieces.bits.main.`**`Model`**`(`*`**kwargs`*`)`

Bases: *`MainModel`*

**`property selected_bit`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property search_term`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property installing_session`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**property finish_was_confirmed**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property show_installed_only**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_normal_bits**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property visible_bits**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.bits.main.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

**property view_for_bit**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

## krrez.apps.pieces.dev_lab package

### Submodules

### krrez.apps.pieces.dev_lab.app module

**class** `krrez.apps.pieces.dev_lab.app.`**Model**(***kwargs*)

Bases: *AppModel*

**class** `krrez.apps.pieces.dev_lab.app.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

### krrez.apps.pieces.dev_lab.bit module

**class** `krrez.apps.pieces.dev_lab.bit.`**Model**(***kwargs*)

Bases: `Model`

**property main**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property name**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property module_path**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

class krrez.apps.pieces.dev_lab.bit.**View**(**args*, ***kwargs*)

Bases: ComposedView[*Model*]

**property header_text**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property module_path_text**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

### krrez.apps.pieces.dev_lab.main module

class krrez.apps.pieces.dev_lab.main.**Model**(***kwargs*)

Bases: *MainModel*

**property selected_custom_bit**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property selected_custom_profile**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property selected_custom_test**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property selected_custom_test_plan**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property all_custom_bits**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_custom_profiles**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_custom_tests**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_custom_test_plans**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_normal_bits**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_test_names**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_test_plan_names**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property all_profile_test_names**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.dev_lab.main.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[`*`Model`*`]`

**property view_for_bit_panel**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property view_for_profile_panel**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property view_for_test_panel**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property view_for_test_plan_panel**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

## krrez.apps.pieces.dev_lab.profile module

**class** `krrez.apps.pieces.dev_lab.profile.`**`Model`**(***kwargs*)

Bases: `Model`

**property profile**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property main**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property selected_bit_name**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property current_profile_bits**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.dev_lab.profile.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[`*`Model`*`]`

**property header_text**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

### krrez.apps.pieces.dev_lab.test module

**class** `krrez.apps.pieces.dev_lab.test.`**`Model`**`(`*****kwargs***`)`

Bases: `Model`

**property test**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property main**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property selected_profile_name**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property current_test_profiles**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property current_test_short_name**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**class** `krrez.apps.pieces.dev_lab.test.`**`View`**`(`**args*, ***kwargs*`)`

Bases: `ComposedView[`*`Model`*`]`

**property header_text**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

### krrez.apps.pieces.dev_lab.test_plan module

**class** `krrez.apps.pieces.dev_lab.test_plan.`**`Model`**`(`***kwargs*`)`

Bases: `Model`

**property test_plan**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property main**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property selected_test_name**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property selected_test_plan_name**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property selected_profile_name**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property current_test_plan_short_name**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property current_test_plan_tests**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property current_test_plan_test_plans**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property current_test_plan_profiles**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.dev_lab.test_plan.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[`*`Model`*`]`

**property header_text**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

### krrez.apps.pieces.log_browser package

#### Submodules

#### krrez.apps.pieces.log_browser.app module

**class** `krrez.apps.pieces.log_browser.app.`**`Model`**`(`*`**kwargs`*`)`

Bases: *`AppModel`*

**class** `krrez.apps.pieces.log_browser.app.`**`View`**`(`*`*args`*`, `*`**kwargs`*`)`

Bases: `ComposedView[`*`Model`*`]`

#### krrez.apps.pieces.log_browser.main module

**class** `krrez.apps.pieces.log_browser.main.`**`Model`**`(`*`**kwargs`*`)`

Bases: *`MainModel`*

**property all_sessions**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property selected_session**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**class** `krrez.apps.pieces.log_browser.main.`**`View`**`(`*`*args`*`, `*`**kwargs`*`)`

Bases: `ComposedView[`*`Model`*`]`

**property view_for_panel**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

### krrez.apps.pieces.runner package

#### Submodules

#### krrez.apps.pieces.runner.app module

**class** `krrez.apps.pieces.runner.app.`**`Model`**`(`*`**kwargs`*`)`

Bases: *`AppModel`*

**property bit_names**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property engine**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property confirm_after_installation**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property installing_session**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property done**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property was_successful**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** krrez.apps.pieces.runner.app.**View**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

### krrez.apps.pieces.runner.main module

**class** krrez.apps.pieces.runner.main.**Model**(***kwargs*)

Bases: `MainModel`

**property installing_session**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property engine**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property bit_names**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property done**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property was_successful**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property confirm_after_installation**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.runner.main.`**`View`**`(`**args*, ***kwargs*`)`

Bases: `ComposedView[`*`Model`*`]`

### krrez.apps.pieces.seeding package

#### Submodules

#### krrez.apps.pieces.seeding.app module

**class** `krrez.apps.pieces.seeding.app.`**`Model`**`(`***kwargs*`)`

Bases: *`AppModel`*

**class** `krrez.apps.pieces.seeding.app.`**`View`**`(`**args*, ***kwargs*`)`

Bases: `ComposedView[`*`Model`*`]`

#### krrez.apps.pieces.seeding.finishing module

**class** `krrez.apps.pieces.seeding.finishing.`**`Model`**`(`***kwargs*`)`

Bases: `Model`

**property session**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property finish_was_confirmed**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.seeding.finishing.`**`View`**`(`**args*, ***kwargs*`)`

Bases: `ComposedView[`*`Model`*`]`

## krrez.apps.pieces.seeding.main module

class `krrez.apps.pieces.seeding.main.Model(**kwargs)`

Bases: `MainModel`

class `State(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)`

Bases: `Enum`

property `state`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `all_profiles`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `selected_profile`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `seed_user`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `after_seeding_summary_message`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `finish_was_confirmed`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `selected_profile_open_parameters`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `additional_seed_config`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

property `all_target_devices`

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property selected_target_device**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property is_form_data_valid**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property seed_session**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property finish_from_here**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property finish_from_here_session**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class _SeedAct**(*main_app*, ***, *profile*, *target_device*)

Bases: *SeedAct*

**Parameters**

- **main_app** (`Model`) –
- **profile** (`krrez.api.Profile`) –
- **target_device** (*krrez.api.Path*) –

**class Profile**(***kwargs*)

Bases: `Model`

**property name**

Return the property value.
**Parameters**
`obj` – The object to get the value from.

**property type**

Return the property value.
**Parameters**
`obj` – The object to get the value from.

**class Target**(***kwargs*)

Bases: `Model`

**property path**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property description**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property label**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**class** `krrez.apps.pieces.seeding.main.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

## krrez.apps.pieces.seeding.new_seeding module

**class** `krrez.apps.pieces.seeding.new_seeding.`**`Model`**(***kwargs*)

Bases: `Model`

**property all_profiles**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property selected_profile**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property all_target_devices**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property selected_target_device**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property selected_profile_open_parameters**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property additional_seed_config**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property start_seed_func**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property is_form_invalid**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**class** `krrez.apps.pieces.seeding.new_seeding.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

**property view_for_open_parameters**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

## krrez.apps.pieces.seeding.session module

**class** `krrez.apps.pieces.seeding.session.`**Model**(***kwargs*)

Bases: `Model`

**property session**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property do_finishing**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property after_seeding_summary_message**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**property after_seeding_summary_message_answer**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

**class** `krrez.apps.pieces.seeding.session.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

**property view_for_summary**

Return the property value.

**Parameters**

`obj` – The object to get the value from.

### krrez.apps.pieces.studio package

#### Submodules

#### krrez.apps.pieces.studio.app module

**class** `krrez.apps.pieces.studio.app.`**`Model`**(***kwargs*)

Bases: *AppModel*

**class** `krrez.apps.pieces.studio.app.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[`*Model*`]`

#### krrez.apps.pieces.studio.help module

**class** `krrez.apps.pieces.studio.help.`**`Model`**(***kwargs*)

Bases: `Model`

**class** `krrez.apps.pieces.studio.help.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[`*Model*`]`

#### krrez.apps.pieces.studio.main module

**class** `krrez.apps.pieces.studio.main.`**`Model`**(***kwargs*)

Bases: *MainModel*

**class** `krrez.apps.pieces.studio.main.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[`*Model*`]`

**property tabs**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

#### krrez.apps.pieces.studio.welcome module

**class** `krrez.apps.pieces.studio.welcome.`**`Model`**(***kwargs*)

Bases: `Model`

**property visible_tab_names**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**property text_html**

Return the property value.

**Parameters**

**obj** – The object to get the value from.

**class** `krrez.apps.pieces.studio.welcome.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[`*Model*`]`

### krrez.apps.pieces.testing package

#### Submodules

#### krrez.apps.pieces.testing.app module

**class** krrez.apps.pieces.testing.app.**Model**(***kwargs*)

Bases: *AppModel*

**property start_with_test_plan**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property window_title**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**class** krrez.apps.pieces.testing.app.**View**(**args*, ***kwargs*)

Bases: ComposedView[*Model*]

#### krrez.apps.pieces.testing.main module

#### krrez.apps.pieces.testing.new_test_run module

**class** krrez.apps.pieces.testing.new_test_run.**Model**(***kwargs*)

Bases: Model

**property all_test_plans**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property selected_test_plan**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property start_test_func**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**property is_form_invalid**

Return the property value.

**Parameters**
**obj** – The object to get the value from.

**class** krrez.apps.pieces.testing.new_test_run.**View**(**args*, ***kwargs*)

Bases: ComposedView[*Model*]

## Submodules

### krrez.apps.pieces.session module

**class** `krrez.apps.pieces.session.`**`Model`**`(**kwargs)`

Bases: `Model`

**`property interactions`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property entries`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property bit_graph_image_svg`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property state_text`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property progress`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property is_finished`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property was_successful`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property session`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**`property headered_state`**

Return the property value.

**Parameters**
**`obj`** – The object to get the value from.

**property show_tree**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property verbose**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property actions**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property _session_watch**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.session.`**View**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

**property view_for_tree**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property interactions**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.session.`**_SessionViewConfigValuesProvider**(*model*)

Bases: `Provider`, `AllAbstractMethodsProvidedByTrick[Provider]`

**Parameters**
**model** (`Model`) –

## krrez.apps.pieces.session_interaction module

**class** `krrez.apps.pieces.session_interaction.`**Model**(***kwargs*)

Bases: `Model`

**Parameters**
**kwargs** (*dict[str, object]*) –

**property method_name**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property args**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property kwargs**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property answer**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.session_interaction.`**`View`**(**args*, ***kwargs*)

Bases: `ComposedView[Model]`

### krrez.apps.pieces.session_with_finish_confirmation module

**class** `krrez.apps.pieces.session_with_finish_confirmation.`**`Model`**(***kwargs*)

Bases: `Model`

**property with_finish_confirmation**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**property finish_was_confirmed**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

**class** `krrez.apps.pieces.session_with_finish_confirmation.`**`View`**(**args*, ***kwargs*)

Bases: `View`

**property confirmation_bar**

Return the property value.

**Parameters**
`obj` – The object to get the value from.

## krrez.aux package

### Submodules

### krrez.aux._pip module

### krrez.aux.data module

### krrez.aux.deploy_info module

### krrez.aux.project_info module

### krrez.coding package

Understand and modify some Krrez 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.

**class** krrez.coding.**Profiles**

Bases: `object`

Code actions on *krrez.api.Profile* implementations.

**class** krrez.coding.**ProfileTests**

Bases: `object`

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

**class** krrez.coding.**Tests**

Bases: `object`

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

**class** krrez.coding.**TestPlans**

Bases: `object`

Code actions on Test Plans (*krrez.api.Bit* implementations with a special name).

### krrez.flow package

The entire mechanism involved in applying bits (like :code:"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** (*Union[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** (*Union[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** (*Union[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** (*Union[str, Path]*) –

**Return type**
: Path

krrez.flow.**KRREZ_SRC_ROOT_DIR = Path('/home/pino/projects/krrez_9/src/krrez')**

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

**Parameters**
: **paths** (*Union[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 Krrez (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** (*Optional[Union[str, Path]]*) –

**DEFAULT_ROOT_PATH = Path('/var/lib/krrez/ctx')**

The default context path.

**Parameters**
: **paths** (*Union[str, Path]*) –

**Return type**
: Path

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.

**class** krrez.flow.**_Session**(*context*, *name*)

Bases: *Session*

Session implementation.

**Parameters**

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

krrez.flow.**is_krrez_machine**()

Whether this machine is enabled to be a Krrez machine.

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

See also *mark_as_krrez_machine()*.

**Return type**
bool

krrez.flow.**mark_as_krrez_machine**(*system_root_path*)

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

See also *is_krrez_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** (*Optional[type['krrez.api.Bit']]*) – The Bit to hold.

**property after: list[*krrez.flow.graph.Node*]**

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

**property bit: Optional[type[*krrez.api.Bit*]]**

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[krrez.api.Bit]*) – The bit to find.

**Return type**
: *Optional*[Node]

**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[krrez.api.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.

**`abstract add_bit`**`(`*bit*`)`

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

**Parameters**
**`bit`** (*`type[krrez.api.Bit]`*) – The Bit to add.

**Return type**
None

**`abstract bit_by_name`**`(`*bit_name*`)`

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

**Parameters**
**`bit_name`** (*`str`*) – The bit name.

**Return type**
*Optional*[type[*krrez.api.Bit*]]

**abstract all_bits**()

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

**Return type**
*Iterable*[type[*krrez.api.Bit*]]

**abstract dependencies_for_bit**(*bit*)

The (editable) list of dependencies for a bit.

**Parameters**
**bit** (*type[krrez.api.Bit]*) – The bit.

**Return type**
list[*krrez.api.internal.Dependency*]

**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.

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[krrez.api.Bit]*) – The first bit.
- **other_bit** (*type[krrez.api.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**
*Optional*[bytes]

`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

### krrez.flow.bit_loader module

Finding and loading bits.

krrez.flow.bit_loader.**all_bits**(*, *accept_cached=False*)

All bits.

See also *all_normal_bits()*.

**Parameters**
**accept_cached** (*bool*) –

**Return type**
*Iterable*[type[*krrez.api.Bit*]]

krrez.flow.bit_loader.**all_normal_bits**()

All normal bits.

Excludes stuff like tests. See also *all_bits()*.

**Return type**
*Iterable*[type[*krrez.api.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`** (*`Optional[Iterable[type[`*`krrez.api.Bit`*`]]]`*) –

**Return type**

type[*krrez.api.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`** (*`Union[str,`* `Bit`*`, type[`*`krrez.api.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`** (*`Union[str,`* `Bit`*`, type[`*`krrez.api.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`** (*`Union[`*`Bit`*`, type[`*`krrez.api.Bit`*`]]`*) –

**Return type**

*Path*

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

The documentation text of this bit.

**Parameters**

**`bit`** (*`Union[`*`Bit`*`, type[`*`krrez.api.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[krrez.api.Bit]*) – The bit type to instantiate.
- **used_by** (*Optional[Bit]*) – The bit that currently runs its apply method (or None). See `krrez.api.Bit._internals.origin_bit`.

**Return type**
Bit

`krrez.flow.bit_loader.`**`krrez_module_directories`**`(`**, with_builtin*`)`

All Krrez module root directories.

**Parameters**
**with_builtin** (*bool*) – Whether to include the builtin one as well.

**Return type**
list[hallyd.fs.Path]

`krrez.flow.bit_loader.`**`_join_package_name`**`(`**name_segments*`)`

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

**Parameters**
**name_segments** (*Optional[str]*) – The name segments to join.

**Return type**
str

`krrez.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

**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

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

Get the configuration value for a key.

**Parameters**

- **key** (*str*) – The key.
- **default** (*Optional[Any]*) – The default value, if the key does not exist.

**Return type**
*Optional*[*Any*]

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

Set the configuration value for a key.

**Parameters**

- **`key`** (*`str`*) – The key.
- **`value`** (*`Optional[Any]`*) – The new value.
- **`confidentially`** (*`bool`*) –

**Return type**
None

`abstract lock`(*key*)

Return a context manager that locks a particular key.

**Parameters**
**`key`** (*`str`*) – The key.

**Return type**
*ContextManager*[None]

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

All keys that are stored.

**Parameters**
**`with_confidential`** (*`bool`*) –

**Return type**
list[str]

`class krrez.flow.config._Controller`(*context*)

Bases: *`Controller`*

**Parameters**
**`context`** (`krrez.flow.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.

`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`*) –

`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.

`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.dialog module

User interaction facilities for usage inside *`krrez.api.Bit`* apply methods.

**class** `krrez.flow.dialog.`**`Style`**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `Enum`

**class** `krrez.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.

**class** `krrez.flow.dialog.`**`Endpoint`**

Bases: *`Processor`*`[Optional[str], Optional[int], Optional[list[int]]]`, `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.

**class** `krrez.flow.dialog.`**`Provider`**

Bases: *`Processor`*`[Future[Optional[str]], Future[Optional[int]], Future[Optional[list[int]]]]`, `ABC`

Abstract 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.

**class** `krrez.flow.dialog.`**`Hub`**(*path*)

Bases: `Hub[_DialogRequest, Optional[Any]]`

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 krrez.flow.dialog.`**`HubEndpoint`**`(`*dialog_hub*`)`

Bases: *Endpoint*, `AllAbstractMethodsProvidedByTrick[`*Endpoint*`]`

Dialog endpoint implementation that passes interaction requests through a *Hub*.

**Parameters**

**`dialog_hub`** (`Hub`) –

`class krrez.flow.dialog.`**`_InteractionRequestFetcher`**`(`**args*, *provider*, ***kwargs*`)`

Bases: `HubWorker[_DialogRequest, Optional[Any]]`

**Parameters**

**`provider`** (`Provider`) –

## krrez.flow.logging module

Logging interface.

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

Bases: `ABC`, `Generic[_TLoggerModeResult]`

`class krrez.flow.logging.`**`Logger`**

Bases: `ABC`

`class krrez.flow.logging.`**`Severity`**`(`*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*`)`

Bases: `Enum`

## krrez.flow.runner module

The engine.

`class krrez.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`krrez.flow.writer.Writer` for each run that feeds a listen and control interface with data, which can be consumed by a *`krrez.flow.watch.Watch`*. It can be customized for special cases by subclassing.

**`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`** (*`Optional[`*`LogBlock`*`]`*) – The optional log block (from another session) to use as root log block.

**Return type**
Watch

**exception** `krrez.flow.runner.WasUnsuccessfulError`

Bases: `Exception`

**exception** `krrez.flow.runner.WasInterruptedError`

Bases: *WasUnsuccessfulError*

## 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.

`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:** `Optional[list[str]]`

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:** `Optional[datetime]`

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

**property was_successful: bool**

Whether this session has finished successfully.

**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.

**class __Monitor**(*path*, *on_changed_func*)

Bases: `FilesystemMonitor`

**Parameters**

- **path** (*Path*) –
- **on_changed_func** (*Callable[[], None]*) –

**exception** `krrez.flow.watch.`**RunFailedError**

Bases: `RuntimeError`

Errors during the session runtime.

**exception** `krrez.flow.watch.`**InvalidStateError**

Bases: `Exception`

Watch is in an invalid state.

## krrez.flow.writer module

Engine session writer.

**class** `krrez.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** (*Optional[*`Writer.LogBlock`*]*) –
- **chosen_bits** (*list[type['krrez.api.Bit']]*) –
- **apply_message_func** (*Callable[[str], str]*) –

**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]]

`class` **MessageLoggerMode**(*log_block*)

Bases: *LoggerMode*[None]

`class` **_ApplyBitLogBlock**(**args*, *bit_name*, *orig_session*, ***kwargs*)

Bases: *LogBlock*

**set_install_bits**(*install_bits*)

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

**Parameters**
**install_bits** (*Iterable[type[krrez.api.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[krrez.api.Bit]]*) – The Bits that get applied at the moment.
- **installed_bits** (*list[type[krrez.api.Bit]]*) – The Bits that got applied.
- **failed_bits** (*list[type[krrez.api.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

**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.profiles namespace

### Submodules

### krrez.profiles.cloud module

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

Bases: *Profile*

**Parameters**

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

### krrez.profiles.iot_box module

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

Bases: *Profile*

**Parameters**

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

### krrez.profiles.null module

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

Bases: *Profile*

**Parameters**

**hostname** (*str*) –

### krrez.profiles.workstation module

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

Bases: *Profile*

**Parameters**

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

### krrez.seeding package

Krrez 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 Krrez to a target machine.

This includes the entire procedure, often starting with the creation of an installation medium, and ending with a fully installed Krrez 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** (*Optional[LogBlock]*) – The engine scope to run in.
- **target** (*Optional[Path]*) – The target device.

**Return type**

*ContextManager*[Watch]

**abstract property next_step_message: str**

Message that describes to the user how to proceed with the installation after *seed()* finished.

**abstract _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]

`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.

`abstract` **`_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]

**`_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.

`class krrez.seeding.`**`_SeedThisMachineDirectly`**

Bases: *`SeedStrategy`*

Seeds Krrez 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.

**`_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.

`class krrez.seeding.`**`SeedRemovableSystemDiskDirectly`**

Bases: *`SeedStrategy`*

Seeds Krrez 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.

**`_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.

`class krrez.seeding.`**`SeedIndirectlyViaRemovable`**

Bases: *`_IndirectSeedStrategy`*

Seeds Krrez 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.

**`_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.

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

Bases: *`_IndirectSeedStrategy`*

Reseeds Krrez 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 Krrez 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 Krrez version).

**Parameters**

**`profile_patchers`** (*`Iterable[Callable[[krrez.api.Profile], None]]`*) –

**`_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 krrez.seeding.**SeedAct**(*, *profile*, *target_device*)

Bases: ABC

**Parameters**

- **profile** (Profile) –
- **target_device** (*Path*) –

class krrez.seeding.**_RemoteContext**(*connection*, *root_path*)

Bases: *Context*

**Parameters**

- **connection** (*Connection*) –
- **root_path** (*Path*) –

## Subpackages

## krrez.seeding.reseed package

## 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.

class krrez.seeding.api.**Stage**(*value*, *names=None*, *, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

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

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

**Parameters**

**type** (*type[~_TConfigValueType]*) –

### 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`** (*`Union[Profile, type[krrez.api.Profile]]`*) –

**Return type**
: *Path*

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

The (short) name of a profile.

See also *`profile_full_name()`*.

**Parameters**
: **`profile`** (*`Union[str, Profile, type[krrez.api.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`** (*`Union[str, Profile, type[krrez.api.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[*krrez.api.Profile*]]

`krrez.seeding.profile_loader.`**`browsable_profiles`**()

All browsable profiles.

See also *`all_profiles()`*.

**Return type**
: list[type[*krrez.api.Profile*]]

### krrez.seeding.system module

System level helpers.

`krrez.seeding.system.`**`turn_into_krrez_system`**`(`*system_root_path*, *, *install_krrez_packages=True*`)`

Turn a target system into a Krrez system.

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

**Parameters**

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

**Return type**

None

### krrez.testing package

Krrez 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`** (*`Optional[Context]`*) – 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`** (*`Optional[Context]`*) – The context.

**Return type**

list[*krrez.flow.Session*]

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

Remove old test sessions.

**Parameters**

**`context`** (*`Optional[Context]`*) – The context.

**Return type**
None

`class krrez.testing._TestingEngine`

Bases: *Engine*

#### Submodules

#### krrez.testing.api module

#### krrez.testing.landmark module

### 5.1.2 Submodules

### 5.1.3 krrez.krrez_cli module

# PYTHON MODULE INDEX

# INDEX