
Annize

Release 7.0.6537

Author name not set

Nov 16, 2025

CONTENTS

1	License	3
2	Up-To-Date?	5
3	Dependencies	7
4	User Manual	9
4.1	First Steps	9
4.2	Annize Projects	11
4.3	Annize Studio	11
4.4	Annize Programming Interface	11
5	Command Line Interface Reference	13
5.1	Positional Arguments	13
5.2	Named Arguments	13
5.3	Sub-commands	13
6	API Reference	15
6.1	annize namespace	15
	Python Module Index	87
	Index	89

Annize is a Python-based framework for automating workflows. It allows users to define objects and tasks that refer to reusable Annize Features, which implement specific functionality. Features can be custom implementations or installed from external packages. Annize primarily serves as an execution engine, while most of the functionality comes from these modular Features.

Projects in Annize are defined using configuration files (usually XML), allowing tasks to be executed with various inputs. The Annize Studio provides an interface to conveniently configure objects, check for errors, and run tasks. It is designed for developers but can be adapted to a wide range of automation needs, offering flexibility for everything from simple to more complex workflows.

LICENSE

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

UP-TO-DATE?

Are you currently reading from another source than the homepage? If you are in doubt whether your package is up-to-date, you should visit the project homepage and check that. You are currently reading the documentation for version 7.0.6537.

DEPENDENCIES

Annize makes use of some third-party parts.



Required: **Python 3.13**



Required: **Python package hallyd** `~= 0.9003`



Required: **Python package klovve[graphical]** `~= 1.6`



Required: **Python package lxml** `~= 6.0`



Required: **Python package pycountry** `~= 24.6`



Required: **Python package pyperclip** `~= 1.11`



Recommended: **GNU/Linux**



Included: **background image** (License: <https://creativecommons.org/licenses/by-sa/3.0>; [from here](#))



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



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



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



Included: **icon set 'Oxygen'** (some icons; [from here](#))



Included: **third-party project logos** ([from here](#))

USER MANUAL

4.1 First Steps

4.1.1 Installation

Visit the Annize homepage to find the installation package that best suits your environment. The homepage offers various options depending on your system and preferences. Then install this package.

The simplest way to install Annize may be through `pip`, Python's package manager. For that way, prepare your `pip` environment at first (typically incl. a `venv`) and then run `pip install annize`.

After installation, you should be able to run Annize. To confirm that Annize is installed correctly, simply run `annize --help`.

4.1.2 The First Annize Project

In the following we will set up a first Annize projects in order to show the steps along the way. For that, we need to implement data structures and logic that we can use in Annize projects. Then we create a new Annize project, set up our functionality there, and finally execute it.

Preparing An Annize Feature

In Annize, all functionality is provided through **Features**. A Feature is a Python submodule within the `annize.features` namespace, which implements specific functionality that an Annize project can reference.

Annize itself primarily serves as the execution engine and does not include many built-in Features. Features can either be implemented manually or installed from additional packages (such as `Annize-pidev`, which provides Features used in Annize's own project).

To create a new Feature, a Python package must be added under the `annize.features` namespace. For example, to create a simple `hello_world` Feature:

1. In your Python packages directory, create the `annize/features` directory.
2. Inside that directory, create a file named `hello_world.py`.
3. Open that file in your preferred Python development environment or text editor.

At first we define a simple data structure:

```
class MyData:

    def __init__(self, foo: int, bar: str):
        self.foo = foo
        self.bar = bar
```

Then we define a task that we can execute (typically putting the `import` lines at the front of the file):

```
import annize.flow.run_context
import annize.user_feedback

class MyTask:

    def __init__(self, foo: int):
        self.foo = foo

    def __call__(self):
        # get all MyData objects defined in the project ...
        my_datas = annize.flow.run_context.objects_by_type(MyData)
        # and for all of them ...
        for my_data in my_datas:
            # if their 'foo' matches the task's one ...
            if my_data.foo == self.foo:
                # tell the user
                annize.user_feedback.message_dialog(my_data.bar, ["OK"])
```

This task would, once executed, collect all `MyData` objects, and show a message to the user for each one that has the 'right' `foo` value. Real Features would not interact with the user very often, but usually just do some real work.

Optionally, we can write Python 'docstrings' for both classes, including Sphinx-based parameter documentation for the constructor.

Creating A New Annize Project

With the functionality implemented, you can now create an Annize project and start using the Features you have defined.

1. Create an empty directory: Start by creating a new empty directory for your Annize project, e.g. somewhere in your home directory.
2. Launch Annize: Open Annize either from the start menu or by running `annize` from the command line.
3. Create a new project: In the Annize interface, create a new project and select the empty directory as the project location.
4. Expand the `project.xml` box.
5. Create a `MyData` object: Create a new `MyData` object inside the `project.xml` box. Assign some values to the `foo` and `bar` properties.
6. Create a `MyTask` object: Add a `MyTask` object to the `project.xml` box and assign a `foo` value.
7. Name the `MyTask`: Assign a name to the `MyTask` object.
8. Save the project: Once the setup is complete, save the project.

Executing The Annize Project

To execute the project, click the play button. A chooser will appear, displaying the name assigned to the `MyTask` object earlier. Select that task to begin execution.

Depending on the values assigned earlier, the execution will either complete immediately or display messages to the user during execution.

You can also create additional `MyTask` objects with different values and names, as well as add more `MyData` objects to the project.

What we have covered so far represents the basic functionality of Annize. With this foundation, you can already start building a variety of projects. The following sections will delve into more advanced details and features.

4.2 Annize Projects

An Annize **project** consists of object definitions that refer to available Annize Features. Some of these objects are typically executable tasks, which can be started during Annize execution.

The configuration files for an Annize project are stored in the `-meta` subdirectory of a project root directory. Some alternative names are allowed, but uncommon. At a minimum, this subdirectory will contain a `project.xml` file.

In addition to `project.xml`, it is possible to add more `project.*.xml` files. The structure of these files is flexible; Annize does not impose restrictions on where or how different objects are defined.

All objects within the project can be assigned names, and these names can be referenced throughout the project wherever needed.

Further details on the project structure are beyond this overview.

4.3 Annize Studio

The Annize **Studio** is what you see when you run `annize` (like you did in the ‘First Steps’).

The Annize Studio provides a comprehensive environment for configuring and managing Annize projects. Through this interface, you can:

- Add, remove, or modify objects: Easily configure your project by adding new objects, removing unnecessary ones, or editing existing configurations, including references to other objects and more.
- Identify configuration issues: The interface highlights problems with the current project setup, helping you quickly identify and resolve issues.
- Access Feature documentation: View documentation for the available Features directly within the interface, providing useful context and guidance as you build your project.
- Start project execution: Once your project is configured, you can initiate execution from the interface, allowing for quick testing and iteration.

4.4 Annize Programming Interface

There are several advanced topics related to the Annize API that are recommended for further reading:

- [`annize.fs`](#) and [`annize.features.files.common`](#).
- [`annize.features.base`](#).
- [`annize.features.task`](#).
- [`annize.i18n`](#) and [`annize.features.i18n.common`](#).
- [`annize.user_feedback`](#).

COMMAND LINE INTERFACE REFERENCE

```
usage: annize [-h] [--project PROJECT]
              [--with-answers-from-json-file WITH_ANSWERS_FROM_JSON_FILE]
              [--with-answers-from-json-string WITH_ANSWERS_FROM_JSON_STRING]
              [--with-answer WITH_ANSWER WITH_ANSWER]
              [command] ...
```

5.1 Positional Arguments

[command] Possible choices: do
 The command to execute.

5.2 Named Arguments

--project Project directory or Annize configuration file (otherwise: the current working directory). Annize will automatically try to find it in parent directories as well.

--with-answers-from-json-file Automate particular user feedback questions with given answers from a JSON file.
Default: []

--with-answers-from-json-string Automate particular user feedback questions with given answers from a JSON string.
Default: []

--with-answer Automate a particular user feedback question with a given answer.
Default: []

5.3 Sub-commands

5.3.1 do

Execute a task.

```
annize do [-h] [task_name]
```

Positional Arguments

task_name	Task name.
	Default: ''

API REFERENCE

6.1 annize namespace

6.1.1 Subpackages

annize.asset package

Submodules

annize.asset.data module

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

Parameters

culture (*str*)

Return type

Path

annize.asset.project_info module

annize.data package

Annize data structures.

Submodules

annize.data.color module

class `annize.data.color.Color`(* (*Keyword-only parameters separator (PEP 3102)*), *red*, *green*, *blue*)

Bases: `object`

A color.

Parameters

- **red** (*float*) – The color’s red component. A value between 0 and 1.
- **green** (*float*) – The color’s green component. A value between 0 and 1.
- **blue** (*float*) – The color’s blue component. A value between 0 and 1.

property red: `float`

The color’s red component. A value between 0 and 1.

This is a component of the RGB color space, as *green* and *blue*. There are other color spaces supported as well.

property green: float

The color's green component. A value between 0 and 1.

This is a component of the RGB color space, as [red](#) and [blue](#). There are other color spaces supported as well.

property blue: float

The color's blue component. A value between 0 and 1.

This is a component of the RGB color space, as [red](#) and [green](#). There are other color spaces supported as well.

property hue: float

The color's hue component. A value between 0 and 1.

This is a component of the HLS color space, as [lightness](#) and [saturation](#). There are other color spaces supported as well.

property lightness: float

The color's hue component. A value between 0 and 1.

This is a component of the HLS color space, as [hue](#) and [saturation](#). There are other color spaces supported as well.

property saturation: float

The color's hue component. A value between 0 and 1.

This is a component of the HLS color space, as [hue](#) and [lightness](#). There are other color spaces supported as well.

with_modified(*, red=None, green=None, blue=None, hue=None, lightness=None, saturation=None)

Return a color with some components set to new values.

Parameters

- **red** (*float* | *None*) – See [red](#).
- **green** (*float* | *None*) – See [green](#).
- **blue** (*float* | *None*) – See [blue](#).
- **hue** (*float* | *None*) – See [hue](#).
- **lightness** (*float* | *None*) – See [lightness](#).
- **saturation** (*float* | *None*) – See [saturation](#).

Return type

[Color](#)

property html_color_spec: str

The HTML color specification of this color.

__get_normed_value(value_2)**Parameters**

- **value_1** (*float* | *None*)
- **value_2** (*float*)

Return type

float

`__html_color_spec__part()`

Parameters

part_value (*int*)

Return type

str

annize.data.container module

class annize.data.container.**Basket**(*args, **kwargs)

Bases: list

annize.data.version module

Version numbers. See [Version](#).

class annize.data.version.**Version**(*, text=None, pattern, **segment_values)

Bases: object

A version number. It has a [text](#) (and a [pattern](#)), but also its [segments_tuples](#).

Parameters

- **text** (*str* / *None*) – The textual representation of the version number. If set, [segments_tuples](#) will be derived from it.
- **pattern** ([VersionPattern](#)) – The version pattern behind the textual representation of the version number.
- **segment_values** (*Any*) – Segment values that define this version number. Well known names are "major", "minor" and "build". This is an alternative to [text](#). Values are often of type *int*.

property segments_tuples: [Sequence](#)[[_SegmentTuple](#)]

This version number's segment tuples. Each tuple contains the segment name (well known ones are "major", "minor" and "build") and the value. The value is usually an *int*, but it could also be a *str* or anything else that is comparable.

If this version number was constructed with a [text](#), segment tuples are derived from it (using [pattern](#)). If not, segment tuples come from the [segment_values](#) specified during construction.

property segments_values: [dict](#)[*str*, *Any*]

Like [segments_tuples](#), but as a dictionary.

property text: *str*

The textual representation of this version.

property pattern: [VersionPattern](#)

The version pattern.

class annize.data.version.**VersionPattern**(*, parts)

Bases: object

A version pattern. Mostly used for translation between the textual representation and the segment tuples of a [Version](#).

Parameters

parts (*Iterable*[[VersionPatternPart](#)]) – The pattern parts.

property parts: Sequence[VersionPatternPart]

The pattern parts.

property segment_names

The pattern segment names.

text_to_segments_tuples(text)

Extract and return segment tuples from a given text.

Parameters

text (str) – The text to extract.

Return type

Sequence[_SegmentTuple]

segments_tuples_to_text(segments_tuples)

Return a textual representation for the given segment tuples.

Parameters

segments_tuples (Iterable[_SegmentTuple]) – The segment tuples.

Return type

str

class annize.data.version.VersionPatternPart

Bases: ABC

A part of a [VersionPattern](#). This can span the entire version pattern, or just one segment of it, or anything between.

abstract property segment_names: Sequence[str]

Names of all segments of this pattern part.

abstract property regexp_string: str

Return a regexp for this pattern part.

abstractmethod segments_tuples_to_text(segments_tuples)

Return a textual representation for the given segment tuples.

Parameters

segments_tuples (Sequence[_SegmentTuple]) – The segment tuples.

Return type

str

abstractmethod str_to_value(segment_name, s)

Return a value of the correct type (often int) for a given text representation of a segment.

Parameters

- **segment_name** (str) – The segment name.
- **s** (str) – The text to convert.

Return type

Any

_abc_impl = <_abc._abc_data object>

```
class annize.data.version.NumericVersionPatternPart(*, name)
```

Bases: [VersionPatternPart](#)

A version pattern part that represents one numeric segment of a version number.

Parameters

name (*str*) – The segment name.

property segment_names

Names of all segments of this pattern part.

property regexp_string

Return a regexp for this pattern part.

segments_tuples_to_text(*segments_tuples*)

Return a textual representation for the given segment tuples.

Parameters

segments_tuples – The segment tuples.

str_to_value(*segment_name*, *s*)

Return a value of the correct type (often `int`) for a given text representation of a segment.

Parameters

- **segment_name** – The segment name.
- **s** – The text to convert.

_abc_impl = <_abc._abc_data object>

```
class annize.data.version.SeparatorVersionPatternPart(*, text)
```

Bases: [VersionPatternPart](#)

A version pattern part that represents a separator in a version number.

Parameters

text (*str*) – The separator text.

property segment_names

Names of all segments of this pattern part.

property regexp_string

Return a regexp for this pattern part.

segments_tuples_to_text(*segments_tuples*)

Return a textual representation for the given segment tuples.

Parameters

segments_tuples – The segment tuples.

str_to_value(*segment_name*, *s*)

Return a value of the correct type (often `int`) for a given text representation of a segment.

Parameters

- **segment_name** – The segment name.
- **s** – The text to convert.

_abc_impl = <_abc._abc_data object>

```
class annize.data.version.OptionalVersionPatternPart(*, parts)
```

Bases: [VersionPatternPart](#)

A version pattern part that represents an optional part of a version number.

Parameters

parts (*Iterable*[[VersionPatternPart](#)]) – The inner parts.

property segment_names

Names of all segments of this pattern part.

property regexp_string

Return a regexp for this pattern part.

segments_tuples_to_text(*segments_tuples*)

Return a textual representation for the given segment tuples.

Parameters

segments_tuples – The segment tuples.

str_to_value(*segment_name*, *s*)

Return a value of the correct type (often `int`) for a given text representation of a segment.

Parameters

- **segment_name** – The segment name.
- **s** – The text to convert.

_abc_impl = `<_abc._abc_data object>`

```
class annize.data.version.ConcatenatedVersionPatternPart(*, parts)
```

Bases: [VersionPatternPart](#)

A version pattern part that represents the concatenation of other parts.

Parameters

parts (*Iterable*[[VersionPatternPart](#)])

property segment_names

Names of all segments of this pattern part.

property regexp_string

Return a regexp for this pattern part.

segments_tuples_to_text(*segments_tuples*)

Return a textual representation for the given segment tuples.

Parameters

segments_tuples – The segment tuples.

str_to_value(*segment_name*, *s*)

Return a value of the correct type (often `int`) for a given text representation of a segment.

Parameters

- **segment_name** – The segment name.
- **s** – The text to convert.

_abc_impl = `<_abc._abc_data object>`

annize.features namespace

Subpackages

annize.features.files namespace

Submodules

annize.features.files.common module

Files and directories.

class annize.features.files.common.**FsEntry**(*, path, root)

Bases: *FilesystemContent*

A filesystem location, either relative to the Annize project root directory or another root.

If it is already known whether the entry is a file or a directory, consider using *File* or *Directory* instead. Special files (e.g. symlinks) can also be represented by a *File*.

Parameters

- **path** (*str* | *Path* | *None*) – The path that points to the referenced content (relative to root).
- **root** (*str* | *Path* | *FilesystemContent* | *None*) – The root directory. If unset, it is the Annize project root directory.

property root: *FilesystemContent*

The root directory.

relative_path is considered to be relative to this one.

property relative_path: *Path*

The path that points to the referenced content (relative to *root*).

_path()

class annize.features.files.common.**File**(*, path, root)

Bases: *FsEntry*

A file location, either relative to the Annize project root directory or another root.

Parameters

- **path** (*str* | *Path* | *None*) – The path that points to the referenced content (relative to root).
- **root** (*str* | *Path* | *FilesystemContent* | *None*) – The root directory. If unset, it is the Annize project root directory.

class annize.features.files.common.**Exclude**(*, by_path_pattern, by_path, by_name_pattern, by_name)

Bases: object

An exclusion definition. Usually used with *Directory* and *DirectoryPart*.

Parameters

- **by_path_pattern** (*str* | *None*) – Exclude by this regexp pattern on the full path.
- **by_path** (*str* | *None*) – Exclude this path.
- **by_name_pattern** (*str* | *None*) – Exclude by this regexp pattern on the file name.
- **by_name** (*str* | *None*) – Exclude this file name.

`__does_exclude(by_text, by_pattern)`

Parameters

- **text** (*str*)
- **by_text** (*str*)
- **by_pattern** (*Pattern*)

Return type

bool

`does_exclude(relative_path, source, destination)`

Return whether a given location is excluded by this exclusion definition.

Parameters

- **relative_path** (*Path*) – The relative path to check for exclusion.
- **source** (*Path*) – The absolute source path.
- **destination** (*Path*) – The absolute destination path.

Return type

bool

`class annize.features.files.common.ExcludeAllBut(*, excludes)`

Bases: [Exclude](#)

A negative exclusion definition.

It will exclude an item whenever `_none_` of the given inner exclude definitions match.

Parameters

excludes (*list* [[Exclude](#)]) – List of inner exclude definitions.

`does_exclude(relative_path, source, destination)`

Return whether a given location is excluded by this exclusion definition.

Parameters

- **relative_path** – The relative path to check for exclusion.
- **source** – The absolute source path.
- **destination** – The absolute destination path.

`class annize.features.files.common.DirectoryPart(*, excludes, root, source_path=None, destination_path=None, path=None, destination_is_parent=False)`

Bases: `object`

A part of a directory. Used in [Directory](#).

Parameters

- **excludes** (*Iterable* [[Exclude](#)]) – List of exclusion definitions.
- **root** (*str* | *Path* | *FileSystemContent* | *None*) – The root directory. If unset, it is the root directory specified for the owning [Directory](#).
- **source_path** (*str* | *Path* | *None*) – The path that points to the referenced content (relative to `root`).

- **destination_path**(*str* | *Path* | *None*) – The relative destination path inside the owning *Directory*.
- **path**(*str* | *Path* | *None*) – Shorter way to set *source_path* and *destination_path* to the same path.
- **destination_is_parent**(*bool*) – Whether to consider the destination path as the parent of the new destination (instead of the new destination itself). The actual destination will have the same basename as the source then.

property excludes: *Sequence*[*Exclude*]

Exclusion definitions.

property root: *FilesystemContent* | *None*

The root directory (or *None* for the owning *Directory*.*root*).

source_path is considered to be relative to this one.

property source_path: *Path*

The path that points to the referenced content (relative to *root*).

property destination_path: *Path*

The relative destination path inside the owning *Directory*.

See also *destination_is_parent*.

property destination_is_parent: *bool*

Whether to consider the destination path as the parent of the new destination (instead of the new destination itself).

class *annize.features.files.common.Directory*(***, *path*, *root*, *excludes*, *parts*, *name*)

Bases: *FsEntry*

A directory location, either relative to the Annize project root directory or another root.

Depending on how it is configured, this might point to a dynamically generated temporary location (e.g. if it is composed of parts or excludes are specified).

Parameters

- **path**(*str* | *None*) – The path that points to the referenced directory (relative to *root*). If set, do not set *parts*!
- **root**(*str* | *Path* | *FilesystemContent* | *None*) – The root directory. If unset, it is the Annize project root directory.
- **excludes**(*Iterable*[*Exclude*]) – Exclusion specifications. If some are specified, this directory will be dynamically generated. If set, do not set *parts*!
- **parts**(*Iterable*[*DirectoryPart*]) – Directory parts. If some are specified, this directory will be dynamically generated. Also, do not set *path* or *excludes*!
- **name**(*str* | *None*) – The name that this directory shall have (instead of its original name). If specified, this directory will be dynamically generated. It must not contain directory separator characters (mostly *" / "*).

property parts: *Sequence*[*DirectoryPart*]

property excludes: *Sequence*[*Exclude*]

_path()

`__transfer_filter_for_exclude()`

Parameters

exclude (*Exclude*)

Return type

`annize.fs.Path.TTransferFilter`

class `annize.features.files.common.ProjectDirectory`

Bases: *FilesystemContent*

The Annize project root directory.

Parameters

generate_func – The content generator function. It has no parameters and returns an absolute path to the content (usually inside some temporary directory).

`_path()`

class `annize.features.files.common.MachineRootDirectory`

Bases: *FilesystemContent*

The machine root directory, i.e. `/`.

Parameters

generate_func – The content generator function. It has no parameters and returns an absolute path to the content (usually inside some temporary directory).

`_path()`

annize.features.i18n namespace

Submodules

annize.features.i18n.common module

Internationalization, i.e. translation and similar tasks.

class `annize.features.i18n.common._ProjectDefinedTranslationProvider`

Bases: *TranslationProvider*

Internally created translation provider for backing *String* instances.

translate(*string_name*, *, *culture*)

Return the translation of a given text for a given culture (or `None` if there is no translation for it).

Note: This does NOT obey the culture's fallbacks (see `Culture.fallback_cultures`)! That functionality is implemented in higher level parts of the API.

Parameters

- **string_name** – The string name.
- **culture** – The culture.

add_translations(*string_name*, *variants*)

Parameters

- **string_name** (*str*)
- **variants** (*dict[str, str]*)

Return type

None

`_ProjectDefinedTranslationProvider__translations_for_string_name(string_name)`**Parameters****string_name** (*str*)**Return type**

dict[str, str]

`_abc_impl = <_abc._abc_data object>``annize.features.i18n.common._translation_provider()``class annize.features.i18n.common.String(*, string_name, stringtr, **variants)`Bases: [ProvidedTrStr](#)

A translatable text defined in an Annize project.

Do not use directly. See `tr()`.**Parameters**

- **string_name** (*str* / *None*) – The string name.
- **stringtr** (*str* / *None*)
- **variants** (*str*)

`_abc_impl = <_abc._abc_data object>``class annize.features.i18n.common.Culture(*, iso_639_1_language_code, region_code, fallback_cultures)`Bases: [Culture](#)

A culture defined in an Annize project.

Do not use directly. See e.g. `from_iso_639_1_lang_code()` and `culture_by_spec()`.**Parameters**

- **english_lang_name** – The language name in English.
- **iso_639_1_language_code** (*str*) – The ISO-639-1 language code, like "en".
- **region_code** (*str* / *None*) – Optional language variant region_code, like "US".
- **fallback_cultures** (*list*[[Culture](#)]) – List of fallback cultures. See `fallback_cultures`.

`class annize.features.i18n.common.ProjectCultures(*, cultures)`Bases: `list`

Definition of an Annize project's target cultures.

Parameters**cultures** (*Sequence*[[Culture](#)])`annize.features.i18n.common.project_cultures()`Return a list of the current Annize project's target cultures. See also [ProjectCultures](#).**Return type***Sequence*[[Culture](#)]

annize.features.i18n.gettext module

gettext-based internationalization.

```
class annize.features.i18n.gettext.UpdatePOs(*, po_directory)
```

Bases: object

Parameters

po_directory (*str* | *Path*) – The directory with .po files.

```
class annize.features.i18n.gettext.GenerateMOs(*, po_directory, mo_directory, file_name)
```

Bases: object

Parameters

- **po_directory** (*str* | *Path* | *FileSystemContent*)
- **mo_directory** (*str* | *Path*)
- **file_name** (*str* | *None*)

```
class annize.features.i18n.gettext.TextSource(*, mo_directory, priority=0)
```

Bases: object

Parameters

- **mo_directory** (*str* | *Path*)
- **priority** (*int*)

Submodules

annize.features.base module

Project base information.

```
class annize.features.base.Data(*, project_name=None, pretty_project_name=None, summary=None,  
                                long_description=None, homepage_url=None, project_directory=None)
```

Bases: object

Parameters

- **project_name** (*str*)
- **pretty_project_name** (*TrStr* | *None*)
- **summary** (*TrStr* | *None*)
- **long_description** (*TrStr* | *None*)
- **homepage_url** (*str* | *None*)
- **project_directory** (*str* | *None*)

```
property project_name: str
```

```
property pretty_project_name: TrStr
```

```
property summary: TrStr
```

```
property long_description: TrStr
```

```
property homepage_url: str
```

```

    property project_directory: str

class annize.features.base.DateTime(*, iso)
    Bases: datetime

    Parameters
        iso (str)

class annize.features.base.Basket(*, items)
    Bases: Basket

    Parameters
        items (list[object])

class annize.features.base.FirstOf(*, objects)
    Bases: Basket

    Parameters
        objects (list[object])

annize.features.base._get_data(key, default)

    Parameters
        • key (str)
        • default (Any)

    Return type
        Any

annize.features.base.project_name()

    Return type
        str

annize.features.base.pretty_project_name()

    Return type
        TrStr

annize.features.base.summary()

    Return type
        TrStr

annize.features.base.long_description()

    Return type
        TrStr

annize.features.base.homepage_url()

    Return type
        str

annize.features.base.project_directory()

    Return type
        Path

```

`annize.features.base.sanitized_project_name(name)`

Parameters

name (*str*)

Return type

str

annize.features.task module

Tasks.

class `annize.features.task.Task(*, inner_tasks, is_advanced=False)`

Bases: `object`

Parameters

- **inner_tasks** (*Sequence[Callable]*)
- **is_advanced** (*bool*)

property is_advanced: `bool`

annize.flow package

Execution of Annize projects.

See e.g. [`annize.flow.runner.Runner`](#) and [`annize.flow.run_context.RunContext`](#).

Submodules

annize.flow.run_context module

Run contexts. Typically used by the infrastructure in the course of the execution of an Annize project.

See also [`RunContext`](#) and [`current\(\)`](#).

class `annize.flow.run_context.RunContext`

Bases: `object`

Holds data for a single execution of an Annize project (usually happening in a [`annize.flow.runner.Runner`](#)).

Beyond a few fixed data, like the root path of the Annize project configuration files, it stores every object that was created by definition in the Annize project configuration. Many parts of this API (e.g. many method names) use the term ‘object’ for all data items stored in a run context.

Each of those objects has at least one name. This can be a “friendly name”, i.e. a name that was explicitly specified in the project. If no name was specified, there will at least be an dynamically generated one, so every object is uniquely addressable by name. The dynamically generated names will differ between one project execution and another one, while friendly names are stable by nature.

There are further ways to access stored data, which do not involve names. See this class’ methods.

For each object in the store, arbitrary metadata can be stored as well.

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

This run context needs to be entered (`with-block`) during project execution.

Do not use directly.

`_IS_TOPLEVEL_OBJECT__METADATA_KEY = '__026G550sfUNm001005x000058mL'`

```
_NAMES__METADATA_KEY = '__026G550sfdcx002005x000058mL'
```

```
ANNIZE_CONFIG_DIRECTORY__NAME = '__026G550sfrr1003005x000058mL'
```

```
prepare(*, annize_config_directory)
```

Prepare the execution.

Needs to be called once, before the actual execution begins, in order to make some basic data available.

Parameters

annize_config_directory (*Path*) – The Annize project configuration root path.

Return type

None

```
object_by_name(name, default=None, *, create_nonexistent=False)
```

Return an object by one of its names (or a default value).

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

Parameters

- **name** (*str*) – An object name.
- **default** (*Any*) – The default value to return when no object exists with the given name.
- **create_nonexistent** (*bool*) – (If the default value is going to be returned because no object existed with the given name) Whether to store the default value in the data store with the given name, so it can be found later.

Return type

Any

```
object_names(obj)
```

Return all object names for a given object (with friendly names first).

This method always returns a non-empty list. Even if the given object was not stored at all yet, it automatically gets added to the store implicitly.

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

Parameters

obj (*Any*) – The object.

Return type

Sequence[*str*]

```
object_name(obj)
```

Return one object name for a given object (preferably a friendly one).

This method always returns a valid name. Even if the given object was not stored at all yet, it automatically gets added to the store implicitly.

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

Parameters

obj (*Any*) – The object.

Return type

str

set_object_name(*obj*, *name*)

Assign a name to an object.

All names assigned earlier remain valid as well.

See also [object_by_name\(\)](#), [object_names\(\)](#) and others.

Parameters

- **obj** (*Any*) – The object.
- **name** (*str*) – The new name.

Return type

None

objects_by_type(*obj_type*, *toplevel_only=True*)

Return all stored objects that are instance of a given type.

See also [add_object\(\)](#) and others.

Parameters

- **obj_type** (*type[T]*) – The type.
- **toplevel_only** (*bool*) – Whether to return only objects that are defined on project level.

Return type

Sequence

add_object(*obj*)

Add an object to the store and return its name.

If the object already is the store, and already has a friendly name, this one is returned. So, in fact, this method has the same effect as [object_name\(\)](#). It might just express your intent better than that one in some cases.

See also [object_by_name\(\)](#), [objects_by_type\(\)](#) and others.

Parameters

obj (*Any*) – The object to add.

Return type

str

is_friendly_name(*name*)

Returns whether the given name is a friendly one.

Parameters

name (*str*) – The name to check.

Return type

bool

is_toplevel_object(*obj*)

Return whether a given object represents the definition of an object on the Annize project file root level (e.g. by the top level tags in .xml configuration files).

Parameters

obj (*Any*) – The object to check.

Return type

bool

mark_object_as_toplevel(*obj*)

Mark an object as a toplevel one. See [is_toplevel_object\(\)](#).

Parameters

obj (*Any*) – The object.

Return type

None

object_metadata(*obj*, *key*, *default=None*)

Return a piece of metadata for a given object (or a default value if there is no value by the given key).

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

Parameters

- **obj** (*Any*) – The object.
- **key** (*str*) – The metadata key.
- **default** (*Any*) – The default value.

Return type

Any

set_object_metadata(*obj*, *key*, *value=None*)

Store a piece of metadata for a given object.

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

Parameters

- **obj** (*Any*) – The object.
- **key** (*str*) – The metadata key.
- **value** (*Any*) – The metadata value to store for this object and key.

Return type

None

__put_object(*name*, *obj*)**Parameters**

- **name** (*str*)
- **obj** (*Any*)

Return type

None

__object_raw_name(*obj*)**Parameters**

obj (*Any*)

Return type

str

__object_metadata_dict(*obj*)**Parameters**

obj (*Any*)

Return type

dict[str, Any]

`annize.flow.run_context.current()`

Return the current run context.

Note: In most cases you do not need to use this function directly. See the other functions defined on module level.

If there is no current run context (i.e. this function is called outside the execution of an Annize project), `OutOfContextError` will be raised.

Return type`RunContext`**exception** `annize.flow.run_context.OutOfContextError`Bases: `TypeError``annize.flow.run_context.object_by_name(name, default=None, *, create_nonexistent=False)`Same as `RunContext.object_by_name()` on the `_current_` run context (`current()`).**Parameters**

- **name** (`str`)
- **default** (`Any`)
- **create_nonexistent** (`bool`)

Return type`Any``annize.flow.run_context.object_names(obj)`Same as `RunContext.object_names()` on the `_current_` run context (`current()`).**Parameters****obj** (`Any`)**Return type**

list[str]

`annize.flow.run_context.object_name(obj)`Same as `RunContext.object_name()` on the `_current_` run context (`current()`).**Parameters****obj** (`Any`)**Return type**

str

`annize.flow.run_context.set_object_name(obj, name)`Same as `RunContext.set_object_name()` on the `_current_` run context (`current()`).**Parameters**

- **obj** (`Any`)
- **name** (`str`)

Return type

None

`annize.flow.run_context.objects_by_type(obj_type, toplevel_only=True)`

Same as `RunContext.objects_by_type()` on the `_current_run` context (`current()`).

Parameters

- `obj_type` (`type[T]`)
- `toplevel_only` (`bool`)

Return type

Sequence

`annize.flow.run_context.add_object(obj)`

Same as `RunContext.add_object()` on the `_current_run` context (`current()`).

Parameters

`obj` (*Any*)

Return type

`str`

`annize.flow.run_context.is_friendly_name(name)`

Same as `RunContext.is_friendly_name()` on the `_current_run` context (`current()`).

Parameters

`name` (*str*)

Return type

`bool`

`annize.flow.run_context.is_toplevel_object(obj)`

Same as `RunContext.is_toplevel_object()` on the `_current_run` context (`current()`).

Parameters

`obj` (*Any*)

Return type

`bool`

`annize.flow.run_context.object_metadata(obj, key, default=None)`

Same as `RunContext.object_metadata()` on the `_current_run` context (`current()`).

Parameters

- `obj` (*Any*)
- `key` (*str*)
- `default` (*Any*)

Return type

Any

`annize.flow.run_context.set_object_metadata(obj, key, value=None)`

Same as `RunContext.set_object_metadata()` on the `_current_run` context (`current()`).

Parameters

- `obj` (*Any*)
- `key` (*str*)
- `value` (*Any*)

Return type

None

annize.flow.runner module

The Annize runner.

See [Runner](#).

```
class annize.flow.runner.Runner(* , project, selected_task=None, user_feedback_answers,
                                user_feedback=None)
```

Bases: ABC

Base class for an Annize runner. It contains the base logic of project materialization and choosing and executing a task.

Parameters

- **project** ([ProjectNode](#))
- **selected_task** (*str* | *None*)
- **user_feedback_answers** (*dict[str, Any]*)
- **user_feedback** ([annize.user_feedback.UserFeedbackController](#))

```
run_runner()
```

Return type

None

```
abstractmethod show_task_chooser()
```

Return type

None

```
abstractmethod show_task_execution()
```

Return type

None

```
abstractmethod show_task_execution_success()
```

Return type

None

```
get_tasks()
```

Return type*list[str]*

```
get_selected_task()
```

Return type*str*

```
set_selected_task(task_name)
```

Parameters**task_name** (*str*)**Return type**

None

is_finished()

Return type
bool

get_success_state()

Return type
Tuple[bool, str]

wait_finished()

Return type
None

__do_run(*project*)

Parameters
project (*ProjectNode*)

__set_tasks(*tasks*)

Parameters
tasks (*list*[str])

Return type
None

__set_success_state(*success, message*)

Parameters

- **success** (*bool*)
- **message** (*str*)

Return type
None

_abc_impl = <_abc._abc_data object>

annize.fs package

Annize filesystem API.

Used by Annize Features.

See [Path](#), [FilesystemContent](#) and others.

class `annize.fs.FilesystemContent(generate_func)`

Bases: `object`

Base class for a source of arbitrary filesystem content.

It provides access to that content by [path\(\)](#). Some implementations will return a static path to already existing content, while other implementations will return a temporary path to ad-hoc generated content.

This content can be a file, a complete directory, or anything else. It could even return a path that points to nothing. It depends on the actual implementation what kind of content it provides.

This type is used instead of plain paths in situations where dynamic filesystem content might be exchanged (usually via some temporary files) instead of already existing files or directories. So, a `FilesystemContent` usually provides a path for reading. There is no strict rule against writing at that path, but that might lead to expected behavior (e.g. when the path points to a temporary copy of something, so changes do not take the

desired effect, or when it has undesired side effects on other consumers of the same instance). There might be features whose internal code does that in order to automatically handle relative paths.

Parameters

generate_func (*Callable*[[*TInputPath*]], *TInputPath*) – The content generator function. It has no parameters and returns an absolute path to the content (usually inside some temporary directory).

path()

Return the path that points to the content.

It always returns the same path and does not do any further processing when called more than once (so it is safe and cheap to call that multiple times).

Return type

Path

class `annize.fs.Path(*args, **kwargs)`

Bases: *Path*, *FilesystemContent*

A path.

This is compatible to `pathlib.Path`, but provides some convenience methods that can save a few lines of code for typical operations.

Each path is also a *FilesystemContent*. However, since *FilesystemContent* only allows absolute paths, using a relative path as a *FilesystemContent* will fail at runtime! See also `content()`.

Parameters

args (*str* / *Path* / *FilesystemContent*) – Path parts. Often this is one string, one *pathlib.Path* or one *FilesystemContent*. The latter one is only allowed as the first part.

_path()

Return itself (in order to implement *FilesystemContent*).

Return type

Path

path()

Return itself (in order to implement *FilesystemContent*).

Return type

Path

children()

Like `iterdir()`, but sorted by name.

Return type

Sequence[*Path*]

ctime()

Return the ctime for this path.

Return type

datetime

mtime()

Return the mtime for this path.

Return type

datetime

write_file(data)

Write data to a file at this path (like `write_text` or `write_bytes`).

Parameters

data (*bytes* | *TrStrOrStr*) – The data to write.

Return type

None

remove(*, missing_ok=True)

Remove the file, directory, symlink, ... at this path.

Parameters

missing_ok (*bool*) – Whether it is okay if there is nothing at this path.

Return type

None

file_size()

Return the file size in bytes.

Return type

int

temp_clone(*, temp_root_path=None, basename=None)

Return a temporary clone of the content at this path.

Parameters

- **temp_root_path** (*str* | *Path* | *None*) – Optional root directory for temporary files. If unset, an OS-default will be used.
- **basename** (*str* | *None*) – Optional new basename. If unset, the original one will be used.

Return type

[Path](#)

TTransferFilter

alias of `Callable[[Path, Path, Path], bool]`

copy_to(destination, *, destination_as_parent=False, merge=False, overwrite=False, transfer_filter=None)

Copy the file, directory, symlink, ... at this path to a given destination. All missing parent directories in the destination path get created automatically.

Parameters

- **destination** (*str* | *Path*) – The destination.
- **destination_as_parent** (*bool*) – Whether to consider the destination as the parent of the new destination (instead of the new destination itself). The actual destination will have the same basename as the source then.
- **merge** (*bool*) – Whether to merge the source content into the destination. If not, each new destination directory will replace the existing one or even fail.
- **overwrite** (*bool*) – Whether to allow overwriting of the destination.
- **transfer_filter** (`Callable[[Path, Path, Path], bool]` | *None*) – The optional transfer filter to use. It can exclude particular parts from the transfer. It is a function with three [Path](#) parameters: The relative path of an item, the absolute source path and the absolute destination path. It returns `False` to skip that item.

Return type[Path](#)**move_to**(*destination*, *, *destination_as_parent=False*, *merge=False*, *overwrite=False*, *transfer_filter=None*)

Move the file, directory, symlink, ... at this path to a given destination. All missing parent directories in the destination path get created automatically.

Parameters

- **destination** (*str* / [Path](#)) – The destination.
- **destination_as_parent** (*bool*) – Whether to consider the destination as the parent of the new destination (instead of the new destination itself). The actual destination will have the same basename as the source then.
- **merge** (*bool*) – Whether to merge the source content into the destination. If not, each new destination directory will replace the existing one or even fail.
- **overwrite** (*bool*) – Whether to allow overwriting of the destination.
- **transfer_filter** (*Callable*[[[Path](#), [Path](#), [Path](#)], *bool*] | *None*) – The optional transfer filter to use. It can exclude particular parts from the transfer. It is a function with three [Path](#) parameters: The relative path of an item, the absolute source path and the absolute destination path. It returns *False* to skip that item.

Return type[Path](#)**class TransferFilters**Bases: *object***class And**(**inner_filters*)Bases: *object***Parameters****inner_filters** (*Path.TTransferFilter*)**class _TransferHelper**Bases: *object***static transfer_to**(*source*, *destination*, *, *merge*, *overwrite*, *destination_as_parent*, *action*, *transfer_filter=None*)**Parameters**

- **source** ([Path](#))
- **destination** ([Path](#))
- **merge** (*bool*)
- **overwrite** (*bool*)
- **destination_as_parent** (*bool*)
- **action** (*Callable*)
- **transfer_filter** (*Callable*[[[Path](#), [Path](#), [Path](#)], *bool*] | *None*)

Return type[Path](#)**static transfer_action_copy**(*source*, *destination*)**Parameters**

- **source** ([Path](#))
- **destination** ([Path](#))

Return type*None*

static `transfer_action_move(source, destination)`

Parameters

- **source** ([Path](#))
- **destination** ([Path](#))

Return type

None

static `_TransferHelper__transfer_piece(action, transfer_filter, source, destination, merge, overwrite, relative_path=)`

Parameters

- **action** ([Callable](#))
- **transfer_filter** ([Callable](#)[[[Path](#), [Path](#), [Path](#)], [bool](#)] | None)
- **source** ([Path](#))
- **destination** ([Path](#))
- **merge** ([bool](#))
- **overwrite** ([bool](#))
- **relative_path** ([str](#))

Return type

None

`annize.fs.content(f, *, root=None)`

Return a [FilesystemContent](#) for an arbitrary given path or [FilesystemContent](#).

If the input already is a valid [FilesystemContent](#), it gets returned as-is. If the input is a string, it automatically gets interpreted as a path (like [Path](#)). If it is a relative path, this function will return a [FilesystemContent](#) that interprets it relative to the current Annize project root directory (which only makes sense when used inside an Annize project execution) or another root location.

Parameters

- **f** ([str](#) | [Path](#) | [FilesystemContent](#)) – The input path or filesystem content.
- **root** ([str](#) | [Path](#) | [FilesystemContent](#) | None) – The path or filesystem content to be used as root directory for relative paths in f.

Return type

[FilesystemContent](#)

`annize.fs.fresh_temp_directory(name=None, *, temp_root_path=None)`

Return a fresh empty temporary directory for arbitrary usage.

This directory will automatically be removed after the Annize project run has been finished. It can only be used for a `with`-block, which removes it directly after this block. Each instance can only be used once in the latter way.

For usage without a `with`-block, see [annize.fs.ext.FreshTempDirectory.path](#).

Parameters

- **name** ([str](#) | [Path](#) | None) – The optional directory name. Otherwise, the implementation will choose a name.
- **temp_root_path** ([str](#) | [Path](#) | None) – Optional root directory for temporary files. If unset, an OS-default will be used.

Return type

[FreshTempDirectory](#)

`annize.fs.dynamic_file(*, content, file_name=None, temp_root_path=None)`

Return a ‘filesystem content’ that provides a file with some given content.

Parameters

- **content** (*str* | *bytes* | *Callable*[[*str* | *bytes*]], *str* | *bytes*) – The content of this dynamic file. This may be either direct content (*str* or *bytes*) or a function that returns content.
- **file_name** (*str* | *None*) – The optional file name. Otherwise, the implementation will choose a name.
- **temp_root_path** (*str* | *Path* | *None*) – Optional root directory for temporary files. If unset, an OS-default will be used.

Return type*FilesystemContent***Submodules****annize.fs.ext module**

Annize filesystem API extensions.

Note: Commonly used functionality is also available in simpler ways (e.g. somehow in [annize.fs](#)).

class `annize.fs.ext.FreshTempDirectory`(*name=None*, *, *temp_root_path=None*)

Bases: `object`

A fresh empty temp directory for arbitrary usage.

See [annize.fs.fresh_temp_directory\(\)](#).

Do not use directly.

Parameters

- **name** (*str* | *Path* | *None*)
- **temp_root_path** (*str* | *Path* | *None*)

property path: *Path*

The path of this temp directory.

It is empty after creation and will be removed automatically after usage.

__cleanup()

class `annize.fs.ext.DynamicFile`(*, *content*, *file_name=None*, *temp_root_path=None*)

Bases: *FilesystemContent*

A filesystem content that provides a file with some given content.

See [annize.fs.dynamic_file\(\)](#).

Do not use directly.

Parameters

- **content** (*str* | *bytes* | *Callable*[[*str* | *bytes*]], *str* | *bytes*)
- **file_name** (*str* | *None*)
- **temp_root_path** (*str* | *Path* | *None*)

_TStaticContent = *str* | *bytes*

_TContentalias of `str | bytes | Callable[[], str | bytes]`**_path()**

```
class annize.fs.ext.Mount(src, dst, *, options=(), mount_command=('mount',),
                          umount_command=('umount',))
```

Bases: `object`

Mounting of a filesystem.

This mounts a filesystem as long as its context is entered (`with-block`).**Parameters**

- **src** (`str | Path`) – The filesystem to mount. Often a device file.
- **dst** (`str | Path`) – The mount-point.
- **options** (`Iterable[str]`) – Additional mount options.
- **mount_command** (`Iterable[str]`) – The mount command to use.
- **umount_command** (`Iterable[str]`) – The umount command to use.

property destination: `Path`

The mount-point.

annize.i18n package

Annize i18n backend.

The most fundamental mechanism around i18n is to get a translatable text (`TrStr`) from somewhere and get a translation from it, e.g. via `TrStr.translate()` or `translate()`.

Usually the translation is based on the current culture (`current_culture()`).

There can be `TrStr` coming from various sources with various implementations. A common one is `ProvidedTrStr`, which is backed by the so-called “translation providers”. One typical translation provider implementation is internally based on `gettext`. There is always at least one translation provider instance of that type, fetching translations from Annize own `gettext` translations. In general, translation providers could be based on arbitrary sources and are not restricted at all to `gettext`.

Other `TrStr` might have arbitrary other ways to translate texts, not backed by translation providers. Often they generate translations dynamically, e.g. by combining other `TrStr`.

At higher level, Annize i18n provides the following functionality:

- It hosts Annize own text translations. They are backed by `gettext` and typically referenced by `tr()` internally.
 - Annize projects are allowed to use those texts when convenient. A translation provider for them always exists, so

a project could contain nodes like `<String xmlns="annize:i18n" string_name="an_int_DebianPackage"/>`. Find all available texts in the top level directory i18n of Annize.

- It allows Annize projects to define and use own translated texts. - Either directly inside project configuration or via `gettext`.

The former can be done with a node like `<String xmlns="annize:i18n"><a:scalar a:arg_name="en">Yes</a:scalar>...</String>`. Usage of `gettext` involves the definition of a `annize.features.i18n.gettext.TextSource` and nodes like `<String`

```
xmlns="annize:i18n"><a:scalar a:arg_name="stringtr">tr("myOwnStringName")</a:scalar></String>.
```

More steps are needed to generate the required .mo-files (see below). Note: Even for texts that are directly defined in the project, if you add a `string_name` to them, you can also reference them in the same way as `gettext` based texts.

- It allows Annize projects to override Annize own text translations. - Either directly inside project configuration or via `gettext` (mostly like described above). It is also

possible add new languages or to override only some languages.

- It helps Annize projects to deal with `gettext` .mo- and .po-files; no matter whether these texts are used in the Annize project configuration or in the project's source code. See [annize.features.i18n.gettext.UpdatePOs](#) and [annize.features.i18n.gettext.GenerateMOs](#).

class `annize.i18n.TrStr`

Bases: `ABC`

Base class for translatable texts.

Each instance can hold the translation for one text for different cultures. In order to translate it to the current culture, the simplest way is to just apply `str()` on it.

See also [translate\(\)](#) and [_translation_for_culture\(\)](#).

translate(*culture=None*)

Return the translation of this text for the current culture or any other one, or raise [TranslationUnavailableError](#) if no translation is available for that culture (or its fallbacks; see [Culture.fallback_cultures](#)).

Parameters

culture (*CultureSpecT*) – The culture.

Return type

`str`

format(**args, **kwargs*)

Return a formatted variant of this text (i.e. similar to Python `str.format()`).

Parameters

- **args** – Formatting args.
- **kwargs** – Formatting kwargs.

Return type

[TrStr](#)

static tr(*string_name*)

Return a translatable text (by querying the registered translation providers). Note: Both `tr` functions are to be used by Annize only. External feature packages can only use them for own strings if they take care to add a translation provider for them.

Parameters

string_name (*str*) – The string name.

Return type

[TrStr](#)

abstractmethod _translation_for_culture(*culture*)

Return the translation of this text for a given culture (or `None` if there is no translation for it).

Note: This is implemented by subclasses, but usually not called directly from outside. See [translate\(\)](#). This does NOT obey the culture's fallbacks.

Parameters**culture** ([Culture](#)) – The culture.**Return type**

str | None

_abc_impl = <_abc._abc_data object>**annize.i18n.translate**(text, *, culture=None)

Translate a given text (if it is not a plain str) to the current culture or any other one, or raise [TranslationUnavailableError](#) if no translation is available for that culture (or its fallbacks; see [Culture.fallback_cultures](#)).

Parameters

- **text** ([TrStrOrStr](#)) – The text to translate.
- **culture** ([CultureSpecT](#)) – The culture.

Return type

str

annize.i18n.trstr(text)

Return a translatable text for a given text.

This is a no-op for translatable texts, but returns a (technically) translatable text for a plain str. In the latter case, the translation will be the input text for all cultures.

This is useful when you need a translatable text (e.g. as input parameter) but maybe only have a plain str.

Parameters**text** ([TrStrOrStr](#)) – The text.**Return type**[TrStr](#)**class annize.i18n.TranslationProvider**

Bases: [ABC](#)

Base class for objects that provide translations for some strings in some languages (here usually called: cultures).

Most translatable texts are backed by translation providers (some only indirectly or not at all). This class is a fundamental part of the Annize i18n API, although only small parts of Annize code need to deal with them directly.

See [translate\(\)](#) and also [add_translation_provider\(\)](#).

abstractmethod translate(string_name, *, culture)

Return the translation of a given text for a given culture (or None if there is no translation for it).

Note: This does NOT obey the culture's fallbacks (see [Culture.fallback_cultures](#))! That functionality is implemented in higher level parts of the API.

Parameters

- **string_name** (str) – The string name.
- **culture** ([Culture](#)) – The culture.

Return type

str | None

_abc_impl = <_abc._abc_data object>

`annize.i18n.add_translation_provider(provider, *, priority=0)`

Add a new translation provider.

When inside an Annize run context (see [annize.flow.run_context](#)), the translation provider will automatically be removed after the run context and will not affect other run contexts.

Parameters

- **provider** ([TranslationProvider](#)) – The new translation provider.
- **priority** (*int*) – The priority. Providers with lower priority value are queried earlier.

Return type

None

class `annize.i18n.Culture(english_lang_name, iso_639_1_language_code, region_code, fallback_cultures)`

Bases: `object`

Representation for an Annize culture. This includes the specification of a language and an optional language variant.

The major purpose of Annize i18n backend is to generate culture-specific translations for some texts.

Enter the culture context (`with-block`) in order to make it the current culture. This can also be done in a nested way (the former current culture does not take any effect meanwhile, but becomes the current culture again after this context). This is done by the UI, but also during the execution of an Annize project (iterating over its target cultures).

Annize projects choose their target cultures by means of [annize.features.i18n.common.Culture](#).

Do not use directly. See e.g. [from_iso_639_1_lang_code\(\)](#) and [culture_by_spec\(\)](#).

Parameters

- **english_lang_name** (*str*) – The language name in English.
- **iso_639_1_language_code** (*str*) – The ISO-639-1 language code, like "en".
- **region_code** (*str* / *None*) – Optional language variant region_code, like "US".
- **fallback_cultures** (*Iterable[Culture]*) – List of fallback cultures. See [fallback_cultures](#).

static `from_iso_639_1_lang_code(iso_639_1_language_code, region_code=None, *, fallback_cultures=())`

Return a culture by its ISO-639-1 language code (and an optional region_code).

Parameters

- **iso_639_1_language_code** (*str*) – The ISO-639-1 language code, like "en".
- **region_code** (*str* / *None*) – Optional language variant region_code, like "US".
- **fallback_cultures** (*Iterable[Culture]*) – List of fallback cultures. See [fallback_cultures](#).

Return type

[Culture](#)

property `english_lang_name: str`

The language name in English.

property iso_639_1_language_code: str

The ISO-639-1 language code, like "en". See also [region_code](#) and [full_name](#).

Note: This is "" for the [unspecified_culture](#).

property region_code: str | None

Optional language variant region_code, like "US". See also [iso_639_1_language_code](#) and [full_name](#).

property full_name: str

The full culture code (incl. the region code), like "en_US" or "en". See also [iso_639_1_language_code](#) and [region_code](#).

Note: This is "" for the [unspecified_culture](#).

property fallback_cultures: Sequence[Culture]

Fallback cultures.

Most parts of the API (unless documented otherwise) try those fallback cultures when an operation was not possible with this culture (i.e. there was no translation available for this culture). For that it would try with the 1st fallback culture, then maybe with its 1st fallback culture, and so on, then maybe with the 2nd fallback culture, until one of them finally succeeds the operation.

Note: For a culture with a region code, fallbacks usually contain the region-less culture implicitly, so e.g. de_DE and de_CH automatically fall back to de.

See also [unspecified_culture](#). That is always implicitly assumed to be a final fallback even if not part of this list.

culture_list()

Return a list that starts with this culture and then all fallback cultures in an expanded way, i.e. including their fallback cultures (recursively).

The result does never contain duplicates and also handles circular references of fallback cultures.

It will always contain [unspecified_culture](#) and its fallbacks as last resort!

For any function that explicitly regards fallback cultures, this is the list they iterate over.

Return type

Iterable[Culture]

__best_system_locale()**Return type**

str

__current_system_locale_setup()**Return type**

_TSystemLocaleSetup

__set_system_locale_setup()**Parameters**

system_locale_setup (*_TSystemLocaleSetup*)

Return type

None

`__set_env__var(value)`

Parameters

- **key** (*str*)
- **value** (*str* | *None*)

Return type

None

class `_TSystemLocaleSetup`(*LC_ALL: str* | *None*, *LANGUAGE: str* | *None*)

Bases: `object`

Parameters

- **LC_ALL** (*str* | *None*)
- **LANGUAGE** (*str* | *None*)

LC_ALL: *str* | *None*

LANGUAGE: *str* | *None*

`annize.i18n.current_culture()`

Return the current culture. If there is no current culture, raise *NoCurrentCultureError*.

During Annize task executions, this is *unspecified_culture* most of the time, but feature implementations may enter contexts for a different current culture, e.g. iterating over the project's target cultures. In UI contexts it is equal to *annize_user_interaction_culture*

Return type

Culture

`annize.i18n.tr(string_name, *, culture=None)`

Return the translation for a text (by querying the registered translation providers) in the current culture or any other one, or raise *TranslationUnavailableError* if no translation is available for that culture (or its fallbacks; see *Culture.fallback_cultures*).

Instead of this function, depending on the use case, *TrStr.tr()* might be the right choice. Note: Both *tr* functions are to be used by Annize only. External feature packages can only use them for own strings if they take care to add a translation provider for them.

Parameters

- **string_name** (*str*) – The string name.
- **culture** (*CultureSpecT*) – The culture.

Return type

str

class `annize.i18n.GettextTranslationProvider`(*mo_path*, *domain_name=None*)

Bases: *TranslationProvider*

A translation provider that is backed by .mo-files from `gettext`.

Parameters

- **mo_path** (*annize.fs.TInputPath*)
- **domain_name** (*str* | *None*)

translate(*string_name*, *, *culture*)

Return the translation of a given text for a given culture (or None if there is no translation for it).

Note: This does NOT obey the culture's fallbacks (see [Culture.fallback_cultures](#))! That functionality is implemented in higher level parts of the API.

Parameters

- **string_name** – The string name.
- **culture** – The culture.

class `_NoneTranslations`(*fp=None*)

Bases: `NullTranslations`

_abc_impl = `<_abc._abc_data object>`

class `annize.i18n.ProvidedTrStr`(*string_name*)

Bases: `TrStr`

Representation for a translatable text backed by the translations providers.

Do not use directly. See `tr()`.

Parameters

string_name (*str*) – The string name.

property `string_name`

_translation_for_culture(*culture*)

Return the translation of this text for a given culture (or None if there is no translation for it).

Note: This is implemented by subclasses, but usually not called directly from outside. See `translate()`. This does NOT obey the culture's fallbacks.

Parameters

culture – The culture.

_abc_impl = `<_abc._abc_data object>`

`annize.i18n._last_resort_culture` = `<annize.i18n.Culture object>`

The last resort culture. In some internal places, this is used as the final fallback if the specified culture (incl. its fallbacks) is not available.

`annize.i18n.unspecified_culture` = `<annize.i18n.Culture object>`

The 'unspecified' culture.

To be taken by default whenever no particular culture is specified, e.g. regarding projects that do not declare any project cultures at all. It is also the `current_culture()` during Annize task executions, unless a feature implementation explicitly enters a context for another current culture (e.g. iterating over all project cultures).

In translation operations, the unspecified culture will fall back to US English (which falls back to 'regionless' English).

Also, any translation operation that takes fallbacks into account (i.e. all that should regularly be used by external code) will finally fall back to the unspecified culture as the last resort.

`annize.i18n.culture_by_spec`(*culture*)

Return a culture for a given culture spec (i.e. a culture, a string representing one or None).

This is a no-op for a culture, return the current culture for None or uses `Culture.from_iso_639_1_lang_code()` for a string (after maybe splitting it into the language code and the region code). For an empty string, this is the `unspecified_culture`.

Parameters

culture (*CultureSpecT*) – The culture spec.

Return type

[Culture](#)

`annize.i18n.friendly_join_string_list(texts)`

Return a translatable string for a list of texts. They usually get concatenated with ", " between, but with something like " and " as the last separator; like "foo, bar and baz".

Parameters

texts (*Iterable[TrStrOrStr]*) – The input texts.

Return type

[TrStr](#)

exception `annize.i18n.NoCurrentCultureError`

Bases: `TypeError`

Error that occurs when the current culture was requested when there is no current culture.

exception `annize.i18n.TranslationUnavailableError(text, language)`

Bases: `TypeError`

Error that occurs when a translatable text was asked for translation to a language where no translation is available for.

Parameters

- **text** ([TrStr](#))
- **language** (*str*)

`annize.i18n._translation_providers()`

Return all translation providers (ordered ascending by their priority).

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

Return type

Sequence[[TranslationProvider](#)]

`annize.i18n._current_translation_providers_lists()`

`annize.i18n._annize_user_interaction_culture()`

Return type

[Culture](#)

class `annize.i18n._FixedTrStr(text)`

Bases: [TrStr](#)

Parameters

text (*str*)

_translation_for_culture (*culture*)

Return the translation of this text for a given culture (or `None` if there is no translation for it).

Note: This is implemented by subclasses, but usually not called directly from outside. See [translate\(\)](#). This does NOT obey the culture's fallbacks.

Parameters

culture – The culture.

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

```
class annize.i18n._FormattedTrStr(original_trstr, args, kwargs)
```

Bases: [TrStr](#)

Parameters

original_trstr ([TrStr](#))

```
_translation_for_culture(culture)
```

Return the translation of this text for a given culture (or None if there is no translation for it).

Note: This is implemented by subclasses, but usually not called directly from outside. See [translate\(\)](#). This does NOT obey the culture’s fallbacks.

Parameters

culture – The culture.

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

```
annize.i18n.annize_user_interaction_culture = <annize.i18n.Culture object>
```

The culture for interaction with the user. During project execution, this is potentially not the same as the [current_culture\(\)](#).

annize.object package

Annize objects.

There is no particular subclass that all Annize objects inherit from! Annize objects can be of arbitrary types (as long as their constructor has a signature that Annize can deal with).

There are some decorators for optional configuration and finetuning of Annize objects’ methods and attributes here.

```
annize.object.explicit_only(parameter_name)
```

Return a decorator function that marks a given parameter as “explicit only”, so potential arguments without an `arg_name` will never automatically be matched to that parameter.

Note: For any parameter with a type that already appeared at earlier parameters of the constructor signature, a similar effect will occur implicitly, because the materializer would always take the first possible parameter when it tries to auto-assign arguments.

Parameters

parameter_name (*str*) – The name of the constructor parameter to mark as “explicit only”.

Submodules

annize.object.config module

Configurations of objects and parts of it. Only used internally, e.g. by the functionality of [annize.object](#).

```
annize.object.config.parameter_config(for_type, parameter_name)
```

Return a parameter configuration for a given parameter of a given object’s constructor.

Parameters

- **for_type** (*type*) – The type.
- **parameter_name** (*str*) – The constructor’s parameter name.

Return type

[ParameterConfig](#)

class `annize.object.config.ParameterConfig`(*explicit_only*)

Bases: `object`

A parameter configuration. It contains additional, Annize-specific configuration for a parameter of an object's constructor.

See [`parameter_config\(\)`](#).

Parameters

`explicit_only` (*bool*)

`explicit_only`: `bool`

Whether this parameter is marked as “explicit only”. See [`annize.object.explicit_only\(\)`](#).

class `annize.object.config.InnerParameterConfig`(*explicit_only=None*)

Bases: `object`

An inner parameter configuration. Similar to [`ParameterConfig`](#) but not frozen and with default values.

Used for keeping configuration data in memory. For usage, see [`ParameterConfig`](#).

Parameters

`explicit_only` (*bool | None*)

`explicit_only`: `bool | None = None`

annize.project package

Annize projects.

See [`Project`](#), [`Node`](#) and also the submodules.

`annize.project.load`(*project_path*, *, *inspector=None*)

Load a project from disk. Return `None` if the given path does not point into an Annize project.

Parameters

- **`project_path`** (*str | Path*) – A path somewhere inside the project to be opened.
- **`inspector`** ([`FullInspector`](#) | *None*) – The custom project inspector to use.

Return type

[`ProjectNode`](#) | *None*

`annize.project.create_new`(*root_directory*, *subdirectory_name*='-meta', *, *inspector=None*)

Create a new Annize project.

This will create an initial version of the Annize project configuration on disk as well.

Parameters

- **`root_directory`** (*str | Path*) – The project root path.
- **`subdirectory_name`** (*str*) – The subdirectory name where to store the Annize configuration files inside the project root directory. This is not arbitrary but must be one of the well known ones!
- **`inspector`** ([`FullInspector`](#) | *None*) – The custom project inspector to use.

Return type

[`ProjectNode`](#)

class `annize.project.Node`

Bases: `ABC`

Nodes are the building blocks of a project.

They exist in a serialized way in the project files (usually xml), and when the project is loaded to memory (see [annize.project.loader](#)) they are represented by a structure of `Node` instances.

Each `Node` has various features (see methods and properties of this class), e.g. it can be observed for changes. Each node can also have children. This is just a base class for more specific node types, though. See also its subclasses in the same module.

The most relevant subclass in many regards is [ObjectNode](#).

`add_change_handler(handler, *, also_watch_children)`

Add a function that handles changes on this node.

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

Parameters

- **`handler`** (`Callable[[ChangeEvent], None]`) – The handler function to add.
- **`also_watch_children`** (`bool`) – Whether this function shall also observe this node's children.

Return type

`None`

`remove_change_handler(handler)`

Remove a change handler function that was added by [add_change_handler\(\)](#) earlier.

If that function was added multiple times, it will remove all of them. If the function was not added, this will do nothing.

Parameters

`handler` (`Callable[[ChangeEvent], None]`) – The handler function to remove.

Return type

`None`

property `parent`: [Node](#) | `None`

This node's parent node.

property `file`: [FileNode](#) | `None`

The file node that contains this node, or itself for file nodes, or `None` if it is not part of a file node.

This is the same as going [parent](#) upwards until a file node is reached.

property `project`: [ProjectNode](#) | `None`

The project node that contains this node, or itself for project nodes, or `None` if it is not part of a project node.

This is the same as going [parent](#) upwards until a project node is reached.

property `children`: `Sequence`[[Node](#)]

This node's child nodes.

`insert_child(i, node)`

Insert a new child node.

Parameters

- **`i`** (`int`) – The position.

- **node** ([Node](#)) – The node to insert.

Return type

None

append_child(*node*)

Append a new child node.

Parameters

- **node** ([Node](#)) – The node to append.

Return type

None

remove_child(*node*)

Remove a child node.

If that node is not a child node, it raises a `ValueError`.

Parameters

- **node** ([Node](#)) – The node to remove.

Return type

None

clone()

Clone this node and return that clone.

The clone is not connected to the original in any way, has no real marshaler (so it cannot be saved) and no changed handler and does not contain the undo history of the original. It is typically used for materialization or similar runtime purposes.

Return type

Self

description(*, *with_children=True*, *multiline=True*)**Parameters**

- **with_children** (*bool*)
- **multiline** (*bool*)

Return type

str

abstractmethod classmethod _allowed_child_types()

Return a list of node types that this node type allows to have as child nodes.

Return type

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

_clone__early(*new_node*)

Execute arbitrary steps during an early stage of node cloning.

Parameters

- **new_node** (*Self*) – The cloned node.

Return type

None

`_clone__late(new_node)`

Execute arbitrary steps during a late stage of node cloning.

Parameters

`new_node` (*Self*) – The cloned node.

Return type

None

`_property_changed(property_name, old_value)`

Parameters

• **`property_name`** (*str*)

• **`old_value`** (*Any*)

Return type

None

`abstractmethod _str_helper()`

Return type

Iterable[*str*]

`__description(indent, with_children, multiline)`

Parameters

• **`indent`** (*int*)

• **`with_children`** (*bool*)

• **`multiline`** (*bool*)

Return type

str

`__changed__child_added(child_node, child_position)`

Parameters

• **`child_node`** (*Node*)

• **`child_position`** (*int*)

Return type

None

`__changed__child_removed(child_node, child_position)`

Parameters

• **`child_node`** (*Node*)

• **`child_position`** (*int*)

Return type

None

`__changed__property_changed(node, property_name, old_value, new_value)`

Parameters

• **`node`** (*Node*)

• **`property_name`** (*str*)

- **old_value** (*Any*)

- **new_value** (*Any*)

Return type

None

__changed__call__handlers(*event*)

Parameters

event ([ChangeEvent](#))

Return type

None

class [ChangeEvent](#)(*target_node*)

Bases: [object](#)

Base class for events on a [Node](#). See subclasses and [Node.add_change_handler\(\)](#).

Parameters

target_node ([Node](#))

property **target_node**: [Node](#)

The target node this event is about.

class [__ChildrenListChangeEvent](#)(*target_node, child_node, child_position*)

Bases: [ChangeEvent](#)

Base class for events on a [Node](#) that are about changes on the list of children. See subclasses.

Parameters

- **target_node** ([Node](#))

- **child_node** ([Node](#))

- **child_position** (*int*)

property **child_node**: [Node](#)

The child node this event is about.

property **child_position**: *int*

The position of the child node in the list of children.

class [ChildAddedEvent](#)(*target_node, child_node, child_position*)

Bases: [__ChildrenListChangeEvent](#)

Node event that occurs when a child node was added.

Parameters

- **target_node** ([Node](#))

- **child_node** ([Node](#))

- **child_position** (*int*)

class [ChildRemovedEvent](#)(*target_node, child_node, child_position*)

Bases: [__ChildrenListChangeEvent](#)

Node event that occurs when a child node was removed.

Parameters

- **target_node** ([Node](#))

- **child_node** ([Node](#))
- **child_position** (*int*)

class PropertyChangeEvent(*target_node, property_name, old_value, new_value*)

Bases: [ChangeEvent](#)

Node event that occurs when a property of a node was changed.

Parameters

- **target_node** ([Node](#))
- **property_name** (*str*)
- **old_value** (*Any*)
- **new_value** (*Any*)

property property_name: **str**

The property name.

property old_value: **Any**

The old property value.

property new_value: **Any**

The new property value.

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

class annize.project.ProjectNode(*annize_config_directory*)

Bases: [Node](#)

An Annize project root node.

Each project has exactly one root node. It has no parent. Its children are the Annize project configuration files. It has no direct serialized representation (or, one could argue, it is the directory that contains these files).

Parameters

annize_config_directory (*str* | *Path*)

property annize_config_directory: **Path**

The “Annize config directory” of this Annize project.

This is not the same as the project’s “root directory”, but a subdirectory like ‘-meta’ inside it.

save()

Store the current state to the Annize project configuration files.

Return type

None

insert_child(*i, node*)

Insert a new child node.

Parameters

- **i** – The position.
- **node** – The node to insert.

remove_child(*node*)

Remove a child node.

If that node is not a child node, it raises a `ValueError`.

Parameters

node – The node to remove.

changes(***, *since*=0, *until*=9223372036854775807)

Return all changes that happened to the project, since the moment of loading it or any later point in time, and until now or any earlier point in time.

All timestamp arguments are based on an artificial clock (which basically increases by 1 for each change). See also [undo_changes\(\)](#).

Parameters

- **since** (*int*) – The timestamp where to start with returning changes (inclusive).
- **until** (*int*) – The timestamp where to stop with return changes (non-inclusive).

Return type

Sequence[[ChangeEvent](#)]

undo_changes(*since*)

Undo all changes that happened to the project since a given point in time. Timestamps are based on an artificial clock; see [changes\(\)](#).

Parameters

since (*int*) – The timestamp where to start with undoing changes (inclusive).

Return type

`None`

static load(*project_location*, ***, *inspector*=`None`)

Load and return a project node for a given Annize project location.

Do not use it directly. See [annize.project.load\(\)](#).

Parameters

- **project_location** (*str* / *Path*) – A path to somewhere inside an Annize project.
- **inspector** ([FullInspector](#) / `None`) – The custom project inspector to use.

Return type

[ProjectNode](#)

classmethod _allowed_child_types()

Return a list of node types that this node type allows to have as child nodes.

_clone__early(*new_node*)

Execute arbitrary steps during an early stage of node cloning.

Parameters

new_node – The cloned node.

_clone__late(*new_node*)

Execute arbitrary steps during a late stage of node cloning.

Parameters

new_node – The cloned node.

_str_helper()

__reset_change_history()

Return type

None

__handle_changed(event)

Parameters

event ([ChangeEvent](#))

Return type

None

__compacted_changes()

Parameters

events ([Sequence](#)[[ChangeEvent](#)])

Return type

[Sequence](#)[[ChangeEvent](#) | None]

__is_inverse_of(event_2)

Parameters

- **event_1** ([ChangeEvent](#))
- **event_2** ([ChangeEvent](#))

Return type

bool

_abc_impl = <_abc._abc_data object>

class `annize.project.FileNode(path, marshaler)`

Bases: [Node](#)

An Annize project file node.

Each project has one file node per configuration file. They are the children of the [ProjectNode](#). The children of a file node are mostly of type [ObjectNode](#), but can also be different ones.

Parameters

- **path** (*str* | [Path](#))
- **marshaler** (`annize.project.file_formats.FileFormat.Marshaler`)

property path: [Path](#)

The file path.

property marshaler: [Marshaler](#)

The marshaler of this file node. Do not use.

__clone__early(new_node)

Execute arbitrary steps during an early stage of node cloning.

Parameters

new_node – The cloned node.

_str_helper()

```
classmethod _allowed_child_types()
```

Return a list of node types that this node type allows to have as child nodes.

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

```
class annize.project.ArgumentNode
```

Bases: [Node](#), ABC

Base class for nodes that can be used as an argument, usually in an [ObjectNode](#).

See subclasses.

```
property name: str | None
```

The name of this argument node.

Names are used for a few purposes (the documentation will mention that where it is important), but primarily you can refer to a named argument with a [ReferenceNode](#) and you can use it for [append_to](#).

```
property append_to: str | None
```

The name of another argument node where this argument node gets appended to its children at runtime.

This essentially makes this argument node appear twice at runtime. It will also be in the place where it was defined; just a reference to that argument is created as a result.

```
property arg_name: str | None
```

The argument name where this argument is associated to in the parent object.

Valid argument names depend on the type of object that the parent is representing.

```
_str_helper()
```

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

```
class annize.project.ObjectNode(feature, type_name)
```

Bases: [ArgumentNode](#)

An Annize project object node.

In a typical Annize project, most nodes are object nodes. Most structure in their project files represent them (usually the tags in xml files). All the other node types are basically related to containing object nodes (like file nodes or the project root node) or have other support purposes.

Children are mostly other object nodes, [ScalarValueNode](#) or [ReferenceNode](#). They are associated to a particular parameter name (of the object type) by their [ArgumentNode.arg_name](#).

Parameters

- **feature** (*str*)
- **type_name** (*str*)

```
property feature: str
```

The Annize Feature name that provides this object.

```
property type_name: str
```

The name of the type of this object.

```
_str_helper()
```

```
classmethod _allowed_child_types()
```

Return a list of node types that this node type allows to have as child nodes.

```

    _abc_impl = <_abc._abc_data object>
class annize.project.ScalarValueNode
    Bases: ArgumentNode
    An Annize project scalar value node.
    It represents a fixed string value.
    property value: Any
        The string that this node represents.
    _str_helper()
    classmethod _allowed_child_types()
        Return a list of node types that this node type allows to have as child nodes.
    __shorten(max_length=100)

        Parameters
        • obj (Any)
        • max_length (int)

        Return type
        str
    _abc_impl = <_abc._abc_data object>
class annize.project.ReferenceNode
    Bases: ArgumentNode
    A reference node.
    This node represents a reference to another argument node (by its ArgumentNode.name)
    property reference_key: str | None
        The name of the node this node references to (or none).
    property on_unresolvable: OnUnresolvableAction
    _str_helper()
    classmethod _allowed_child_types()
        Return a list of node types that this node type allows to have as child nodes.
    class OnUnresolvableAction(*values)
        Bases: Enum
        FAIL = 'fail'
        SKIP = 'skip'
    _abc_impl = <_abc._abc_data object>
class annize.project.IgnoreUnavailableFeatureNode
    Bases: Node
    An Annize project ignore-unavailable-Feature node.

```

property feature: `str`

The name of the Feature that gets checked by this node. Empty string or "*" (the default) means all features.

_str_helper()

classmethod _allowed_child_types()

Return a list of node types that this node type allows to have as child nodes.

_abc_impl = `<_abc._abc_data object>`

exception `annize.project.FeatureUnavailableError(feature_name)`

Bases: `ModuleNotFoundError`

Parameters

feature_name (`str`)

exception `annize.project.BadStructureError(message)`

Bases: `ValueError`

Parameters

message (`str`)

exception `annize.project.MaterializerError(message)`

Bases: `TypeError`

Parameters

message (`str`)

exception `annize.project.ParserError(message)`

Bases: `ValueError`

Parsing error like bad input XML.

Parameters

message (`str`)

exception `annize.project.UnresolvableReferenceError(reference_key)`

Bases: [`MaterializerError`](#)

Parameters

reference_key (`str`)

Subpackages

`annize.project.file_formats` package

File formats for Annize configuration files.

See also the submodules.

class `annize.project.file_formats.FileFormat`

Bases: `ABC`

A file format for Annize configuration files.

class `Marshaler`

Bases: `ABC`

Base class for marshalers. A marshaler is responsible for one configuration file (i.e. it is associated to one [`annize.project.FileNode`](#)). It is provided by the [`FileFormat`](#) implementation when it creates file nodes and is responsible for keeping internal structures up-to-date whenever any changes to any node

(inside that file node) get applied. Based on that, it provides functionality like `save_file_node()` and others.

abstractmethod `add_change(change)`

Handle a given change, e.g. keep internal data structure up-to-date. Called for any change that occurs inside this file node.

Parameters

change (`ChangeEvent`) – The change.

Return type

None

abstractmethod `save_file_node()`

Save this file node back to disk.

Return type

None

abstractmethod `serialize_node(node)`

Return a serialized byte string for a given node, e.g. for clipboard operations.

Parameters

node (`ArgumentNode`) – The node to serialize.

Return type

bytes

`_abc_impl = <_abc._abc_data object>`

class `NullMarshaler`

Bases: `Marshaler`

A marshaler that does nothing. It should only be used in particular situations, like for temporarily created nodes.

add_change(change)

Handle a given change, e.g. keep internal data structure up-to-date. Called for any change that occurs inside this file node.

Parameters

change – The change.

save_file_node()

Save this file node back to disk.

serialize_node(node)

Return a serialized byte string for a given node, e.g. for clipboard operations.

Parameters

node – The node to serialize.

`_abc_impl = <_abc._abc_data object>`

abstractmethod `load_file_node(file, inspector)`

Read the given file and return a project file node for it.

That file node has a marshaler, which keeps track of changes (it gets notified by the infrastructure for each change) and is able to save the node back to its file.

Parameters

- **file** (`str` / `Path`) – The file to load.
- **inspector** (`FullInspector`) – The project inspector to use.

Return type`FileNode`**abstractmethod** `new_file_node(file, inspector)`

Return a new empty project file node.

That file node has a marshaler; see `load_file_node()`.**Parameters**

- **file** (`str` / `Path`) – The new file. It should not exist already.
- **inspector** (`FullInspector`) – The project inspector to use.

Return type`FileNode`**serialize_node**(`node`)

Return a serialized byte string for a node, e.g. for clipboard operations.

Parameters**node** (`ArgumentNode`) – The node to serialize.**Return type**`bytes`**abstractmethod** `deserialize_node(s, inspector)`

Return a node for a serialized string, e.g. for clipboard operations.

Parameters

- **s** (`bytes`) – The serialized string.
- **inspector** (`FullInspector`) – The project inspector to use.

Return type`ArgumentNode``_abc_impl = <_abc._abc_data object>``annize.project.file_formats.register_file_format(format_name)`

Return a decorator that registers a file format.

Parameters**format_name** (`str`) – The format name.**Return type**`Callable``annize.project.file_formats.file_format(format_name)`Return a file format by its name (or None if not available). See also `all_file_format_names()`.**Parameters****format_name** (`str`) – The format name. A typical name is "xml".**Return type**`FileFormat` | `None``annize.project.file_formats.all_file_format_names()`

Return all known file format names.

Return type`Sequence[str]`

`annize.project.file_formats.load_project(project_annize_config_directory, *, inspector)`

Load an Annize project from its configuration directory.

Do not use it directly. See `annize.project.load()`.

Parameters

- **project_annize_config_directory** (*str* / *Path*) – The Annize project configuration directory
- **inspector** (*FullInspector*) – The project inspector to use.

Return type

ProjectNode

Subpackages

`annize.project.file_formats.xml` package

Subpackages

`annize.project.file_formats.xml.node_representation_handlers` package

Submodules

`annize.project.file_formats.xml.node_representation_handlers.argument` module

`annize.project.file_formats.xml.node_representation_handlers.file` module

`annize.project.file_formats.xml.node_representation_handlers.ignore_unavailable_feature` module

`annize.project.file_formats.xml.node_representation_handlers.object` module

`annize.project.file_formats.xml.node_representation_handlers.reference` module

`annize.project.file_formats.xml.node_representation_handlers.scalar_value` module

Submodules

`annize.project.file_formats.xml.marshaler` module

`annize.project.materializer` package

Materializing of Annize projects into a working runtime structure (usually used by the Runner application).

See `materialize()`.

All submodules are only used internally by this one. There is a core part, some preprocessor functions, some behaviors that implement what it does for different types of project nodes, and the object factory.

class `annize.project.materializer.MaterializationResult`(*root_objects*, *node_association*, *problems*)

Bases: `object`

Parameters

- **root_objects** (*list*[*Any*])
- **node_association** (*dict*[*Node*, *list*[*Any*]])
- **problems** (*dict*[*Node* / *None*, *list*[*Exception*]])

property `root_objects`: `list[Any]`

objects_for_node(*node*)

Parameters

node ([Node](#))

Return type

`list[Any] | None`

erroneous_nodes()

Return type

`list[Node]`

errors_for_node(*node*)

Parameters

node (*Any*)

Return type

`list[Exception]`

`annize.project.materializer.materialize(project, *, feature_loader=None)`

Parameters

- **project** ([ProjectNode](#))
- **feature_loader** ([FeatureLoader](#) | *None*)

Return type

[MaterializationResult](#)

`annize.project.materializer._translate_from_clone(real_nodes_for_clones, node_association, errors)`

`annize.project.materializer._node_clone_link(original_project_node, cloned_project_node)`

Parameters

- **original_project_node** ([ProjectNode](#))
- **cloned_project_node** ([ProjectNode](#))

Return type

`dict[Node, Node]`

Subpackages

`annize.project.materializer.behaviors` package

Behaviors.

See [Behavior](#).

class `annize.project.materializer.behaviors.Behavior`

Bases: `ABC`

A behavior implements what the materializer does for a given node. See subclasses in the submodules.

early_node_context(*early_node_materialization*)

Context for early materialization steps. It works similar to [node_context\(\)](#), but the early contexts get entered before the main work (of [node_context\(\)](#)) begins, and get left afterward. These early contexts are only used for very particular preparation steps. They do not participate in the actual materialization.

Parameters

early_node_materialization ([EarlyNodeMaterialization](#))

Return type

ContextManager

abstractmethod node_context(*node_materialization, desperate*)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** ([NodeMaterialization](#)) – The node materialization for the current node.
- **desperate** (*bool*) – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

Return type

ContextManager

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

Submodules

annize.project.materializer.behaviors.argument module

See [ArgumentBehavior](#) and [AssociateArgumentNodeBehavior](#).

class annize.project.materializer.behaviors.argument.**ArgumentBehavior**(*create_object_func, *, feature_loader*)

Bases: [Behavior](#)

Behavior that handles argument nodes (incl. creation of an object for an object node).

Parameters

feature_loader ([FeatureLoader](#))

node_context(*node_materialization, desperate*)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** – The node materialization for the current node.
- **desperate** – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

`_abc_impl = <_abc._abc_data object>`

class annize.project.materializer.behaviors.argument.**AssociateArgumentNodeBehavior**(*association*)

Bases: [*Behavior*](#)

Parameters

association (*dict*[[*ArgumentNode*](#), *list*[*Any*]])

node_context(*node_materialization*, *desperate*)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** – The node materialization for the current node.
- **desperate** – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

`_abc_impl = <_abc._abc_data object>`

annize.project.materializer.behaviors.basket module

See [*BasketBehavior*](#).

class annize.project.materializer.behaviors.basket.**BasketBehavior**

Bases: [*Behavior*](#)

Behavior that handles baskets.

node_context(*node_materialization*, *desperate*)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** – The node materialization for the current node.
- **desperate** – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

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

annize.project.materializer.behaviors.block module

See [BlockBehavior](#).

class annize.project.materializer.behaviors.block.**BlockBehavior**

Bases: [Behavior](#)

Behavior that handles block.

early_node_context(*early_node_materialization*)

Context for early materialization steps. It works similar to [node_context\(\)](#), but the early contexts get entered before the main work (of [node_context\(\)](#)) begins, and get left afterward. These early contexts are only used for very particular preparation steps. They do not participate in the actual materialization.

node_context(*node_materialization, desperate*)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** – The node materialization for the current node.
- **desperate** – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

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

annize.project.materializer.behaviors.feature_unavailable module

See [FeatureUnavailableBehavior](#).

class

annize.project.materializer.behaviors.feature_unavailable.**FeatureUnavailableBehavior**

Bases: [Behavior](#)

Behavior that handles ignore-unavailable-Feature nodes.

early_node_context(*early_node_materialization*)

Context for early materialization steps. It works similar to [node_context\(\)](#), but the early contexts get entered before the main work (of [node_context\(\)](#)) begins, and get left afterward. These early contexts are only used for very particular preparation steps. They do not participate in the actual materialization.

node_context(*node_materialization, desperate*)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** – The node materialization for the current node.
- **desperate** – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

`__context_skip_node_feature_ignore_list(node)`

`__context_catch_exceptions(node_materialization, featureignorelist)`

`_abc_impl = <_abc._abc_data object>`

annize.project.materializer.behaviors.reference module

See [ReferenceBehavior](#).

class annize.project.materializer.behaviors.reference.**ReferenceBehavior**

Bases: [Behavior](#)

Behavior that handles reference nodes.

node_context(node_materialization, desperate)

For a node, the materializer will enter the context returned by this function for all behaviors.

The materializer itself does that for the root node. Behaviors itself are responsible for triggering that same process on children.

So, any node gets materialized in the context of all behaviors on all parent nodes. Actual materialization logic happens in this function, in the course of setting up and taking down all these contexts.

Note: Behaviors might fail in some situation, e.g. if a reference is not resolvable yet. The outer routine will retry the materialization process until all behaviors finally succeed or some errors persist.

Parameters

- **node_materialization** – The node materialization for the current node.
- **desperate** – Whether this is a desperate (i.e. late) attempt to materialize, so it is e.g. allowed to consider an unresolvable reference as finally unresolvable.

`_abc_impl = <_abc._abc_data object>`

Submodules

annize.project.materializer.core module

Inner core parts of the project materializer. Only used internally by the parent package.

class annize.project.materializer.core.**EarlyNodeMaterialization**(materializer, node, store)

Bases: object

Parameters

- **materializer** ([ProjectMaterializer](#))
- **node** ([Node](#))
- **store** (*dict*)

property node: [Node](#)

early_materialize_children()

class annize.project.materializer.core.**NodeMaterialization**(*materializer, node, store*)

Bases: object

Parameters

- **materializer** ([ProjectMaterializer](#))
- **node** ([Node](#))
- **store** (*dict*)

property node: [Node](#)

set_materialized_result(*result*)

Parameters

result (*Iterable[Any]*)

Return type

None

set_problems(*problems*)

Parameters

problems (*Iterable[Exception]*)

materialized_children_tuples(*, *desperate*)

Parameters

desperate (*bool*)

materialized_children(*, *desperate*)

Parameters

desperate (*bool*)

Return type

Iterable[Any]

try_get_materialization_for_node(*node*)

Parameters

node ([Node](#))

property has_result

property result: *Sequence[Any]*

property problems: *Sequence[Exception]*

class annize.project.materializer.core.**ProjectMaterializer**(*node, *, behaviors*)

Bases: object

Parameters

- **node** ([Node](#))
- **behaviors** (*Iterable[annize.project.materializer.behaviors.Behavior]*)

materialize()

Return type

`tuple[Sequence[Any] | None, dict[Node, Sequence[Exception]]]`

_early_materialize(*node*, *early_materialization_store*)

_materialize_hlp_childobjs(*node*, *materialization_store*, *desperate*)

Parameters

- **node** (`Node`)
- **materialization_store** (`dict`)
- **desperate** (`bool`)

Return type

`list[tuple[Node, Sequence[Any]]]`

__early_materialization_for_node(*node*, *early_materialization_store*)

Parameters

- **node** (`Node`)
- **early_materialization_store** (`dict`)

Return type

`EarlyNodeMaterialization`

__materialization_for_node(*node*, *materialization_store*)

Parameters

- **node** (`Node`)
- **materialization_store** (`dict`)

Return type

`NodeMaterialization`

__materialize(*node*, *materialization_store*, *desperate*)

Parameters

- **node** (`Node`)
- **materialization_store** (`dict`)
- **desperate** (`bool`)

Return type

`None`

__erroneous_nodes(*old_erroneous_nodes*)

exception `annize.project.materializer.core.InternalError`

Bases: `Exception`

exception `annize.project.materializer.core.ChildrenNotMaterializableError`(*node*)

Bases: `InternalError`

Parameters

node (`Node`)

annize.project.materializer.object_factory module

Creation of objects. See [create_object\(\)](#).

class annize.project.materializer.object_factory._CreateObjectHelper

Bases: object

static [create_object](#)(*of_type*, *args*, *kwargs*)

Parameters

- **of_type** (*type*)
- **args** (*Iterable*)
- **kwargs** (*dict*)

static [_CreateObjectHelper__fill_empty_lists](#)(*parameter_info*, *args*, *kwargs*)

static [_CreateObjectHelper__fill_unspecified_optionals](#)(*parameter_info*, *args*, *kwargs*)

static [_CreateObjectHelper__put_item_into_kwargs](#)(*arg*, *kwargs*, *kwarg_name*, *param_type_info*)

static [_CreateObjectHelper__shift_args_to_kwargs](#)(*of_type*, *inspector*, *args*, *kwargs*)

annize.project.materializer.object_factory.[create_object](#)(*of_type*, *args*, *kwargs*)

Parameters

- **of_type** (*type*)
- **args** (*Iterable*)
- **kwargs** (*dict*)

exception annize.project.materializer.object_factory.[MultipleValuesForSingleArgumentError](#)(*arg_name*)

Bases: [TypeError](#)

Parameters

arg_name (*str*)

annize.project.materializer.preprocessors module

Some preprocessor functions used by the materializer.

Only used internally by the parent package.

annize.project.materializer.preprocessors.[resolve_appendtonodes](#)(*topnode*)

Parameters

topnode ([Node](#))

Return type

[Node](#)

Submodules

annize.project.feature_loader module

Feature module loader.

See [FeatureLoader](#).

class annize.project.feature_loader.**FeatureLoader**

Bases: ABC

Base class for a Feature module loader.

abstractmethod **load_feature**(*name*)

Parameters

name (*str*)

Return type

Any | None

abstractmethod **all_available_feature_names**()

Return type

list[str]

_abc_impl = <_abc._abc_data object>

class annize.project.feature_loader.**DefaultFeatureLoader**

Bases: [FeatureLoader](#)

Default Feature module loader.

_FEATURES_NAMESPACE = 'annize.features'

_COMMON_NAMESPACE_POSTFIX = 'common'

load_feature(*name*)

all_available_feature_names()

__find_feature_modules_in_package(*package_name*)

Parameters

package_name (*str*)

Return type

list[str]

_abc_impl = <_abc._abc_data object>

annize.project.inspector module

Project inspector.

See [FullInspector](#).

class annize.project.inspector.**BasicInspector**

Bases: object

Project inspectors are used in order to get various additional metadata about parts of a project, which are needed e.g. for project parsing and materialization or project configuration UIs.

This inspector type has restricted functionality, but has no dependencies and so is simple to instantiate. See also [FullInspector](#).

type_info(*for_type*)

Return basic type information for a given type, e.g. whether it is a scalar or list and whether it is optional.

Parameters

for_type (*type*) – The type to gather information for.

Return type`TypeInfo`**parameter_info**(*for_type*)

For a given type, return a mapping with type information for each of its constructor's keyword parameters. If it has a parameter for variable keyword arguments, it assumes that these refer to parameters of a superclass constructor and inspects them as well.

It will always contain each keyword parameter, even the ones without type annotation.

For some special types, like `enum.Enum` subclasses, it returns a different mapping, which the materializer will understand when instantiating these objects!

Parameters

for_type (*Callable*) – The type to gather constructor parameter information for.

Return type`Mapping[str, TypeInfo]`**all_named_nodes**(*root_node*)

Return all nodes that have a name assigned in a tree of nodes given by its root node.

Parameters

root_node (*Node*) – The root node. For the entire project, use its project node.

Return type`Sequence[ArgumentNode]`**node_by_name**(*name*, *root_node*)

Return the node with the given name in a tree of nodes given by its root node (or none).

Parameters

- **name** (*str*) – The node name.
- **root_node** (*Node*) – The root node. For the entire project, use its project node.

Return type`Node | None`**resolve_reference_node**(*node*, *, *deep=True*)

For a given argument node, resolve it if it is a reference node, or return the node itself otherwise. Return `None` if it cannot be resolved.

Parameters

- **node** (*ArgumentNode*) – The node to resolve.
- **deep** (*bool*) – Whether to deeply resolve it until a non-reference node is reached (instead of only resolving a single step at most).

Return type`ArgumentNode | None`**possible_argument_names_for_child_in_parent**(*child_type*, *parent_type*)

Return the list of possible argument names that a child with a given type can have in a parent with a given type, according to the parent's `parameter_info()`.

In the context of Annize objects, this list is only useful for children without an `arg_name`, in order to determine it automatically. The first argument name in that list is the preferred one by convention (this is e.g. what the project materializer does). The list will also never contain argument names that are marked as “explicit only” in the parent implementation.

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

Parameters

- **child_type** (*type*) – The child type.
- **parent_type** (*type*) – The parent type.

Return type

Sequence[*str*]

possible_argument_infos_for_child_in_parent(*child_type*, *parent_type*)

Similar to [`possible_argument_names_for_child_in_parent\(\)`](#), but also returns the type info for each possible argument name.

Parameters

- **child_type** (*type*) – The child type.
- **parent_type** (*type*) – The parent type.

Return type

Sequence[*tuple*[*str*, *TypeInfo*]]

type_documentation(*type_*, *, *with_parameters=False*)

Return documentation text for a given type. Parts of it might be in the current i18n culture, but most of it will be in English or whatever language was used in the docstrings.

Parameters

- **type** – The type.
- **with_parameters** (*bool*) – Whether to include documentation for its constructor parameters as well.
- **type_** (*type*)

Return type

str

_possible_argument_names_for_child_in_parent(*child_type*, *parent_type*, *parent_parameter_info*)

Parameters

- **child_type** (*type*)
- **parent_type** (*type*)
- **parent_parameter_info** (*Mapping*[*str*, *TypeInfo*])

Return type

Sequence[*str*]

__type_info(*for_type*, *as_optional=False*)

Parameters

- **for_type** (*type*)
- **as_optional** (*bool*)

Return type

TypeInfo

```

__type_documentation__summary()

    Parameters
        type_doc (str)

    Return type
        str

__type_documentation__parameter(param_name)

    Parameters
        • type_ (type)
        • param_name (str)

    Return type
        str

__type_documentation__parameter_block(param_name)

    Parameters
        • func_doc (str)
        • param_name (str)

    Return type
        Sequence[str]

__type_documentation__parameter_from_lines()

    Parameters
        param_doc_lines (Sequence[str])

    Return type
        str

class TypeInfo
    Bases: ABC
    abstract property name: str
    abstract property type: type | None
    abstract property is_optional: bool
    abstract property allows_multiple_args: bool
    abstract property inner_type_info: TypeInfo | None
    abstractmethod matches_type(type_)
        Parameters
            type_ (type)
        Return type
            bool
    abstractmethod matches_inner_type(type_)
        Parameters
            type_ (type)
        Return type
            bool

```

```
    _abc_impl = <_abc._abc_data object>

class _ScalarTypeInfo(name, type_, is_optional)
    Bases: TypeInfo
        Parameters
            • name (str)
            • type_ (type | None)
            • is_optional (bool)
        property name
        property type
        matches_type(type_)
        matches_inner_type(type_)
        property is_optional
        property inner_type_info
        property allows_multiple_args
    _abc_impl = <_abc._abc_data object>

class _ListTypeInfo(name, is_optional, inner_type_info)
    Bases: \_ScalarTypeInfo
        Parameters
            • name (str)
            • is_optional (bool)
        property allows_multiple_args
        property inner_type_info
    _abc_impl = <_abc._abc_data object>

class _UnionTypeInfo(name, is_optional, union_member_type_infos)
    Bases: \_ScalarTypeInfo
        Parameters
            • name (str)
            • is_optional (bool)
        matches_type(type_)
    _abc_impl = <_abc._abc_data object>

class annize.project.inspector.FullInspector(*, feature_loader=None)
    Bases: BasicInspector

    Project inspectors are used in order to get various additional metadata about parts of a project, which are needed
    e.g. for project parsing and materialization or project configuration UIs.

    This inspector type has full functionality, but has dependencies. See also BasicInspector.
```

Parameters

feature_loader ([FeatureLoader](#) / *None*) – The custom feature loader to use.

match_arguments(*node*)

For a given node, determine for each child node to which argument it matches, and return these argument matchings (taking care of type annotations and arguments' `arg_name`).

Whenever a child cannot be mapped to a particular argument, it is mapped to the "" argument. Whenever more than one argument name would be possible, the first one is taken.

Parameters

node ([Node](#)) – The node to check.

Return type

ArgumentMatchings

argument_type_for_argument_node(*node*)

For a given argument node, return its argument type, or *None* if it was unavailable. For reference nodes, it will resolve the reference and return *None* if it was unresolvable.

Parameters

node ([ArgumentNode](#)) – The argument node.

Return type

type | *None*

creatable_type_info(*for_type*)

Return creatable type information for a given type. This includes the functionality of `type_info()`, but it also includes information for turning it into an argument node.

Parameters

for_type (*type*) – The type to gather information for.

Return type

CreatableTypeInfo

creatables_for_node_argument(*node*, *parameter_name*)

For a given node and parameter name, return a list of all creatable infos that would be valid for this parameter.

Parameters

- **node** ([Node](#)) – The node.
- **parameter_name** (*str*) – The parameter name.

Return type

Sequence[[CreatableInfo](#)]

creatable_types_for_node_argument(*node*, *parameter_name*)

For a given node and parameter name, return a list of all creatable type infos that would be valid for this parameter.

This only returns a list of actual types. You probably should use `creatables_for_node_argument()` instead.

Parameters

- **node** ([Node](#)) – The node.
- **parameter_name** (*str*) – The parameter name.

Return type

Sequence[[CreatableTypeInfo](#)]

possible_reference_targets_for_node_argument(*node*, *parameter_name*)

For a given node and parameter name, return a list of all named nodes (so they can be referenced) that would be valid arguments for this parameter.

Parameters

- **node** ([Node](#)) – The node.
- **parameter_name** (*str*) – The parameter name.

Return type

Sequence[[ArgumentNode](#)]

possible_append_to_targets_for_node(*node*)

For a given node, return a list of all named nodes that would be valid `append_to` targets.

Parameters

node ([ArgumentNode](#)) – The node.

Return type

Sequence[[ArgumentNode](#)]

__all_creatable_types(*, *with_value_types=True*)

Parameters

with_value_types (*bool*)

Return type

Sequence[[CreatableTypeInfo](#)]

__type_full_name()

Parameters

for_type (*type*)

Return type

str

class ArgumentMatching(*arg_name*, *nodes*, *allows_multiple_args*)

Bases: `object`

Parameters

- **arg_name** (*str*)
- **nodes** (*Iterable*[[Node](#)])
- **allows_multiple_args** (*bool*)

property `arg_name`: *str*

property `allows_multiple_args`: *bool*

property `nodes`: *Sequence*[[Node](#)]

class ArgumentMatchings(*all_matchings*)

Bases: `object`

Parameters

all_matchings (*Iterable*[[FullInspector.ArgumentMatching](#)])

```

matching_by_arg_name(arg_name)
    Parameters
        arg_name (str)
    Return type
        ArgumentMatching | None

all()
    Return type
        Sequence[ArgumentMatching]

class CreatableTypeInfo(name, type_, is_optional, feature_name, type_short_name)
    Bases: \_ScalarTypeInfo

    Parameters
        • feature_name (str | None)
        • type_short_name (str)

    property feature_name: str | None
    property type_short_name: str

    _abc_impl = <_abc._abc_data object>

class CreatableInfo(type_info, name, kwargs)
    Bases: object

    Parameters
        • type_info (FullInspector.CreatableTypeInfo)
        • name (str)

    property type_info: CreatableTypeInfo
    property name: str
    property kwargs

```

annize.project.loader module

Loading Annize projects from disk.

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

annize.project.loader.load_project(project_location, *, inspector=None)

Load a project from disk. Return None if the given path does not lead to a location inside an Annize project.

Do not use it directly. See [annize.project.load\(\)](#).

Parameters

- **project_location** (str | Path) – A path to somewhere inside an Annize project.
- **inspector** ([FullInspector](#) | None) – The custom project inspector to use.

Return type

[ProjectNode](#) | None

`annize.project.loader.project_annize_config_main_file(project_location)`

Return the main configuration file for an Annize project given by a path (the path may point to somewhere inside the project; not only inside the Annize configuration directory), or `None` if the given path does not lead to a location inside an Annize project.

This is a file with a name like `project.xml`.

Parameters

project_location (*str* / *Path*) – A location somewhere inside the Annize project.

Return type

Path | `None`

`annize.project.loader.project_annize_config_directory(project_location)`

Return the configuration directory for an Annize project given by a path (the path may point to somewhere inside the project; not only inside the Annize configuration directory), or `None` if the given path does not lead to a location inside an Annize project.

This is a directory with a name like `-meta` (or another name in `ANNIZE_CONFIGURATION_DIRECTORY_NAMES`).

Parameters

project_location (*str* / *Path*) – A location somewhere inside the Annize project.

Return type

Path | `None`

`annize.project.loader.project_root_directory(project_location)`

Return the project root directory for an Annize project given by a path (the path may point to somewhere inside the project; not only inside the Annize configuration directory), or `None` if the given path does not lead to a location inside an Annize project.

This is a directory with a subdirectory like `-meta` (or another name in `ANNIZE_CONFIGURATION_DIRECTORY_NAMES`).

Parameters

project_location (*str* / *Path*) – A location somewhere inside the Annize project.

Return type

Path | `None`

`annize.project.loader.is_valid_annize_configuration_file_name(name)`

Return whether a given name is a valid Annize configuration file name.

Parameters

name (*str*) – The file name to check.

Return type

`bool`

annize.ui package

`annize.ui.app(app_name, **kwargs)`

Parameters

app_name (*str*)

Subpackages

annize.ui.apps package

Subpackages

annize.ui.apps.runner package

Subpackages

annize.ui.apps.runner.models package

Submodules

annize.ui.apps.runner.models.main module

annize.ui.apps.runner.models.task_chooser module

annize.ui.apps.runner.models.task_execution module

annize.ui.apps.runner.models.user_feedback module

annize.ui.apps.runner.views package

Submodules

annize.ui.apps.runner.views.main module

annize.ui.apps.runner.views.task_chooser module

annize.ui.apps.runner.views.task_execution module

annize.ui.apps.runner.views.user_feedback module

annize.ui.apps.studio package

Subpackages

annize.ui.apps.studio.models package

Submodules

annize.ui.apps.studio.models.add_child module

annize.ui.apps.studio.models.choose_reference_target module

annize.ui.apps.studio.models.main module

annize.ui.apps.studio.models.main_tab_panel module

annize.ui.apps.studio.models.object_editor module

annize.ui.apps.studio.models.object_help module

annize.ui.apps.studio.models.problems_list module

annize.ui.apps.studio.models.project_config module

annize.ui.apps.studio.views package

Submodules

annize.ui.apps.studio.views.add_child module

annize.ui.apps.studio.views.choose_reference_target module

annize.ui.apps.studio.views.main module

annize.ui.apps.studio.views.main_tab_panel module

annize.ui.apps.studio.views.object_editor module

annize.ui.apps.studio.views.object_help module

annize.ui.apps.studio.views.problems_list module

annize.ui.apps.studio.views.project_config module

annize.user_feedback package

class annize.user_feedback.UserFeedbackController

Bases: ABC

abstractmethod message_dialog(*message, answers, config_key*)

Parameters

- **message** (*str*)
- **answers** (*list[str]*)
- **config_key** (*str | None*)

Return type

int

abstractmethod input_dialog(*question, suggested_answer, config_key*)

Parameters

- **question** (*str*)
- **suggested_answer** (*str*)
- **config_key** (*str | None*)

Return type

str | None

abstractmethod choice_dialog(*question, choices, config_key*)

Parameters

- **question** (*str*)
- **choices** (*list[str]*)
- **config_key** (*str | None*)

Return type

int | None

```

    _abc_impl = <_abc._abc_data object>

class annize.user_feedback.NullUserFeedbackController
    Bases: object
    message_dialog(*)
    input_dialog(*)
    choice_dialog(*)

exception annize.user_feedback.UnsatisfiableUserFeedbackAttemptError
    Bases: RuntimeError

annize.user_feedback._controllers_tuples_for_context(context)

    Parameters
        context (RunContext)

    Return type
        Sequence[tuple[int, UserFeedbackController]]

annize.user_feedback._controllers_for_context(context)

    Parameters
        context (RunContext)

    Return type
        list[UserFeedbackController]

annize.user_feedback._add_controller_to_context(*, controller, context, priority_index=0)

    Parameters
        • controller (UserFeedbackController)
        • context (RunContext)
        • priority_index (int)

    Return type
        None

annize.user_feedback.message_dialog(message, answers, *, config_key=None)

    Parameters
        • message (TrStrOrStr)
        • answers (Iterable[TrStrOrStr])
        • config_key (str | None)

    Return type
        int

annize.user_feedback.input_dialog(message, *, suggested_answer, config_key=None)

    Parameters
        • message (TrStrOrStr)
        • suggested_answer (TrStrOrStr)
        • config_key (str | None)

```

Return type

str | None

`annize.user_feedback.choice_dialog(message, choices, *, config_key=None)`**Parameters**

- **message** (*TrStrOrStr*)
- **choices** (*Iterable[TrStrOrStr]*)
- **config_key** (*str* | *None*)

Return type

int | None

Submodules**annize.user_feedback.static module**`class annize.user_feedback.static.StaticUserFeedbackController(answers)`Bases: *UserFeedbackController***Parameters****answers** (*dict[str, Any]*)`add_answer(config_key, value)`**Parameters**

- **config_key** (*str*)
- **value** (*Any*)

Return type

None

`__get_answer(config_key)`**Parameters****config_key** (*str*)**Return type***Any*`message_dialog(message, answers, config_key)``input_dialog(question, suggested_answer, config_key)``choice_dialog(question, choices, config_key)``_abc_impl = <_abc._abc_data object>`**6.1.2 Submodules****6.1.3 annize.annize_cli module**

The Annize CLI.

`annize.annize_cli.main()`

`annize.annize_cli.parser(*, only_documentation=True)`

Parameters

`only_documentation` (*bool*)

Return type

ArgumentParser

class `annize.annize_cli.Commands`(*project, with_answers_from_json_file, with_answers_from_json_string, with_answer, **_*)

Bases: `object`

Parameters

- `project` (*str*)
- `with_answers_from_json_file` (*Iterable[str]*)
- `with_answers_from_json_string` (*Iterable[str]*)
- `with_answer` (*Iterable[Tuple[str, str]]*)

`__initial_cwd` = `'/home/pino/projects/annize'`

classmethod `__answers_from_json_files`(*destination, with_answers_from_json_files*)

Parameters

- `destination` (*dict*)
- `with_answers_from_json_files` (*Iterable[str]*)

classmethod `__answers_from_json_strings`(*destination, with_answers_from_json_strings*)

Parameters

- `destination` (*dict*)
- `with_answers_from_json_strings` (*Iterable[str]*)

classmethod `__answers_from_single_answers`(*destination, with_answers*)

Parameters

- `destination` (*dict*)
- `with_answers` (*Iterable[Tuple[str, str]]*)

do(*task_name, **_*)

Parameters

`task_name` (*str*)

studio(***_*)

`annize.annize_cli._setup_logging(*, debug=False)`

Parameters

`debug` (*bool*)

PYTHON MODULE INDEX

a

- annize, 15
- annize.annize_cli, 84
- annize.asset, 15
- annize.asset.data, 15
- annize.asset.project_info, 15
- annize.data, 15
- annize.data.color, 15
- annize.data.container, 17
- annize.data.version, 17
- annize.features, 21
- annize.features.base, 26
- annize.features.files, 21
- annize.features.files.common, 21
- annize.features.i18n, 24
- annize.features.i18n.common, 24
- annize.features.i18n.gettext, 26
- annize.features.task, 28
- annize.flow, 28
- annize.flow.run_context, 28
- annize.flow.runner, 34
- annize.fs, 35
- annize.fs.ext, 40
- annize.i18n, 41
- annize.object, 49
- annize.object.config, 49
- annize.project, 50
- annize.project.feature_loader, 71
- annize.project.file_formats, 60
- annize.project.inspector, 72
- annize.project.loader, 79
- annize.project.materializer, 63
- annize.project.materializer.behaviors, 64
- annize.project.materializer.behaviors.argument, 65
- annize.project.materializer.behaviors.basket, 66
- annize.project.materializer.behaviors.block, 67
- annize.project.materializer.behaviors.feature_unavailable, 67
- annize.project.materializer.behaviors.reference, 68
- annize.project.materializer.core, 68
- annize.project.materializer.object_factory, 71
- annize.project.materializer.preprocessors, 71
- annize.ui, 80
- annize.ui.apps, 81
- annize.user_feedback, 82
- annize.user_feedback.static, 84

INDEX

Symbols

<code>_COMMON_NAMESPACE_POSTFIX</code>	(<i>annize.project.feature_loader.DefaultFeatureLoader</i> attribute), 72	<code>__answers_from_json_files()</code>	(<i>annize.annize_cli.Commands</i> class method), 85
<code>_CreateObjectHelper</code>	(class in <i>annize.project.materializer.object_factory</i>), 71	<code>__answers_from_json_strings()</code>	(<i>annize.annize_cli.Commands</i> class method), 85
<code>_CreateObjectHelper__fill_empty_lists()</code>	(<i>annize.project.materializer.object_factory._CreateObjectHelper</i> static method), 71	<code>__answers_from_single_answers()</code>	(<i>annize.annize_cli.Commands</i> class method), 85
<code>_CreateObjectHelper__fill_unspecified_optionals()</code>	(<i>annize.project.materializer.object_factory._CreateObjectHelper</i> static method), 71	<code>__best_system_locale()</code>	(<i>annize.i18n.Culture</i> method), 45
<code>_CreateObjectHelper__put_item_into_kwargs()</code>	(<i>annize.project.materializer.object_factory._CreateObjectHelper</i> static method), 71	<code>__changed__call__handlers()</code>	(<i>annize.project.Node</i> method), 54
<code>_CreateObjectHelper__shift_args_to_kwargs()</code>	(<i>annize.project.materializer.object_factory._CreateObjectHelper</i> static method), 71	<code>__changed__child_added()</code>	(<i>annize.project.Node</i> method), 53
<code>_FEATURES_NAMESPACE</code>	(<i>annize.project.feature_loader.DefaultFeatureLoader</i> attribute), 72	<code>__changed__child_removed()</code>	(<i>annize.project.Node</i> method), 53
<code>_FixedTrStr</code>	(class in <i>annize.i18n</i>), 48	<code>__changed__property_changed()</code>	(<i>annize.project.Node</i> method), 53
<code>_FormattedTrStr</code>	(class in <i>annize.i18n</i>), 49	<code>__cleanup()</code>	(<i>annize.fs.ext.FreshTempDirectory</i> method), 40
<code>_IS_TOPLEVEL_OBJECT__METADATA_KEY</code>	(<i>annize.flow.run_context.RunContext</i> attribute), 28	<code>__compacted_changes()</code>	(<i>annize.project.ProjectNode</i> method), 57
<code>_NAMES__METADATA_KEY</code>	(<i>annize.flow.run_context.RunContext</i> attribute), 28	<code>__context_catch_exceptions()</code>	(<i>annize.project.materializer.behaviors.feature_unavailable.FeatureUnavailable</i> method), 68
<code>_ProjectDefinedTranslationProvider</code>	(class in <i>annize.features.i18n.common</i>), 24	<code>__context_skip_node_feature_ignore_list()</code>	(<i>annize.project.materializer.behaviors.feature_unavailable.FeatureUnavailable</i> method), 68
<code>_ProjectDefinedTranslationProvider__translations_for_string_name()</code>	(<i>annize.features.i18n.common._ProjectDefinedTranslationProvider</i> method), 25	<code>__current_system_locale_setup()</code>	(<i>annize.i18n.Culture</i> method), 45
<code>_TContent</code>	(<i>annize.fs.ext.DynamicFile</i> attribute), 40	<code>__description()</code>	(<i>annize.project.Node</i> method), 53
<code>_TStaticContent</code>	(<i>annize.fs.ext.DynamicFile</i> attribute), 40	<code>__do_run()</code>	(<i>annize.flow.runner.Runner</i> method), 35
<code>_TransferHelper__transfer_piece()</code>	(<i>annize.fs.Path._TransferHelper</i> static method), 39	<code>__does_exclude()</code>	(<i>annize.features.files.common.Exclude</i> method), 22
<code>__all_creatable_types()</code>	(<i>annize.project.inspector.FullInspector</i> method), 78	<code>__early_materialization_for_node()</code>	(<i>annize.project.materializer.core.ProjectMaterializer</i> method), 70
		<code>__erroneous_nodes()</code>	(<i>annize.project.materializer.core.ProjectMaterializer</i> method), 70

<i>nize.project.materializer.core.ProjectMaterializer</i> method), 70	<i>nize.project.inspector.BasicInspector</i> method), 74
<code>__find_feature_modules_in_package()</code> (<i>annize.project.feature_loader.DefaultFeatureLoader</i> method), 72	<code>__type_full_name()</code> (<i>annize.project.inspector.FullInspector</i> method), 78
<code>__get_answer()</code> (<i>annize.user_feedback.static.StaticUserFeedbackInfo</i>) (<i>annize.project.inspector.BasicInspector</i> method), 84	<code>__type_info()</code> (<i>annize.project.inspector.BasicInspector</i> method), 74
<code>__get_normed_value()</code> (<i>annize.data.color.Color</i> method), 16	<code>_abc_impl</code> (<i>annize.data.version.ConcatenatedVersionPatternPart</i> attribute), 20
<code>__handle_changed()</code> (<i>annize.project.ProjectNode</i> method), 57	<code>_abc_impl</code> (<i>annize.data.version.NumericVersionPatternPart</i> attribute), 19
<code>__html_color_spec__part()</code> (<i>annize.data.color.Color</i> method), 16	<code>_abc_impl</code> (<i>annize.data.version.OptionalVersionPatternPart</i> attribute), 20
<code>__initial_cwd</code> (<i>annize.annize_cli.Commands</i> attribute), 85	<code>_abc_impl</code> (<i>annize.data.version.SeparatorVersionPatternPart</i> attribute), 19
<code>__is_inverse_of()</code> (<i>annize.project.ProjectNode</i> method), 57	<code>_abc_impl</code> (<i>annize.data.version.VersionPatternPart</i> attribute), 18
<code>__materialization_for_node()</code> (<i>annize.project.materializer.core.ProjectMaterializer</i> method), 70	<code>_abc_impl</code> (<i>annize.features.i18n.common.String</i> attribute), 25
<code>__materialize()</code> (<i>annize.project.materializer.core.ProjectMaterializer</i> method), 70	<code>_abc_impl</code> (<i>annize.features.i18n.common._ProjectDefinedTranslationProvider</i> attribute), 25
<code>__object_metadata_dict()</code> (<i>annize.flow.run_context.RunContext</i> method), 31	<code>_abc_impl</code> (<i>annize.flow.runner.Runner</i> attribute), 35
<code>__object_raw_name()</code> (<i>annize.flow.run_context.RunContext</i> method), 31	<code>_abc_impl</code> (<i>annize.i18n.GettextTranslationProvider</i> attribute), 47
<code>__put_object()</code> (<i>annize.flow.run_context.RunContext</i> method), 31	<code>_abc_impl</code> (<i>annize.i18n.ProvidedTrStr</i> attribute), 47
<code>__reset_change_history()</code> (<i>annize.project.ProjectNode</i> method), 57	<code>_abc_impl</code> (<i>annize.i18n.TrStr</i> attribute), 43
<code>__set_env__var()</code> (<i>annize.i18n.Culture</i> method), 45	<code>_abc_impl</code> (<i>annize.i18n.TranslationProvider</i> attribute), 43
<code>__set_success_state()</code> (<i>annize.flow.runner.Runner</i> method), 35	<code>_abc_impl</code> (<i>annize.i18n._FixedTrStr</i> attribute), 48
<code>__set_system_locale_setup()</code> (<i>annize.i18n.Culture</i> method), 45	<code>_abc_impl</code> (<i>annize.i18n._FormattedTrStr</i> attribute), 49
<code>__set_tasks()</code> (<i>annize.flow.runner.Runner</i> method), 35	<code>_abc_impl</code> (<i>annize.project.ArgumentNode</i> attribute), 58
<code>__shorten()</code> (<i>annize.project.ScalarValueNode</i> method), 59	<code>_abc_impl</code> (<i>annize.project.FileNode</i> attribute), 58
<code>__transfer_filter_for_exclude()</code> (<i>annize.features.files.common.Directory</i> method), 23	<code>_abc_impl</code> (<i>annize.project.IgnoreUnavailableFeatureNode</i> attribute), 60
<code>__type_documentation__parameter()</code> (<i>annize.project.inspector.BasicInspector</i> method), 75	<code>_abc_impl</code> (<i>annize.project.Node</i> attribute), 55
<code>__type_documentation__parameter_block()</code> (<i>annize.project.inspector.BasicInspector</i> method), 75	<code>_abc_impl</code> (<i>annize.project.ObjectNode</i> attribute), 58
<code>__type_documentation__parameter_from_lines()</code> (<i>annize.project.inspector.BasicInspector</i> method), 75	<code>_abc_impl</code> (<i>annize.project.ProjectNode</i> attribute), 57
<code>__type_documentation__summary()</code> (<i>annize.project.inspector.BasicInspector</i> method), 75	<code>_abc_impl</code> (<i>annize.project.ReferenceNode</i> attribute), 59
	<code>_abc_impl</code> (<i>annize.project.ScalarValueNode</i> attribute), 59
	<code>_abc_impl</code> (<i>annize.project.feature_loader.DefaultFeatureLoader</i> attribute), 72
	<code>_abc_impl</code> (<i>annize.project.feature_loader.FeatureLoader</i> attribute), 72
	<code>_abc_impl</code> (<i>annize.project.file_formats.FileFormat</i> attribute), 62
	<code>_abc_impl</code> (<i>annize.project.file_formats.FileFormat.Marshaler</i> attribute), 61
	<code>_abc_impl</code> (<i>annize.project.file_formats.FileFormat.NullMarshaler</i> attribute), 61
	<code>_abc_impl</code> (<i>annize.project.inspector.BasicInspector.TypeInfo</i> attribute), 75
	<code>_abc_impl</code> (<i>annize.project.inspector.BasicInspector._ListTypeInfo</i> attribute), 76

_abc_impl(annize.project.inspector.BasicInspector._ScalarTypeInfol-
 lers_tuples_for_context() (in module
 annize.user_feedback), 83
 _abc_impl(annize.project.inspector.BasicInspector._UnionTypeInfor-
 mation_translation_providers_lists() (in
 module annize.i18n), 48
 _abc_impl(annize.project.inspector.FullInspector.CreatableEarlylyma-
 terialize() (an-
 nize.project.materializer.core.ProjectMaterializer
 method), 70
 _abc_impl(annize.project.materializer.behaviors.Behavior
 attribute), 65
 _abc_impl(annize.project.materializer.behaviors.argument.AssumeSoftlycul-
 ture (in module annize.i18n), 47
 _abc_impl(annize.project.materializer.behaviors.argument.AssociateArgu-
 mentNodeBehavior (an-
 nize.project.materializer.core.ProjectMaterializer
 method), 70
 _abc_impl(annize.project.materializer.behaviors.basket.BasketRelation-
 link() (in module an-
 nize.project.materializer), 64
 _abc_impl(annize.project.materializer.behaviors.block.BlockBehavior
 (annize.features.files.common.Directory
 method), 23
 _abc_impl(annize.project.materializer.behaviors.feature_unavailablefea-
 ture_unavailable_behavior (annize.features.files.common.FsEntry method),
 21
 _abc_impl(annize.project.materializer.behaviors.reference.ReferenceBehav-
 ior (annize.features.files.common.MachineRootDirectory
 method), 24
 _abc_impl(annize.user_feedback.UserFeedbackController_path() (annize.features.files.common.ProjectDirectory
 method), 24
 _abc_impl(annize.user_feedback.static.StaticUserFeedbackControllerPath() (annize.js.Path method), 36
 _abc_impl(annize.user_feedback.static.StaticUserFeedbackControllerPath() (annize.js.ext.DynamicFile method), 41
 _add_controller_to_context() (in module an-
 nize.user_feedback), 83
 _allowed_child_types() (annize.project.FileNode
 class method), 57
 _allowed_child_types() (an-
 nize.project.IgnoreUnavailableFeatureNode
 class method), 60
 _allowed_child_types() (annize.project.Node class
 method), 52
 _allowed_child_types() (annize.project.ObjectNode
 class method), 58
 _allowed_child_types() (annize.project.ProjectNode
 class method), 56
 _allowed_child_types() (an-
 nize.project.ReferenceNode class method),
 59
 _allowed_child_types() (an-
 nize.project.ScalarValueNode class method),
 59
 _annize_user_interaction_culture() (in module
 annize.i18n), 48
 _clone__early() (annize.project.FileNode method), 57
 _clone__early() (annize.project.Node method), 52
 _clone__early() (annize.project.ProjectNode
 method), 56
 _clone__late() (annize.project.Node method), 52
 _clone__late() (annize.project.ProjectNode method),
 56
 _controllers_for_context() (in module an-
 nize.user_feedback), 83
 _translate_from_clone() (in module an-
 nize.project.materializer), 64
 _translation_for_culture() (an-
 nize.i18n.ProvidedTrStr method), 47
 _translation_for_culture() (annize.i18n.TrStr
 method), 42
 _translation_for_culture() (an-
 nize.i18n._FixedTrStr method), 48
 _translation_for_culture() (an-
 nize.i18n._FormattedTrStr method), 49
 _translation_provider() (in module an-

nize.features.i18n.common), 25
_translation_providers() (in module *annize.i18n*), 48

A

add_answer() (*annize.user_feedback.static.StaticUserFeedbackHandler* module, 84
method), 84
add_change() (*annize.project.file_formats.FileFormat.Marshaler* module, 17
method), 61
add_change() (*annize.project.file_formats.FileFormat.NullMarshaler* module, 17
method), 61
add_change_handler() (*annize.project.Node* method), 51
add_object() (*annize.flow.run_context.RunContext* method), 30
add_object() (in module *annize.flow.run_context*), 33
add_translation_provider() (in module *annize.i18n*), 43
add_translations() (*annize.features.i18n.common._ProjectDefinedTranslationsProvider* module, 24
method), 24
all() (*annize.project.inspector.FullInspector.ArgumentMatching* module, 24
method), 79
all_available_feature_names() (*annize.project.feature_loader.DefaultFeatureLoader* module, 72
method), 72
all_available_feature_names() (*annize.project.feature_loader.FeatureLoader* module, 72
method), 72
all_file_format_names() (in module *annize.project.file_formats*), 62
all_named_nodes() (*annize.project.inspector.BasicInspector* method), 73
allows_multiple_args (*annize.project.inspector.BasicInspector._ListTypeInfo* module, 40
property), 76
allows_multiple_args (*annize.project.inspector.BasicInspector._ScalarTypeInfo* module, 41
property), 76
allows_multiple_args (*annize.project.inspector.BasicInspector.TypeInfo* module, 49
property), 75
allows_multiple_args (*annize.project.inspector.FullInspector.ArgumentMatching* module, 71
property), 78

annize module, 15
annize.annize_cli module, 84
annize.asset module, 15
annize.asset.data module, 15
annize.asset.project_info module, 15
annize.data module, 15
annize.data.color module, 15
annize.data.container module, 17
annize.data.version module, 17
annize.features module, 21
annize.features.base module, 26
annize.features.files module, 21
annize.features.files.common module, 21
annize.features.i18n module, 24
annize.features.i18n.common module, 24
annize.features.i18n.gettext module, 26
annize.features.task module, 28
annize.flow module, 28
annize.flow.run_context module, 28
annize.flow.runner module, 34
annize.fs module, 35
annize.fs.ext module, 40
annize.i18n module, 41
annize.object module, 49
annize.object.config module, 49
annize.project module, 50
annize.project.feature_loader module, 71
annize.project.file_formats module, 60
annize.project.inspector module, 72
annize.project.loader module, 79
annize.project.materializer module, 63

annize.project.materializer.behaviors
 module, 64
 annize.project.materializer.behaviors.argument
 module, 65
 annize.project.materializer.behaviors.basket
 module, 66
 annize.project.materializer.behaviors.block
 module, 67
 annize.project.materializer.behaviors.feature
 module, 67
 annize.project.materializer.behaviors.reference
 module, 68
 annize.project.materializer.core
 module, 68
 annize.project.materializer.object_factory
 module, 71
 annize.project.materializer.preprocessors
 module, 71
 annize.ui
 module, 80
 annize.ui.apps
 module, 81
 annize.user_feedback
 module, 82
 annize.user_feedback.static
 module, 84
 annize_config_directory (an-
 nize.project.ProjectNode property), 55
 ANNIZE_CONFIG_DIRECTORY__NAME (an-
 nize.flow.run_context.RunContext attribute),
 29
 annize_user_interaction_culture (in module an-
 nize.i18n), 49
 app() (in module annize.ui), 80
 append_child() (annize.project.Node method), 52
 append_to (annize.project.ArgumentNode property), 58
 arg_name (annize.project.ArgumentNode property), 58
 arg_name (annize.project.inspector.FullInspector.Argument
 property), 78
 argument_type_for_argument_node() (an-
 nize.project.inspector.FullInspector method),
 77
 ArgumentBehavior (class in an-
 nize.project.materializer.behaviors.argument),
 65
 ArgumentNode (class in annize.project), 58
 AssociateArgumentNodeBehavior (class in an-
 nize.project.materializer.behaviors.argument),
 66

B

BadStructureError, 60
 BasicInspector (class in annize.project.inspector), 72
 _ListTypeInfo (class in an-
 nize.project.inspector), 76
 _ScalarTypeInfo (class in an-
 nize.project.inspector), 76
 _UnionTypeInfo (class in an-
 nize.project.inspector), 76
 TypeInfo (class in an-
 nize.project.inspector), 75
 Basket (class in annize.data.container), 17
 Basket (class in annize.features.base), 27
 BasketBehavior (class in an-
 nize.project.materializer.behaviors.basket),
 66
 Behavior (class in an-
 nize.project.materializer.behaviors), 64
 BlockBehavior (class in an-
 nize.project.materializer.behaviors.block),
 67
 blue (annize.data.color.Color property), 16

C

changes() (annize.project.ProjectNode method), 56
 child_node (annize.project.Node.__ChildrenListChangeEvent
 property), 54
 child_position (annize.project.Node.__ChildrenListChangeEvent
 property), 54
 children (annize.project.Node property), 51
 children() (annize.fs.Path method), 36
 ChildrenNotMaterializableError, 70
 choice_dialog() (an-
 nize.user_feedback.NullUserFeedbackController
 method), 83
 choice_dialog() (an-
 nize.user_feedback.static.StaticUserFeedbackController
 method), 84
 choice_dialog() (an-
 nize.user_feedback.UserFeedbackController
 method), 82
 choice_dialog() (in module annize.user_feedback), 84
 clone() (annize.project.Node method), 52
 Color (class in annize.data.color), 15
 Commands (class in annize.annize_cli), 85
 ConcatenatedVersionPatternPart (class in an-
 nize.data.version), 20
 content() (in module annize.fs), 39
 copy_to() (annize.fs.Path method), 37
 creatable_type_info() (an-
 nize.project.inspector.FullInspector method),
 77
 creatable_types_for_node_argument() (an-
 nize.project.inspector.FullInspector method),
 77
 creatables_for_node_argument() (an-
 nize.project.inspector.FullInspector method),

77
 create_new() (in module *annize.project*), 50
 create_object() (in module *annize.project.materializer.object_factory._CreateObjectHelper*), 71
 create_object() (in module *annize.project.materializer.object_factory*), 71
 ctime() (*annize.fs.Path* method), 36
 Culture (class in *annize.features.i18n.common*), 25
 Culture (class in *annize.i18n*), 44
 Culture._TSystemLocaleSetup (class in *annize.i18n*), 46
 culture_by_spec() (in module *annize.i18n*), 47
 culture_list() (*annize.i18n.Culture* method), 45
 current() (in module *annize.flow.run_context*), 32
 current_culture() (in module *annize.i18n*), 46

D

Data (class in *annize.features.base*), 26
 DateTime (class in *annize.features.base*), 27
 DefaultFeatureLoader (class in *annize.project.feature_loader*), 72
 description() (*annize.project.Node* method), 52
 deserialize_node() (*annize.project.file_formats.FileFormat* method), 62
 destination (*annize.fs.ext.Mount* property), 41
 destination_is_parent (*annize.features.files.common.DirectoryPart* property), 23
 destination_path (*annize.features.files.common.DirectoryPart* property), 23
 Directory (class in *annize.features.files.common*), 23
 DirectoryPart (class in *annize.features.files.common*), 22
 do() (*annize.annize_cli.Commands* method), 85
 does_exclude() (*annize.features.files.common.Exclude* method), 22
 does_exclude() (*annize.features.files.common.ExcludeAllBut* method), 22
 dynamic_file() (in module *annize.fs*), 39
 DynamicFile (class in *annize.fs.ext*), 40

E

early_materialize_children() (*annize.project.materializer.core.EarlyNodeMaterialization* method), 69
 early_node_context() (*annize.project.materializer.behaviors.Behavior* method), 64
 early_node_context() (*annize.project.materializer.behaviors.block.BlockBehavior* method), 67
 early_node_context() (*annize.project.materializer.behaviors.feature_unavailable.FeatureUnavailableBehavior* method), 67
 EarlyNodeMaterialization (class in *annize.project.materializer.core*), 68
 english_lang_name (*annize.i18n.Culture* property), 44
 erroneous_nodes() (*annize.project.materializer.MaterializationResult* method), 64
 errors_for_node() (*annize.project.materializer.MaterializationResult* method), 64
 Exclude (class in *annize.features.files.common*), 21
 ExcludeAllBut (class in *annize.features.files.common*), 22
 excludes (*annize.features.files.common.Directory* property), 23
 excludes (*annize.features.files.common.DirectoryPart* property), 23
 explicit_only (*annize.object.config.InnerParameterConfig* attribute), 50
 explicit_only (*annize.object.config.ParameterConfig* attribute), 50
 explicit_only() (in module *annize.object*), 49

F

FAIL (*annize.project.ReferenceNode.OnUnresolvableAction* attribute), 59
 fallback_cultures (*annize.i18n.Culture* property), 45
 feature (*annize.project.IgnoreUnavailableFeatureNode* property), 59
 feature (*annize.project.ObjectNode* property), 58
 feature_name (*annize.project.inspector.FullInspector.CreatableTypeInfo* property), 79
 FeatureLoader (class in *annize.project.feature_loader*), 71
 FeatureUnavailableBehavior (class in *annize.project.materializer.behaviors.feature_unavailable*), 67
 FeatureUnavailableError, 60
 file (*annize.project.Node* property), 51
 File (class in *annize.features.files.common*), 21
 file_format() (in module *annize.project.file_formats*), 62
 file_size() (*annize.fs.Path* method), 37
 FileFormat (class in *annize.project.file_formats*), 60
 FileFormat.Marshaler (class in *annize.project.file_formats*), 60
 FileFormat.NullMarshaler (class in *annize.project.file_formats*), 61
 FileNode (class in *annize.project*), 57
 FilesystemContent (class in *annize.fs*), 35
 FirstOf (class in *annize.features.base*), 27

format() (*annize.i18n.TrStr method*), 42
 fresh_temp_directory() (*in module annize.fs*), 39
 FreshTempDirectory (*class in annize.fs.ext*), 40
 friendly_join_string_list() (*in module annize.i18n*), 48
 from_iso_639_1_lang_code() (*annize.i18n.Culture static method*), 44
 FsEntry (*class in annize.features.files.common*), 21
 full_name (*annize.i18n.Culture property*), 45
 FullInspector (*class in annize.project.inspector*), 76
 FullInspector.ArgumentMatching (*class in annize.project.inspector*), 78
 FullInspector.ArgumentMatchings (*class in annize.project.inspector*), 78
 FullInspector.CreatableInfo (*class in annize.project.inspector*), 79
 FullInspector.CreatableTypeInfo (*class in annize.project.inspector*), 79

G

GenerateMOs (*class in annize.features.i18n.gettext*), 26
 get_selected_task() (*annize.flow.runner.Runner method*), 34
 get_success_state() (*annize.flow.runner.Runner method*), 35
 get_tasks() (*annize.flow.runner.Runner method*), 34
 GettextTranslationProvider (*class in annize.i18n*), 46
 GettextTranslationProvider._NoneTranslations (*class in annize.i18n*), 47
 green (*annize.data.color.Color property*), 15

H

has_result (*annize.project.materializer.core.NodeMaterialization property*), 69
 homepage_url (*annize.features.base.Data property*), 26
 homepage_url() (*in module annize.features.base*), 27
 html_color_spec (*annize.data.color.Color property*), 16
 hue (*annize.data.color.Color property*), 16

I

IgnoreUnavailableFeatureNode (*class in annize.project*), 59
 inner_type_info (*annize.project.inspector.BasicInspector._ListTypeInfo property*), 76
 inner_type_info (*annize.project.inspector.BasicInspector._ScalarTypeInfo property*), 76
 inner_type_info (*annize.project.inspector.BasicInspector.TypeInfo property*), 75

K

InnerParameterConfig (*class in annize.object.config*), 50
 input_dialog() (*annize.user_feedback.NullUserFeedbackController method*), 83
 input_dialog() (*annize.user_feedback.static.StaticUserFeedbackController method*), 84
 input_dialog() (*annize.user_feedback.UserFeedbackController method*), 82
 input_dialog() (*in module annize.user_feedback*), 83
 insert_child() (*annize.project.Node method*), 51
 insert_child() (*annize.project.ProjectNode method*), 55

L

InternalError, 70
 is_advanced (*annize.features.task.Task property*), 28
 is_finished() (*annize.flow.runner.Runner method*), 34
 is_friendly_name() (*annize.flow.run_context.RunContext method*), 30
 is_friendly_name() (*in module annize.flow.run_context*), 33
 is_optional (*annize.project.inspector.BasicInspector._ScalarTypeInfo property*), 76
 is_optional (*annize.project.inspector.BasicInspector.TypeInfo property*), 75
 is_toplevel_object() (*annize.flow.run_context.RunContext method*), 30
 is_toplevel_object() (*in module annize.flow.run_context*), 33
 is_valid_annize_configuration_file_name() (*in module annize.project.loader*), 80
 iso_639_1_language_code (*annize.i18n.Culture property*), 44

M

kwargs (*annize.project.inspector.FullInspector.CreatableInfo property*), 79

load_project() (in module *annize.project.file_formats*), 62
 load_project() (in module *annize.project.loader*), 79
 long_description (*annize.features.base.Data* property), 26
 long_description() (in module *annize.features.base*), 27

M

MachineRootDirectory (class in *annize.features.files.common*), 24
 main() (in module *annize.annize_cli*), 84
 mark_object_as_toplevel() (in *annize.flow.run_context.RunContext* method), 30
 marshaler (*annize.project.FileNode* property), 57
 match_arguments() (in *annize.project.inspector.FullInspector* method), 77
 matches_inner_type() (in *annize.project.inspector.BasicInspector._ScalarTypeInfo* method), 76
 matches_inner_type() (in *annize.project.inspector.BasicInspector.TypeInfo* method), 75
 matches_type() (*annize.project.inspector.BasicInspector._ScalarTypeInfo* method), 76
 matches_type() (*annize.project.inspector.BasicInspector._UnionTypeInfo* method), 76
 matches_type() (*annize.project.inspector.BasicInspector.TypeInfo* method), 75
 matching_by_arg_name() (in *annize.project.inspector.FullInspector.ArgumentMatching* method), 78
 MaterializationResult (class in *annize.project.materializer*), 63
 materialize() (*annize.project.materializer.core.ProjectMaterializer* method), 69
 materialize() (in module *annize.project.materializer*), 64
 materialized_children() (in *annize.project.materializer.core.NodeMaterialization* method), 69
 materialized_children_tuples() (in *annize.project.materializer.core.NodeMaterialization* method), 69
 MaterializerError, 60
 message_dialog() (in *annize.user_feedback.NullUserFeedbackController* method), 83
 message_dialog() (in *annize.user_feedback.static.StaticUserFeedbackController* method), 84
 message_dialog() (in *annize.user_feedback.UserFeedbackController* method), 82
 message_dialog() (in module *annize.user_feedback*), 83
 module
 annize, 15
 annize.annize_cli, 84
 annize.asset, 15
 annize.asset.data, 15
 annize.asset.project_info, 15
 annize.data, 15
 annize.data.color, 15
 annize.data.container, 17
 annize.data.version, 17
 annize.features, 21
 annize.features.base, 26
 annize.features.files, 21
 annize.features.files.common, 21
 annize.features.i18n, 24
 annize.features.i18n.common, 24
 annize.features.i18n.gettext, 26
 annize.features.task, 28
 annize.flow, 28
 annize.flow.run_context, 28
 annize.flow.runner, 34
 annize.fs, 35
 annize.fs.ext, 40
 annize.i18n, 41
 annize.object, 49
 annize.object.config, 49
 annize.project, 50
 annize.project.feature_loader, 71
 annize.project.file_formats, 60
 annize.project.inspector, 72
 annize.project.loader, 79
 annize.project.materializer, 63
 annize.project.materializer.behaviors, 64
 annize.project.materializer.behaviors.argument, 65
 annize.project.materializer.behaviors.basket, 66
 annize.project.materializer.behaviors.block, 67
 annize.project.materializer.behaviors.feature_unavailable, 67
 annize.project.materializer.behaviors.reference, 68
 annize.project.materializer.core, 68
 annize.project.materializer.object_factory, 71
 annize.project.materializer.preprocessors, 71
 annize.ui, 80

annize.ui.apps, 81
 annize.user_feedback, 82
 annize.user_feedback.static, 84
 Mount (class in annize.fs.ext), 41
 move_to() (annize.fs.Path method), 38
 mtime() (annize.fs.Path method), 36
 MultipleValuesForSingleArgumentError, 71

N

name (annize.project.ArgumentNode property), 58
 name (annize.project.inspector.BasicInspector._ScalarTypeInfo property), 76
 name (annize.project.inspector.BasicInspector.TypeInfo property), 75
 name (annize.project.inspector.FullInspector.CreatableInfo property), 79
 new_file_node() (annize.project.file_formats.FileFormat method), 62
 new_value (annize.project.Node.PropertyChangedEvent property), 55
 NoCurrentCultureError, 48
 node (annize.project.materializer.core.EarlyNodeMaterialization property), 68
 node (annize.project.materializer.core.NodeMaterialization property), 69
 Node (class in annize.project), 50
 Node.__ChildrenListChangeEvent (class in annize.project), 54
 Node.ChangeEvent (class in annize.project), 54
 Node.ChildAddedEvent (class in annize.project), 54
 Node.ChildRemovedEvent (class in annize.project), 54
 Node.PropertyChangedEvent (class in annize.project), 55
 node_by_name() (annize.project.inspector.BasicInspector method), 73
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 65
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 66
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 66
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 65
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 67
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 67
 node_context() (annize.project.materializer.behaviors.argument.ContextArgumentNodeBehavior method), 68
 NodeMaterialization (class in annize.project.materializer.core), 69
 nodes (annize.project.inspector.FullInspector.ArgumentMatching property), 78
 NullUserFeedbackController (class in annize.user_feedback), 83
 NumericVersionPatternPart (class in annize.data.version), 18

O

object_by_name() (annize.flow.run_context.RunContext method), 29
 object_by_name() (in module annize.flow.run_context), 32
 object_metadata() (annize.flow.run_context.RunContext method), 31
 object_metadata() (in module annize.flow.run_context), 33
 object_name() (annize.flow.run_context.RunContext method), 29
 object_name() (in module annize.flow.run_context), 32
 object_names() (annize.flow.run_context.RunContext method), 29
 object_names() (in module annize.flow.run_context), 32
 ObjectNode (class in annize.project), 58
 objects_by_type() (annize.flow.run_context.RunContext method), 30
 objects_by_type() (in module annize.flow.run_context), 32
 objects_for_node() (annize.project.materializer.MaterializationResult method), 64
 old_value (annize.project.Node.PropertyChangedEvent property), 55
 on_unresolvable (annize.project.ReferenceNode property), 59
 OptionalVersionPatternPart (class in annize.data.version), 19
 OutOfContextError, 22
 OutOfContextErrorNodeBehavior

P

parameter_config() (in module annize.object.config), 49
 parameter_info() (annize.project.inspector.BasicInspector method), 73
 parameter_info() (in module annize.object.config), 49
 parent (annize.project.Node property), 51
 parse() (in module annize.annize_cli), 84
 ParserError, 60
 parts (annize.data.version.VersionPattern property), 17
 parts (annize.features.files.common.Directory property), 23
 path (annize.fs.ext.FreshTempDirectory property), 40

- path (*annize.project.FileNode* property), 57
 Path (class in *annize.fs*), 36
 path() (*annize.fs.FilesystemContent* method), 36
 path() (*annize.fs.Path* method), 36
 Path._TransferHelper (class in *annize.fs*), 38
 Path.TransferFilters (class in *annize.fs*), 38
 Path.TransferFilters.And (class in *annize.fs*), 38
 pattern (*annize.data.version.Version* property), 17
 possible_append_to_targets_for_node() (*annize.project.inspector.FullInspector* method), 78
 possible_argument_infos_for_child_in_parent() (*annize.project.inspector.BasicInspector* method), 74
 possible_argument_names_for_child_in_parent() (*annize.project.inspector.BasicInspector* method), 73
 possible_reference_targets_for_node_argument() (*annize.project.inspector.FullInspector* method), 77
 prepare() (*annize.flow.run_context.RunContext* method), 29
 pretty_project_name (*annize.features.base.Data* property), 26
 pretty_project_name() (in module *annize.features.base*), 27
 problems (*annize.project.materializer.core.NodeMaterialization* property), 69
 project (*annize.project.Node* property), 51
 project_annize_config_directory() (in module *annize.project.loader*), 80
 project_annize_config_main_file() (in module *annize.project.loader*), 79
 project_cultures() (in module *annize.features.i18n.common*), 25
 project_directory (*annize.features.base.Data* property), 26
 project_directory() (in module *annize.features.base*), 27
 project_name (*annize.features.base.Data* property), 26
 project_name() (in module *annize.features.base*), 27
 project_root_directory() (in module *annize.project.loader*), 80
 ProjectCultures (class in *annize.features.i18n.common*), 25
 ProjectDirectory (class in *annize.features.files.common*), 24
 ProjectMaterializer (class in *annize.project.materializer.core*), 69
 ProjectNode (class in *annize.project*), 55
 property_name (*annize.project.Node.PropertyChangedEvent* property), 55
 ProvidedTrStr (class in *annize.i18n*), 47
- ## R
- readme_pdf() (in module *annize.asset.data*), 15
 red (*annize.data.color.Color* property), 15
 reference_key (*annize.project.ReferenceNode* property), 59
 ReferenceBehavior (class in *annize.project.materializer.behaviors.reference*), 68
 ReferenceNode (class in *annize.project*), 59
 ReferenceNode.OnUnresolvableAction (class in *annize.project*), 59
 regexp_string (*annize.data.version.ConcatenatedVersionPatternPart* property), 20
 regexp_string (*annize.data.version.NumericVersionPatternPart* property), 19
 regexp_string (*annize.data.version.OptionalVersionPatternPart* property), 20
 regexp_string (*annize.data.version.SeparatorVersionPatternPart* property), 19
 regexp_string (*annize.data.version.VersionPatternPart* property), 18
 region_code (*annize.i18n.Culture* property), 45
 register_file_format() (in module *annize.project.file_formats*), 62
 relative_path (*annize.features.files.common.FsEntry* property), 21
 remove() (*annize.fs.Path* method), 37
 remove_change_handler() (*annize.project.Node* method), 51
 remove_child() (*annize.project.Node* method), 52
 remove_child() (*annize.project.ProjectNode* method), 55
 resolve_appendtonodes() (in module *annize.project.materializer.preprocessors*), 71
 resolve_reference_node() (*annize.project.inspector.BasicInspector* method), 73
 result (*annize.project.materializer.core.NodeMaterialization* property), 69
 root (*annize.features.files.common.DirectoryPart* property), 23
 root (*annize.features.files.common.FsEntry* property), 21
 root_objects (*annize.project.materializer.MaterializationResult* property), 63
 run_runner() (*annize.flow.runner.Runner* method), 34
 RunContext (class in *annize.flow.run_context*), 28
 Runner (class in *annize.flow.runner*), 34
- ## S
- sanitized_project_name() (in module *annize.features.base*), 27
 saturation (*annize.data.color.Color* property), 16

save() (annize.project.ProjectNode method), 55
 save_file_node() (annize.project.file_formats.FileFormat.Marshaler method), 61
 save_file_node() (annize.project.file_formats.FileFormat.NullMarshaler method), 61
 ScalarValueNode (class in annize.project), 59
 segment_names (annize.data.version.ConcatenatedVersionPatternPart property), 20
 segment_names (annize.data.version.NumericVersionPatternPart property), 19
 segment_names (annize.data.version.OptionalVersionPatternPart property), 20
 segment_names (annize.data.version.SeparatorVersionPatternPart property), 19
 segment_names (annize.data.version.VersionPatternPart property), 18
 segment_names (annize.data.version.VersionPatternPart property), 18
 segments_tuples (annize.data.version.Version property), 17
 segments_tuples_to_text() (annize.data.version.ConcatenatedVersionPatternPart method), 20
 segments_tuples_to_text() (annize.data.version.NumericVersionPatternPart method), 19
 segments_tuples_to_text() (annize.data.version.OptionalVersionPatternPart method), 20
 segments_tuples_to_text() (annize.data.version.SeparatorVersionPatternPart method), 19
 segments_tuples_to_text() (annize.data.version.VersionPatternPart method), 18
 segments_tuples_to_text() (annize.data.version.VersionPatternPart method), 18
 segments_values (annize.data.version.Version property), 17
 SeparatorVersionPatternPart (class in annize.data.version), 19
 serialize_node() (annize.project.file_formats.FileFormat method), 62
 serialize_node() (annize.project.file_formats.FileFormat.Marshaler method), 61
 serialize_node() (annize.project.file_formats.FileFormat.NullMarshaler method), 61
 set_materialized_result() (annize.project.materializer.core.NodeMaterialization method), 69
 set_object_metadata() (annize.flow.run_context.RunContext method), 31
 set_object_metadata() (in module annize.flow.run_context), 33
 set_object_name() (annize.flow.run_context.RunContext method), 29
 set_object_name() (in module annize.flow.run_context), 32
 set_problems() (annize.project.materializer.core.NodeMaterialization method), 69
 set_selected_task() (annize.flow.runner.Runner method), 34
 show_task_chooser() (annize.flow.runner.Runner method), 34
 show_task_execution() (annize.flow.runner.Runner method), 34
 show_task_execution_success() (annize.flow.runner.Runner method), 34
 SKIP (annize.project.ReferenceNode.OnUnresolvableAction attribute), 59
 source_path (annize.features.files.common.DirectoryPart property), 23
 StaticUserFeedbackController (class in annize.user_feedback.static), 84
 str_to_value() (annize.data.version.ConcatenatedVersionPatternPart method), 20
 str_to_value() (annize.data.version.NumericVersionPatternPart method), 19
 str_to_value() (annize.data.version.OptionalVersionPatternPart method), 20
 str_to_value() (annize.data.version.SeparatorVersionPatternPart method), 19
 str_to_value() (annize.data.version.VersionPatternPart method), 18
 String (class in annize.features.i18n.common), 25
 string_name (annize.i18n.ProvidedTrStr property), 47
 studio() (annize.annize_cli.Commands method), 85
 summary (annize.features.base.Data property), 26
 summary() (in module annize.features.base), 27

T

target_node (annize.project.Node.ChangeEvent property), 54
 Task (class in annize.features.task), 28
 temp_clone() (annize.fs.Path method), 37
 text (annize.data.version.Version property), 17
 text_to_segments_tuples() (annize.data.version.VersionPatternPart method), 18
 TextSource (class in annize.features.i18n.gettext), 26

`tr()` (*annize.i18n.TrStr static method*), 42
`tr()` (*in module annize.i18n*), 46
`transfer_action_copy()` (*annize.fs.Path._TransferHelper static method*), 38
`transfer_action_move()` (*annize.fs.Path._TransferHelper static method*), 38
`transfer_to()` (*annize.fs.Path._TransferHelper static method*), 38
`translate()` (*annize.features.i18n.common._ProjectDefinedTranslationProvider method*), 24
`translate()` (*annize.i18n.GettextTranslationProvider method*), 46
`translate()` (*annize.i18n.TranslationProvider method*), 43
`translate()` (*annize.i18n.TrStr method*), 42
`translate()` (*in module annize.i18n*), 43
`TranslationProvider` (*class in annize.i18n*), 43
`TranslationUnavailableError`, 48
`TrStr` (*class in annize.i18n*), 42
`trstr()` (*in module annize.i18n*), 43
`try_get_materialization_for_node()` (*annize.project.materializer.core.NodeMaterialization method*), 69
`TTransferFilter` (*annize.fs.Path attribute*), 37
`type` (*annize.project.inspector.BasicInspector._ScalarTypeInfo property*), 76
`type` (*annize.project.inspector.BasicInspector.TypeInfo property*), 75
`type_documentation()` (*annize.project.inspector.BasicInspector method*), 74
`type_info` (*annize.project.inspector.FullInspector.CreatableInfo property*), 79
`type_info()` (*annize.project.inspector.BasicInspector method*), 72
`type_name` (*annize.project.ObjectNode property*), 58
`type_short_name` (*annize.project.inspector.FullInspector.CreatableTypeInfo property*), 79

U

`undo_changes()` (*annize.project.ProjectNode method*), 56
`UnresolvableReferenceError`, 60
`UnsatisfiableUserFeedbackAttemptError`, 83
`unspecified_culture` (*in module annize.i18n*), 47
`UpdatePOs` (*class in annize.features.i18n.gettext*), 26
`UserFeedbackController` (*class in annize.user_feedback*), 82

V

`value` (*annize.project.ScalarValueNode property*), 59

`Version` (*class in annize.data.version*), 17
`VersionPattern` (*class in annize.data.version*), 17
`VersionPatternPart` (*class in annize.data.version*), 18

W

`wait_finished()` (*annize.flow.runner.Runner method*), 35
`with_modified()` (*annize.data.color.Color method*), 16
`write_file()` (*annize.fs.Path method*), 36