

---

**Annize**

***Release 7.0.6539***

**Author name not set**

**29.11.2025**



---

## Inhaltsverzeichnis

---

<b>1</b>	<b>Lizenz</b>	<b>3</b>
<b>2</b>	<b>Up-to-date?</b>	<b>5</b>
<b>3</b>	<b>Abhängigkeiten</b>	<b>7</b>
<b>4</b>	<b>Benutzerhandbuch</b>	<b>9</b>
4.1	Erste Schritte . . . . .	9
4.2	Annize-Projekte . . . . .	11
4.3	Annize Studio . . . . .	11
4.4	Annize Programmierschnittstelle . . . . .	11
<b>5</b>	<b>Kommandozeilenschnittstellen-Referenz</b>	<b>13</b>
5.1	Positional Arguments . . . . .	13
5.2	Named Arguments . . . . .	13
5.3	Sub-commands . . . . .	14
<b>6</b>	<b>API-Referenz</b>	<b>15</b>
6.1	annize namespace . . . . .	15
	<b>Python-Modulindex</b>	<b>87</b>
	<b>Stichwortverzeichnis</b>	<b>89</b>



Annize ist ein auf Python basierendes Framework zur Automatisierung von Workflows. Es ermöglicht Nutzern, Objekte und Tasks zu definieren, die auf wiederverwendbare Annize-Features verweisen, die spezifische Funktionalitäten implementieren. Diese Features können entweder benutzerdefinierte Implementierungen oder aus externen Paketen installierte Module sein. Annize dient hauptsächlich als Ausführungseengine, während die meisten Funktionalitäten von diesen modularen Features bereitgestellt werden.

Projekte in Annize werden über Konfigurationsdateien (normalerweise XML) definiert, die es ermöglichen, Tasks mit unterschiedlichen Eingabewerten auszuführen. Das Annize Studio bietet eine Benutzeroberfläche, um Objekte komfortabel zu konfigurieren, Fehler zu überprüfen und Tasks auszuführen. Es ist für Entwickler konzipiert, kann aber an eine Vielzahl von Automatisierungsbedürfnissen angepasst werden und bietet Flexibilität für alles von einfachen bis hin zu komplexeren Workflows.



# KAPITEL 1

---

## Lizenz

---

Annize wird unter den Bedingungen der AGPL 3 Lizenz verteilt. Das betrifft ebenso alle Dateien ohne Lizenzkopf (Nichtquelldateien, wie Grafiken), außer sie sind ausdrücklich als Drittinhalt gekennzeichnet. Lesen Sie die Sektion ‚Abhängigkeiten‘ bezüglich enthaltener Drittinhalte.





## KAPITEL 2

---

Up-to-date?

---

Lesen Sie diesen Text von einer anderen Quelle als der Homepage? Falls Sie unsicher sind ob Ihr Paket aktuell ist, sollten Sie die Homepage besuchen und das prüfen. Sie lesen derzeit die Dokumentation für Version 7.0.6539.



---

## Abhängigkeiten

---

Annize nutzt einige Teile von Drittanbietern.



Benötigt: **Python 3.13**



Benötigt: **Python package hallyd**  $\approx$  0.9003



Benötigt: **Python package klovve[graphical]**  $\approx$  1.6



Benötigt: **Python package lxml**  $\approx$  6.0



Benötigt: **Python package pycountry**  $\approx$  24.6



Benötigt: **Python package pyperclip**  $\approx$  1.11



Empfohlen: **GNU/Linux**



Enthalten: **background image** (Lizenz: <https://creativecommons.org/licenses/by-sa/3.0/>; von [hier](#))



Enthalten: **font ,Inconsolata‘** (for websites; by Raph Levien; Lizenz: OFL; [von hier](#))



Enthalten: **font ,Khula‘** (for websites; by Erin McLaughlin; Lizenz: OFL; [von hier](#))



Enthalten: **font ,Symbola‘** (for logo symbol; Lizenz: free for use; [von hier](#))



Enthalten: **icon set ,Oxygen‘** (some icons; [von hier](#))



Enthalten: **third-party project logos** ([von hier](#))

## 4.1 Erste Schritte

### 4.1.1 Installation

Besuche die Annize-Website, um das Installationspaket zu finden, das am besten zu deiner Umgebung passt. Die Website bietet verschiedene Optionen je nach deinem System und deinen Präferenzen. Installiere dann dieses Paket.

Der einfachste Weg, Annize zu installieren, ist möglicherweise über `pip`, den Paketmanager von Python. Bereite dazu zunächst deine `pip`-Umgebung vor (einschließlich einer `venv`) und führe dann `pip install annize` aus.

Nach der Installation solltest du Annize ausführen können. Um sicherzustellen, dass Annize korrekt installiert wurde, führe einfach `annize --help` aus.

### 4.1.2 Das erste Annize-Projekt

Im Folgenden richten wir ein erstes Annize-Projekt ein, um die Schritte zu zeigen. Dazu implementieren wir Datenstrukturen und Logik, die wir später in Annize-Projekten verwenden können. Anschließend erstellen wir ein neues Annize-Projekt, richten dort unsere Funktionalitäten ein und führen es abschließend aus.

#### Ein Annize Feature vorbereiten

In Annize wird alle Funktionalität durch **Features** bereitgestellt. Ein Feature ist ein Python-Submodul im Namespace `annize.features`, das eine bestimmte Funktionalität implementiert, auf die ein Annize-Projekt später zugreifen kann.

Annize selbst dient hauptsächlich als Ausführungseingabe und enthält nur wenige eingebaute Features. Features können entweder manuell implementiert oder aus zusätzlichen Paketen installiert werden (wie z.B. `Annize-pidev`, das Features für Annizes eigenes Projekt bereitstellt).

Um ein neues Feature zu erstellen, muss ein Python-Paket unter dem Namespace `annize.features` hinzugefügt werden. Zum Beispiel, um ein einfaches `hello_world`-Feature zu erstellen:

1. Erstelle in deinem Python-Paketverzeichnis das Verzeichnis `annize/features`.
2. Erstelle in diesem Verzeichnis eine Datei mit dem Namen `hello_world.py`.
3. Öffne diese Datei in deiner bevorzugten Python-Entwicklungsumgebung oder einem Texteditor.

Zunächst definieren wir eine einfache Datenstruktur:

```
class MyData:

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

Dann definieren wir einen Task, den wir ausführen können (typischerweise mit den `import`-Zeilen am Anfang der Datei):

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

class MyTask:

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

    def __call__(self):
        # hole alle MyData-Objekte, die im Projekt definiert sind ...
        my_datas = annize.flow.run_context.objects_by_type(MyData)
        # und für alle diese Objekte ...
        for my_data in my_datas:
            # falls ihr 'foo' mit dem des Tasks übereinstimmt ...
            if my_data.foo == self.foo:
                # teile es dem Nutzer mit
                annize.user_feedback.message_dialog(my_data.bar, ["OK"])
```

Diese Aufgabe würde, sobald ausgeführt, alle `MyData`-Objekte sammeln und für jedes mit dem „richtigen“ `foo`-Wert eine Nachricht für den Benutzer anzeigen. Reale Features interagieren eher selten mit dem Benutzer, sondern führen in der Regel reale Aufgaben aus.

Optional können wir für die beiden Klassen zur Dokumentation Python ‚docstrings‘ verfassen, inklusive Sphinx-basierter Parameterdokumentation beim Konstruktor.

## Ein neues Annize-Projekt erstellen

Mit der implementierten Funktionalität kannst du nun ein Annize-Projekt erstellen und die definierten Features nutzen.

1. Erstelle ein leeres Verzeichnis: Beginne damit, ein neues leeres Verzeichnis für dein Annize-Projekt zu erstellen, z.B. irgendwo in deinem Benutzerverzeichnis.
2. Starte Annize: Öffne Annize entweder aus dem Startmenü oder durch Ausführen von `annize` auf der Kommandozeile.
3. Erstelle ein neues Projekt: Erstelle im Annize-Interface ein neues Projekt und wähle das leere Verzeichnis als Projektverzeichnis aus.
4. Klappe die `project.xml`-Box aus.
5. Erstelle ein `MyData`-Objekt: Erstelle ein neues `MyData`-Objekt in `project.xml`. Weise den Eigenschaften `foo` und `bar` Werte zu.
6. Erstelle ein `MyTask`-Objekt: Füge ein `MyTask`-Objekt in `project.xml` hinzu und weise ihm einen Wert für `foo` zu.
7. Benenne das `MyTask`: Weise dem `MyTask`-Objekt einen Namen zu.

8. Speichere das Projekt: Sobald die Konfiguration abgeschlossen ist, speichere das Projekt.

### Das Annize-Projekt ausführen

Um das Projekt auszuführen, klicke auf den Play-Button. Ein Auswahlfenster erscheint und zeigt den vorher zugewiesenen Namen des MyTask-Objekts an. Wähle diesen Task aus, um die Ausführung zu starten.

Je nach den zuvor gewählten Werten wird die Ausführung entweder sofort abgeschlossen oder zeigt während der Ausführung Nachrichten für den Benutzer an.

Du kannst auch weitere MyTask-Objekte mit anderen Werten und Namen sowie weitere MyData-Objekte zum Projekt hinzufügen.

Was wir bisher behandelt haben, stellt die grundlegende Funktionalität von Annize dar. Mit dieser Grundlage kannst du bereits eine Vielzahl von Projekten erstellen. Die folgenden Abschnitte werden detailliertere und fortgeschrittene Funktionen und Themen behandeln.

## 4.2 Annize-Projekte

Ein **Annize-Projekt** besteht aus Objektdefinitionen, die auf verfügbare Annize-Features verweisen. Einige dieser Objekte sind typischerweise ausführbare Tasks, die mittels der Ausführung von Annize gestartet werden können.

Die Konfigurationsdateien für ein Annize-Projekt werden im `-meta`-Unterverzeichnis des Projektstammverzeichnisses abgelegt. Es sind auch alternative Namen erlaubt, aber kaum gebräuchlich. Dieses Verzeichnis enthält zumindest eine `project.xml`-Datei.

Zusätzlich zu `project.xml` ist es möglich, weitere `project.*.xml`-Dateien hinzuzufügen. Die Struktur dieser Dateien ist flexibel; Annize stellt keine Anforderungen daran, wo oder wie verschiedene Objekte definiert werden.

Alle Objekte im Projekt können Namen zugewiesen bekommen, die überall im Projekt referenziert werden können.

Weitere Details zur Projektstruktur gehen über diese Übersicht hinaus.

## 4.3 Annize Studio

Das Annize **Studio** erscheint, wenn du `annize` ausführst (wie bereits in den „Ersten Schritten“).

Das Annize Studio bietet eine umfassende Umgebung zur Konfiguration und Verwaltung von Annize-Projekten. Über diese Oberfläche kannst du:

- Objekte hinzufügen, entfernen oder ändern: Konfiguriere dein Projekt, indem du neue Objekte hinzufügst, unnötige entfernst oder bestehende Konfigurationen bearbeitest, einschließlich Referenzen auf andere Objekte und mehr.
- Konfigurationsprobleme erkennen: Die Oberfläche hebt Probleme mit der aktuellen Projektkonfiguration hervor und hilft dir, diese schnell zu identifizieren und zu lösen.
- Auf Feature-Dokumentation zugreifen: Sieh dir die Dokumentation der verfügbaren Features direkt innerhalb der Oberfläche an, um nützliche Informationen und Anleitungen zu erhalten.
- Die Projektausführung starten: Sobald das Projekt konfiguriert ist, kannst du die Ausführung direkt aus der Oberfläche starten, um schnelle Tests und Iterationen durchzuführen.

## 4.4 Annize Programmierschnittstelle

Es gibt mehrere fortgeschrittene Themen zur Annize-API, die für ein tieferes Verständnis empfohlen werden:

- `annize.fs` und `annize.features.files.common`.

- *annize.features.base.*
- *annize.features.task.*
- *annize.i18n* und *annize.features.i18n.common.*
- *annize.user\_feedback.*



---

## Kommandozeilenschnittstellen-Referenz

---

```
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

<b>task_name</b>	Task name.
	Default: ''

## 6.1 annize namespace

### 6.1.1 Subpackages

**annize.asset** package

**Submodules**

**annize.asset.data** module

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

**Parameter**

**culture** (*str*)

**Rückgabety**

*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.

**Parameter**

- **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.

**Parameter**

- **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](#).

**Rückgabetyp**

[Color](#)

**property html\_color\_spec: str**

The HTML color specification of this color.

`__get_normed_value(value_2)`

Parameter

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

Rückgabetyp

*float*

`__html_color_spec__part()`

Parameter

`part_value` (*int*)

Rückgabetyp

*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`.

Parameter

- `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](#).

**Parameter**

**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.

**Parameter**

**text** (*str*) – The text to extract.

**Rückgabotyp**

*Sequence*[\_SegmentTuple]

**segments\_tuples\_to\_text**(segments\_tuples)

Return a textual representation for the given segment tuples.

**Parameter**

**segments\_tuples** (*Iterable*[\_SegmentTuple]) – The segment tuples.

**Rückgabotyp**

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.

**Parameter**

**segments\_tuples** (*Sequence*[\_SegmentTuple]) – The segment tuples.

**Rückgabotyp**

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.

**Parameter**

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

**Rückgabety***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.

**Parameter****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.

**Parameter****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.**Parameter**

- **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.

**Parameter****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.

**Parameter****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.**Parameter**

- **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.

**Parameter**

**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.

**Parameter**

**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.

**Parameter**

- **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.

**Parameter**

**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.

**Parameter**

**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.

**Parameter**

- **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](#).

#### Parameter

- **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.

#### Parameter

- **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](#).

#### Parameter

- **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*)

**Parameter**

- **text** (*str*)
- **by\_text** (*str*)
- **by\_pattern** (*Pattern*)

**Rückgabetyt**

bool

**does\_exclude**(*relative\_path*, *source*, *destination*)

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

**Parameter**

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

**Rückgabetyt**

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.

**Parameter**

**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.

**Parameter**

- **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](#).

**Parameter**

- **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).

#### Parameter

- **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()`

**Parameter**  
**exclude** (*Exclude*)

**Rückgabety**  
annize.fs.Path.TTransferFilter

**class** annize.features.files.common.**ProjectDirectory**

Bases: *FilesystemContent*

The Annize project root directory.

**Parameter**  
**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. /.

**Parameter**  
**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.

**Parameter**  
• **string\_name** – The string name.  
• **culture** – The culture.

**add\_translations**(*string\_name*, *variants*)

**Parameter**  
• **string\_name** (*str*)

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

**Rückgabotyp**

None

**\_ProjectDefinedTranslationProvider\_\_translations\_for\_string\_name**(*string\_name*)

**Parameter**

**string\_name** (*str*)

**Rückgabotyp**

*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()`.

**Parameter**

- **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()`.

**Parameter**

- **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.

**Parameter**

**cultures** (*Sequence[Culture]*)

`annize.features.i18n.common.project_cultures()`

Return a list of the current Annize project's target cultures. See also [ProjectCultures](#).

**Rückgabotyp**

*Sequence[Culture]*

### annize.features.i18n.gettext module

gettext-based internationalization.

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

Bases: object

**Parameter**

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

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

Bases: object

**Parameter**

- **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

**Parameter**

- **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

**Parameter**

- **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

    Parameter
        iso (str)

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

    Parameter
        items (list[object])

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

    Parameter
        objects (list[object])

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

    Parameter
        • key (str)
        • default (Any)

    Rückgabotyp
        Any

annize.features.base.project_name()

    Rückgabotyp
        str

annize.features.base.pretty_project_name()

    Rückgabotyp
        TrStr

annize.features.base.summary()

    Rückgabotyp
        TrStr

annize.features.base.long_description()

    Rückgabotyp
        TrStr

annize.features.base.homepage_url()

    Rückgabotyp
        str

annize.features.base.project_directory()

    Rückgabotyp
        Path

```

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

**Parameter**

**name** (*str*)

**Rückgabotyp**

*str*

## **annize.features.task module**

Tasks.

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

Bases: `object`

**Parameter**

- **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 = '__026LF88xLld5001004A1t0031Ht'`



```
_NAMES__METADATA_KEY = '__026LF88xLwg5002004A1t0031Ht'
```

```
ANNIZE_CONFIG_DIRECTORY__NAME = '__026LF88xLyza003004A1t0031Ht'
```

```
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.

**Parameter**

**annize\_config\_directory** (*Path*) – The Annize project configuration root path.

**Rückgabety**

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\(\)](#).

**Parameter**

- **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.

**Rückgabety**

*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\(\)](#).

**Parameter**

**obj** (*Any*) – The object.

**Rückgabety**

*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\(\)](#).

**Parameter**

**obj** (*Any*) – The object.

**Rückgabety**

*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.

**Parameter**

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

**Rückgabetyt**

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.

**Parameter**

- **obj\_type** (*type[T]*) – The type.
- **toplevel\_only** (*bool*) – Whether to return only objects that are defined on project level.

**Rückgabetyt**

*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.

**Parameter**

**obj** (*Any*) – The object to add.

**Rückgabetyt**

*str*

**is\_friendly\_name**(*name*)

Returns whether the given name is a friendly one.

**Parameter**

**name** (*str*) – The name to check.

**Rückgabetyt**

*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).

**Parameter**

**obj** (*Any*) – The object to check.

**Rückgabetyt**

*bool*

**mark\_object\_as\_toplevel(*obj*)**

Mark an object as a toplevel one. See [is\\_toplevel\\_object\(\)](#).

**Parameter**

**obj** (*Any*) – The object.

**Rückgabetyt**

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\(\)](#).

**Parameter**

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

**Rückgabetyt**

*Any*

**set\_object\_metadata(*obj*, *key*, *value=None*)**

Store a piece of metadata for a given object.

See also [object\\_metadata\(\)](#).

**Parameter**

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

**Rückgabetyt**

None

**\_\_put\_object(*name*, *obj*)****Parameter**

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

**Rückgabetyt**

None

**\_\_object\_raw\_name(*obj*)****Parameter**

**obj** (*Any*)

**Rückgabetyt**

*str*

**\_\_object\_metadata\_dict(*obj*)****Parameter**

**obj** (*Any*)

**Rückgabotyp**

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.**Rückgabotyp**`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()`).**Parameter**

- **name** (`str`)
- **default** (`Any`)
- **create\_nonexistent** (`bool`)

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

list[str]

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

str

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

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

**Rückgabotyp**

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()`).

**Parameter**

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

**Rückgabotyp**

*Sequence*

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

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

**Parameter**

`obj` (*Any*)

**Rückgabotyp**

`str`

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

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

**Parameter**

`name` (*str*)

**Rückgabotyp**

`bool`

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

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

**Parameter**

`obj` (*Any*)

**Rückgabotyp**

`bool`

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

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

**Parameter**

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

**Rückgabotyp**

*Any*

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

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

**Parameter**

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

**Rückgabetyt**

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, feature_loader=None)
```

Bases: ABC

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

### Parameter

- **project** ([ProjectNode](#))
- **selected\_task** (*str* / *None*)
- **user\_feedback\_answers** (*dict[str, Any]*)
- **user\_feedback** ([annize.user\\_feedback.UserFeedbackController](#) / *None*)
- **feature\_loader** ([FeatureLoader](#) / *None*)

**run\_runner()**

**Rückgabetyt**

None

**abstractmethod show\_task\_chooser()**

**Rückgabetyt**

None

**abstractmethod show\_task\_execution()**

**Rückgabetyt**

None

**abstractmethod show\_task\_execution\_success()**

**Rückgabetyt**

None

**get\_tasks()**

**Rückgabetyt**

list[str]

**get\_selected\_task()**

**Rückgabetyt**

str

**set\_selected\_task(task\_name)**

**Parameter**

**task\_name** (*str*)

```

    Rückgabotyp
    None

is_finished()

    Rückgabotyp
    bool

get_success_state()

    Rückgabotyp
    Tuple[bool, str]

wait_finished()

    Rückgabotyp
    None

__do_run(project)

    Parameter
    project (ProjectNode)

__set_tasks(tasks)

    Parameter
    tasks (list[str])

    Rückgabotyp
    None

__set_success_state(success, message)

    Parameter
    • success (bool)
    • message (str)

    Rückgabotyp
    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.

**Parameter**

**generate\_func** (*Callable*[[*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).

**Rückgabetyp**

`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()`.

**Parameter**

**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`).

**Rückgabetyp**

`Path`

**path()**

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

**Rückgabetyp**

`Path`

**children()**

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

**Rückgabetyp**

*Sequence*[`Path`]

**ctime()**

Return the ctime for this path.

**Rückgabetyp**

*datetime*

**mtime()**

Return the mtime for this path.



**Rückgabotyp***datetime***write\_file(*data*)**

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

**Parameter**

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

**Rückgabotyp**

None

**remove(\*, *missing\_ok=True*)**

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

**Parameter**

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

**Rückgabotyp**

None

**file\_size()**

Return the file size in bytes.

**Rückgabotyp**

int

**temp\_clone(\*, *temp\_root\_path=None*, *basename=None*)**

Return a temporary clone of the content at this path.

**Parameter**

- **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.

**Rückgabotyp***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.

**Parameter**

- **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.

### Rückgabotyp

[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.

### Parameter

- **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.

### Rückgabotyp

[Path](#)

**class TransferFilters**

Bases: `object`

**class And**(*\*inner\_filters*)

Bases: `object`

### Parameter

**inner\_filters** (*Path.TTransferFilter*)

**class \_TransferHelper**

Bases: `object`

**static transfer\_to**(*source*, *destination*, \*, *merge*, *overwrite*, *destination\_as\_parent*, *action*, *transfer\_filter=None*)

### Parameter

- **source** ([Path](#))
- **destination** ([Path](#))
- **merge** (*bool*)
- **overwrite** (*bool*)
- **destination\_as\_parent** (*bool*)
- **action** (*Callable*)
- **transfer\_filter** (*Callable*[[[Path](#), [Path](#), [Path](#)], *bool*] / *None*)

### Rückgabotyp

[Path](#)

**static transfer\_action\_copy**(*source*, *destination*)

### Parameter

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

**Rückgabotyp**

None

**static** `transfer_action_move`(*source*, *destination*)**Parameter**

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

**Rückgabotyp**

None

**static** `_TransferHelper__transfer_piece`(*action*, *transfer\_filter*, *source*, *destination*, *merge*, *overwrite*, *relative\_path*=")**Parameter**

- **action** ([Callable](#))
- **transfer\_filter** ([Callable](#)[[[Path](#), [Path](#), [Path](#)], [bool](#)] | [None](#))
- **source** ([Path](#))
- **destination** ([Path](#))
- **merge** ([bool](#))
- **overwrite** ([bool](#))
- **relative\_path** ([str](#))

**Rückgabotyp**

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.

**Parameter**

- **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*.

**Rückgabotyp**[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](#).

**Parameter**

- **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.

**Rückgabotyp**[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.

#### Parameter

- **content** (*str* | *bytes* | *Callable*[[*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.

#### Rückgabotyp

`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.

#### Parameter

- **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.

#### Parameter

- **content** (*str* | *bytes* | *Callable*[[*str* | *bytes*]])
- **file\_name** (*str* | *None*)
- **temp\_root\_path** (*str* | *Path* | *None*)

```
_TStaticContent = str | bytes
```

```
_TContent
```

```
alias 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).

#### Parameter

- **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 directo-
ry 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](#)).

**Parameter**

**culture** (*CultureSpecT*) – The culture.

**Rückgabety**

`str`

**format**(*\*args, \*\*kwargs*)

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

**Parameter**

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

**Rückgabety**

`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.

**Parameter**

**string\_name** (*str*) – The string name.

**Rückgabety**

`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.

**Parameter**

**culture** (`Culture`) – The culture.

**Rückgabetyt**

`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`).

**Parameter**

- **text** (`TrStrOrStr`) – The text to translate.
- **culture** (`CultureSpecT`) – The culture.

**Rückgabetyt**

`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`.

**Parameter**

**text** (`TrStrOrStr`) – The text.

**Rückgabetyt**

`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.

**Parameter**

- **string\_name** (`str`) – The string name.
- **culture** (`Culture`) – The culture.

**Rückgabetyp**

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.

**Parameter**

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

**Rückgabetyp**

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\(\)](#).

**Parameter**

- **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).

**Parameter**

- **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](#).

**Rückgabetyp**[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.

**Rückgabetyt**

*Iterable[Culture]*

**\_\_best\_system\_locale()**

**Rückgabetyt**

str

**\_\_current\_system\_locale\_setup()**

**Rückgabetyt**

*\_TSystemLocaleSetup*

**\_\_set\_system\_locale\_setup()**

**Parameter**

**system\_locale\_setup** (*\_TSystemLocaleSetup*)

**Rückgabetyt**

None

`__set_env__var(value)`

**Parameter**

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

**Rückgabety**

*None*

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

Bases: `object`

**Parameter**

- **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*

**Rückgabety**

*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.

**Parameter**

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

**Rückgabety**

*str*

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

Bases: *TranslationProvider*

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

**Parameter**

- **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.

**Parameter**

- **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()`.

**Parameter**

**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.

**Parameter**

**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`.

**Parameter**

**culture** (*CultureSpecT*) – The culture spec.

**Rückgabetyt**

*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".

**Parameter**

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

**Rückgabetyt**

*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.

**Parameter**

- **text** (*TrStr*)
- **language** (*str*)

`annize.i18n._translation_providers()`

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

See also [`add\_translation\_provider\(\)`](#).

**Rückgabetyt**

*Sequence[TranslationProvider]*

`annize.i18n._current_translation_providers_lists()`

`annize.i18n._annize_user_interaction_culture()`

**Rückgabetyt**

*Culture*

**class** `annize.i18n._FixedTrStr(text)`

Bases: *TrStr*

**Parameter**

**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.

**Parameter**

**culture** – The culture.

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

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

Bases: [TrStr](#)

**Parameter**

**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.

**Parameter**

**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.

**Parameter**

**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.

**Parameter**

- **for\_type** (*type*) – The type.
- **parameter\_name** (*str*) – The constructor's parameter name.

**Rückgabety**

[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\(\)`](#).

**Parameter**

**`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`](#).

**Parameter**

**`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.

**Parameter**

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

**Rückgabety**

[`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.

**Parameter**

- **`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.

**Rückgabety**

[`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\(\)](#).

**Parameter**

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

**Rückgabetyp**

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.

**Parameter**

- **handler** (*Callable*[[[ChangeEvent](#)], None]) – The handler function to remove.

**Rückgabetyp**

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.

**Parameter**

- **i** (*int*) – The position.

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

**Rückgabotyp**

None

**append\_child**(*node*)

Append a new child node.

**Parameter**

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

**Rückgabotyp**

None

**remove\_child**(*node*)

Remove a child node.

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

**Parameter**

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

**Rückgabotyp**

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.

**Rückgabotyp**

*Self*

**description**(*\*, with\_children=True, multiline=True*)

**Parameter**

- **with\_children** (*bool*)
- **multiline** (*bool*)

**Rückgabotyp**

str

**abstractmethod classmethod \_allowed\_child\_types**()

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

**Rückgabotyp**

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

**\_clone\_\_early**(*new\_node*)

Execute arbitrary steps during an early stage of node cloning.

**Parameter**

- **new\_node** (*Self*) – The cloned node.

**Rückgabotyp**

None



**`_clone__late(new_node)`**

Execute arbitrary steps during a late stage of node cloning.

**Parameter**

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

**Rückgabety**

None

**`_property_changed(property_name, old_value)`**

**Parameter**

- **`property_name`** (*str*)

- **`old_value`** (*Any*)

**Rückgabety**

None

**`abstractmethod _str_helper()`**

**Rückgabety**

*Iterable*[*str*]

**`__description(indent, with_children, multiline)`**

**Parameter**

- **`indent`** (*int*)

- **`with_children`** (*bool*)

- **`multiline`** (*bool*)

**Rückgabety**

*str*

**`__changed__child_added(child_node, child_position)`**

**Parameter**

- **`child_node`** (*Node*)

- **`child_position`** (*int*)

**Rückgabety**

None

**`__changed__child_removed(child_node, child_position)`**

**Parameter**

- **`child_node`** (*Node*)

- **`child_position`** (*int*)

**Rückgabety**

None

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

**Parameter**

- **`node`** (*Node*)

- **`property_name`** (*str*)

- **old\_value** (*Any*)

- **new\_value** (*Any*)

**Rückgabety**

None

**\_\_changed\_\_call\_\_handlers**(*event*)

**Parameter**

**event** ([ChangeEvent](#))

**Rückgabety**

None

**class ChangeEvent**(*target\_node*)

Bases: `object`

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

**Parameter**

**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.

**Parameter**

- **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.

**Parameter**

- **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.

**Parameter**

- **target\_node** ([Node](#))

- **child\_node** ([Node](#))
- **child\_position** (*int*)

**class** `PropertyChangedEvent`(*target\_node*, *property\_name*, *old\_value*, *new\_value*)

Bases: [ChangeEvent](#)

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

**Parameter**

- **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).

**Parameter**

**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.

**Rückgabety**

None

**insert\_child**(*i*, *node*)

Insert a new child node.

**Parameter**

- **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`.**Parameter****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\(\)](#).**Parameter**

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

**Rückgabetyp***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\(\)](#).**Parameter****since** (*int*) – The timestamp where to start with undoing changes (inclusive).**Rückgabetyp**

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\(\)](#).**Parameter**

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

**Rückgabetyp**[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.

**Parameter****new\_node** – The cloned node.**\_clone\_\_late**(*new\_node*)

Execute arbitrary steps during a late stage of node cloning.

**Parameter****new\_node** – The cloned node.

```

_str_helper()

__reset_change_history()

    Rückgabetyt
    None

__handle_changed(event)

    Parameter
    event (ChangeEvent)

    Rückgabetyt
    None

__compacted_changes()

    Parameter
    events (Sequence[ChangeEvent])

    Rückgabetyt
    Sequence[ChangeEvent | None]

__is_inverse_of(event_2)

    Parameter
    • event_1 (ChangeEvent)
    • event_2 (ChangeEvent)

    Rückgabetyt
    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.

**Parameter**

- **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.

**Parameter**

**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](#).

#### Parameter

- **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)

        Parameter
            • obj (Any)
            • max_length (int)

        Rückgabetyt
            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

**Parameter**

**feature\_name** (*str*)

**exception annize.project.BadStructureError(*message*)**

Bases: ValueError

**Parameter**

**message** (*str*)

**exception annize.project.MaterializerError(*message*)**

Bases: TypeError

**Parameter**

**message** (*str*)

**exception annize.project.ParserError(*message*)**

Bases: ValueError

Parsing error like bad input XML.

**Parameter**

**message** (*str*)

**exception annize.project.UnresolvableReferenceError(*reference\_key*)**

Bases: [MaterializerError](#)

**Parameter**

**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.

**Parameter**

**change** ([ChangeEvent](#)) – The change.

**Rückgabotyp**

None

**abstractmethod save\_file\_node()**

Save this file node back to disk.

**Rückgabotyp**

None

**abstractmethod serialize\_node(*node*)**

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

**Parameter**

**node** ([ArgumentNode](#)) – The node to serialize.

**Rückgabotyp**

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.

**Parameter**

**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.

**Parameter**

**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.

**Parameter**

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

**Rückgabotyp**

[FileNode](#)

**abstractmethod** `new_file_node(file, inspector)`

Return a new empty project file node.

That file node has a marshaler; see [load\\_file\\_node\(\)](#).

**Parameter**

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

**Rückgabotyp**

[FileNode](#)

**serialize\_node(node)**

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

**Parameter**

**node** ([ArgumentNode](#)) – The node to serialize.

**Rückgabotyp**

bytes

**abstractmethod** `deserialize_node(s, inspector)`

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

**Parameter**

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

**Rückgabotyp**

[ArgumentNode](#)

`_abc_impl = <_abc._abc_data object>`

`annize.project.file_formats.register_file_format(format_name)`

Return a decorator that registers a file format.

**Parameter**

**format\_name** (*str*) – The format name.

**Rückgabotyp**

*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\(\)](#).

**Parameter**

**format\_name** (*str*) – The format name. A typical name is "xml".

**Rückgabotyp**

[FileFormat](#) | None

`annize.project.file_formats.all_file_format_names()`

Return all known file format names.

**Rückgabotyp**

*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()`.

#### Parameter

- **project\_annize\_config\_directory** (*str* / *Path*) – The Annize project configuration directory
- **inspector** (*FullInspector*) – The project inspector to use.

#### Rückgabotyp

*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`

#### Parameter

- **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*)

**Parameter**  
**node** (`Node`)

**Rückgabotyp**  
`list[Any] | None`

**erroneous\_nodes**()

**Rückgabotyp**  
`list[Node]`

**errors\_for\_node**(*node*)

**Parameter**  
**node** (*Any*)

**Rückgabotyp**  
`list[Exception]`

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

**Parameter**

- **project** (`ProjectNode`)
- **feature\_loader** (`FeatureLoader` | *None*)

**Rückgabotyp**  
`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)`

**Parameter**

- **original\_project\_node** (`ProjectNode`)
- **cloned\_project\_node** (`ProjectNode`)

**Rückgabotyp**  
`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.

**Parameter**

**early\_node\_materialization** ([EarlyNodeMaterialization](#))

**Rückgabetyp**

*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.

**Parameter**

- **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.

**Rückgabetyp**

*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).

**Parameter**

**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.

**Parameter**

- **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](#)

**Parameter**

**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.

**Parameter**

- **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.

**Parameter**

- **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.

#### Parameter

- **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.

**Parameter**

- **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.

**Parameter**

- **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`

**Parameter**

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



```

property node: Node

early_materialize_children()

class annize.project.materializer.core.NodeMaterialization(materializer, node, store)
    Bases: object

        Parameter
            • materializer (ProjectMaterializer)
            • node (Node)
            • store (dict)

property node: Node

set_materialized_result(result)

        Parameter
            result (Iterable[Any])

        Rückgabetyt
            None

set_problems(problems)

        Parameter
            problems (Iterable[Exception])

materialized_children_tuples(*, desperate)

        Parameter
            desperate (bool)

materialized_children(*, desperate)

        Parameter
            desperate (bool)

        Rückgabetyt
            Iterable[Any]

try_get_materialization_for_node(node)

        Parameter
            node (Node)

property has_result

property result: Sequence[Any]

property problems: Sequence[Exception]

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

        Parameter
            • node (Node)
            • behaviors (Iterable[annize.project.materializer.behaviors.Behavior])

```

**materialize()**

**Rückgabetyt**

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

**\_early\_materialize**(*node*, *early\_materialization\_store*)

**\_materialize\_hlp\_childobjs**(*node*, *materialization\_store*, *desperate*)

**Parameter**

- **node** (*Node*)
- **materialization\_store** (*dict*)
- **desperate** (*bool*)

**Rückgabetyt**

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

**\_\_early\_materialization\_for\_node**(*node*, *early\_materialization\_store*)

**Parameter**

- **node** (*Node*)
- **early\_materialization\_store** (*dict*)

**Rückgabetyt**

*EarlyNodeMaterialization*

**\_\_materialization\_for\_node**(*node*, *materialization\_store*)

**Parameter**

- **node** (*Node*)
- **materialization\_store** (*dict*)

**Rückgabetyt**

*NodeMaterialization*

**\_\_materialize**(*node*, *materialization\_store*, *desperate*)

**Parameter**

- **node** (*Node*)
- **materialization\_store** (*dict*)
- **desperate** (*bool*)

**Rückgabetyt**

*None*

**\_\_erroneous\_nodes**(*old\_erroneous\_nodes*)

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

Bases: *Exception*

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

Bases: *InternalError*

**Parameter**

**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*)

**Parameter**

- **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*)

**Parameter**

- **of\_type** (*type*)
- **args** (*Iterable*)
- **kwargs** (*dict*)

**exception** annize.project.materializer.object\_factory.MultipleValuesForSingleArgumentError(*arg\_name*)

Bases: TypeError

**Parameter**

**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*)

**Parameter**

**topnode** ([Node](#))

**Rückgabety**

[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*)

Parameter

**name** (*str*)

Rückgabotyp

*Any* | None

**abstractmethod** all\_available\_feature\_names()

Rückgabotyp

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*)

Parameter

**package\_name** (*str*)

Rückgabotyp

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.

Parameter

**for\_type** (*type*) – The type to gather information for.

**Rückgabotyp**  
*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!

**Parameter**  
**for\_type** (*Callable*) – The type to gather constructor parameter information for.

**Rückgabotyp**  
*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.

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

**Rückgabotyp**  
*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).

**Parameter**

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

**Rückgabotyp**  
*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.

**Parameter**

- **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).

**Rückgabotyp**  
*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\(\)`](#).

**Parameter**

- **child\_type** (*type*) – The child type.
- **parent\_type** (*type*) – The parent type.

**Rückgabetyt**

*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.

**Parameter**

- **child\_type** (*type*) – The child type.
- **parent\_type** (*type*) – The parent type.

**Rückgabetyt**

*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.

**Parameter**

- **type** – The type.
- **with\_parameters** (*bool*) – Whether to include documentation for its constructor parameters as well.
- **type\_** (*type*)

**Rückgabetyt**

*str*

**\_possible\_argument\_names\_for\_child\_in\_parent**(*child\_type*, *parent\_type*, *parent\_parameter\_info*)

**Parameter**

- **child\_type** (*type*)
- **parent\_type** (*type*)
- **parent\_parameter\_info** (*Mapping*[*str*, [`TypeInfo`](#)])

**Rückgabetyt**

*Sequence*[*str*]

**\_\_type\_info**(*for\_type*, *as\_optional=False*)

**Parameter**

- **for\_type** (*type*)
- **as\_optional** (*bool*)

**Rückgabetyt**

[`TypeInfo`](#)

```

__type_documentation__summary()

    Parameter
        type_doc (str)

    Rückgabetyt
        str

__type_documentation__parameter(param_name)

    Parameter
        • type_ (type)
        • param_name (str)

    Rückgabetyt
        str

__type_documentation__parameter_block(param_name)

    Parameter
        • func_doc (str)
        • param_name (str)

    Rückgabetyt
        Sequence[str]

__type_documentation__parameter_from_lines()

    Parameter
        param_doc_lines (Sequence[str])

    Rückgabetyt
        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_)

        Parameter
            type_ (type)

        Rückgabetyt
            bool

    abstractmethod matches_inner_type(type_)

        Parameter
            type_ (type)

        Rückgabetyt
            bool

```

```
    _abc_impl = <_abc._abc_data object>

class _ScalarTypeInfo(name, type_, is_optional)
    Bases: TypeInfo
        Parameter
            • 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
        Parameter
            • 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
        Parameter
            • 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.
```



**Parameter**

**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.

**Parameter**

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

**Rückgabetyt**

*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.

**Parameter**

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

**Rückgabetyt**

*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.

**Parameter**

**for\_type** (*type*) – The type to gather information for.

**Rückgabetyt**

*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.

**Parameter**

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

**Rückgabetyt**

*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.

**Parameter**

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

**Rückgabetyt**

*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.

**Parameter**

- **node** (*Node*) – The node.
- **parameter\_name** (*str*) – The parameter name.

**Rückgabetyt**

*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.

**Parameter**

**node** (*ArgumentNode*) – The node.

**Rückgabetyt**

*Sequence*[*ArgumentNode*]

**\_\_all\_creatable\_types**(*\**, *with\_value\_types=True*)

**Parameter**

**with\_value\_types** (*bool*)

**Rückgabetyt**

*Sequence*[*CreatableTypeInfo*]

**\_\_type\_full\_name**()

**Parameter**

**for\_type** (*type*)

**Rückgabetyt**

*str*

**class ArgumentMatching**(*arg\_name*, *nodes*, *allows\_multiple\_args*)

Bases: *object*

**Parameter**

- **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*

**Parameter**

**all\_matchings** (*Iterable*[*FullInspector.ArgumentMatching*])

```

matching_by_arg_name(arg_name)
    Parameter
        arg_name (str)
    Rückgabotyp
        ArgumentMatching | None

all()
    Rückgabotyp
        Sequence[ArgumentMatching]

class CreatableTypeInfo(name, type_, is_optional, feature_name, type_short_name)
    Bases: ScalarTypeInfo
    Parameter
        • 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
    Parameter
        • 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\(\)](#).

### Parameter

- **project\_location** (str | Path) – A path to somewhere inside an Annize project.
- **inspector** (FullInspector | None) – The custom project inspector to use.

### Rückgabotyp

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`.

**Parameter**

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

**Rückgabotyp**

*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`).

**Parameter**

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

**Rückgabotyp**

*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`).

**Parameter**

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

**Rückgabotyp**

*Path* | `None`

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

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

**Parameter**

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

**Rückgabotyp**

`bool`

## **annize.ui package**

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

**Parameter**

**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*)

**Parameter**

- **message** (*str*)
- **answers** (*list[str]*)
- **config\_key** (*str | None*)

**Rückgabetyp**

*int*

**abstractmethod input\_dialog**(*question, suggested\_answer, config\_key*)

**Parameter**

- **question** (*str*)
- **suggested\_answer** (*str*)
- **config\_key** (*str | None*)

**Rückgabetyp**

*str | None*

**abstractmethod choice\_dialog**(*question, choices, config\_key*)

**Parameter**

- **question** (*str*)
- **choices** (*list[str]*)
- **config\_key** (*str | None*)

**Rückgabetyp**

*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)

    Parameter
        context (RunContext)

    Rückgabetyt
        Sequence[tuple[int, UserFeedbackController]]

annize.user_feedback._controllers_for_context(context)

    Parameter
        context (RunContext)

    Rückgabetyt
        list[UserFeedbackController]

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

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

    Rückgabetyt
        None

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

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

    Rückgabetyt
        int

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

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

```

**Rückgabotyp**

str | None

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

- **message** (*TrStrOrStr*)
- **choices** (*Iterable[TrStrOrStr]*)
- **config\_key** (*str* | *None*)

**Rückgabotyp**

int | None

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

- **config\_key** (*str*)
- **value** (*Any*)

**Rückgabotyp**

None

`__get_answer(config_key)`**Parameter****config\_key** (*str*)**Rückgabotyp***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)
```

**Parameter**

**only\_documentation** (*bool*)

**Rückgabotyp**

*ArgumentParser*

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

Bases: object

**Parameter**

- **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)
```

**Parameter**

- **destination** (*dict*)
- **with\_answers\_from\_json\_files** (*Iterable[str]*)

```
classmethod __answers_from_json_strings(destination, with_answers_from_json_strings)
```

**Parameter**

- **destination** (*dict*)
- **with\_answers\_from\_json\_strings** (*Iterable[str]*)

```
classmethod __answers_from_single_answers(destination, with_answers)
```

**Parameter**

- **destination** (*dict*)
- **with\_answers** (*Iterable[Tuple[str, str]]*)

```
do(task_name, **_)
```

**Parameter**

**task\_name** (*str*)

```
studio(**_)
```

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

**Parameter**

**debug** (*bool*)



### 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



## Sonderzeichen

- `_COMMON_NAMESPACE_POSTFIX` (Attribut von `annize.project.feature_loader.DefaultFeatureLoader`), 72
- `_CreateObjectHelper` (Klasse in `annize.project.materializer.object_factory`), 71
- `_CreateObjectHelper__fill_empty_lists()` (statische Methode von `annize.project.materializer.object_factory._CreateObjectHelper`), 71
- `_CreateObjectHelper__fill_unspecified_optionals()` (statische Methode von `annize.project.materializer.object_factory._CreateObjectHelper`), 71
- `_CreateObjectHelper__put_item_into_kwargs()` (statische Methode von `annize.project.materializer.object_factory._CreateObjectHelper`), 71
- `_CreateObjectHelper__shift_args_to_kwargs()` (statische Methode von `annize.project.materializer.object_factory._CreateObjectHelper`), 71
- `_FEATURES_NAMESPACE` (Attribut von `annize.project.feature_loader.DefaultFeatureLoader`), 72
- `_FixedTrStr` (Klasse in `annize.i18n`), 48
- `_FormattedTrStr` (Klasse in `annize.i18n`), 49
- `_IS_TOPLEVEL_OBJECT__METADATA_KEY` (Attribut von `annize.flow.run_context.RunContext`), 28
- `_NAMES__METADATA_KEY` (Attribut von `annize.flow.run_context.RunContext`), 28
- `_ProjectDefinedTranslationProvider` (Klasse in `annize.features.i18n.common`), 24
- `_ProjectDefinedTranslationProvider__translations_for_string_name()` (Methode von `annize.features.i18n.common._ProjectDefinedTranslationProvider`), 25
- `_TContent` (Attribut von `annize.fs.ext.DynamicFile`), 41
- `_TStaticContent` (Attribut von `annize.fs.ext.DynamicFile`), 40
- `_TransferHelper__transfer_piece()` (statische Methode von `annize.fs.Path._TransferHelper`), 39
- `__all_creatable_types()` (Methode von `annize.project.inspector.FullInspector`), 78
- `__answers_from_json_files()` (Klassenmethode von `annize.annize_cli.Commands`), 85
- `__answers_from_json_strings()` (Klassenmethode von `annize.annize_cli.Commands`), 85
- `__answers_from_single_answers()` (Klassenmethode von `annize.annize_cli.Commands`), 85
- `__best_system_locale()` (Methode von `annize.i18n.Culture`), 45
- `__changed__call__handlers()` (Methode von `annize.project.Node`), 54
- `__changed__child_added()` (Methode von `annize.project.Node`), 53
- `__changed__child_removed()` (Methode von `annize.project.Node`), 53
- `__changed__property_changed()` (Methode von `annize.project.Node`), 53
- `__cleanup()` (Methode von `annize.fs.ext.FreshTempDirectory`), 40
- `__compacted_changes()` (Methode von `annize.project.ProjectNode`), 57
- `__context_catch_exceptions()` (Methode von `annize.project.materializer.behaviors.feature_unavailable.FeatureUnavailable`), 68
- `__context_skip_node_feature_ignore_list()` (Methode von `annize.project.materializer.behaviors.feature_unavailable.FeatureUnavailable`), 68
- `__current_system_locale_setup()` (Methode von `annize.i18n.Culture`), 45
- `__description()` (Methode von `annize.project.Node`), 53
- `__do_run()` (Methode von `annize.flow.runner.Runner`), 35
- `__does_exclude()` (Methode von `annize.project.ProjectNode`), 57

`ze.features.files.common.Exclude)`, 22  
`__early_materialization_for_node()` (Methode von `anni-ze.project.materializer.core.ProjectMaterializer`), 70  
`__erroneous_nodes()` (Methode von `anni-ze.project.materializer.core.ProjectMaterializer`), 70  
`__find_feature_modules_in_package()` (Methode von `anni-ze.project.feature_loader.DefaultFeatureLoader`), 72  
`__get_answer()` (Methode von `anni-ze.user_feedback.static.StaticUserFeedbackController`), 84  
`__get_normed_value()` (Methode von `anni-ze.data.color.Color`), 16  
`__handle_changed()` (Methode von `anni-ze.project.ProjectNode`), 57  
`__html_color_spec__part()` (Methode von `anni-ze.data.color.Color`), 17  
`__initial_cwd` (Attribut von `anni-ze.annize_cli.Commands`), 85  
`__is_inverse_of()` (Methode von `anni-ze.project.ProjectNode`), 57  
`__materialization_for_node()` (Methode von `anni-ze.project.materializer.core.ProjectMaterializer`), 70  
`__materialize()` (Methode von `anni-ze.project.materializer.core.ProjectMaterializer`), 70  
`__object_metadata_dict()` (Methode von `anni-ze.flow.run_context.RunContext`), 31  
`__object_raw_name()` (Methode von `anni-ze.flow.run_context.RunContext`), 31  
`__put_object()` (Methode von `anni-ze.flow.run_context.RunContext`), 31  
`__reset_change_history()` (Methode von `anni-ze.project.ProjectNode`), 57  
`__set_env__var()` (Methode von `annize.i18n.Culture`), 45  
`__set_success_state()` (Methode von `anni-ze.flow.runner.Runner`), 35  
`__set_system_locale_setup()` (Methode von `annize.i18n.Culture`), 45  
`__set_tasks()` (Methode von `anni-ze.flow.runner.Runner`), 35  
`__shorten()` (Methode von `anni-ze.project.ScalarValueNode`), 59  
`__transfer_filter_for_exclude()` (Methode von `annize.features.files.common.Directory`), 24  
`__type_documentation__parameter()` (Methode von `annize.project.inspector.BasicInspector`), 75  
`__type_documentation__parameter_block()` (Methode von `anni-ze.project.inspector.BasicInspector`), 75  
`__type_documentation__summary()` (Methode von `annize.project.inspector.BasicInspector`), 74  
`__type_full_name()` (Methode von `anni-ze.project.inspector.FullInspector`), 78  
`__type_info()` (Methode von `anni-ze.project.inspector.BasicInspector`), 74  
`_abc_impl` (Attribut von `anni-ze.data.version.ConcatenatedVersionPatternPart`), 20  
`_abc_impl` (Attribut von `anni-ze.data.version.NumericVersionPatternPart`), 19  
`_abc_impl` (Attribut von `anni-ze.data.version.OptionalVersionPatternPart`), 20  
`_abc_impl` (Attribut von `anni-ze.data.version.SeparatorVersionPatternPart`), 20  
`_abc_impl` (Attribut von `anni-ze.data.version.VersionPatternPart`), 19  
`_abc_impl` (Attribut von `anni-ze.features.i18n.common.String`), 25  
`_abc_impl` (Attribut von `anni-ze.features.i18n.common._ProjectDefinedTranslationProvider`), 25  
`_abc_impl` (Attribut von `annize.flow.runner.Runner`), 35  
`_abc_impl` (Attribut von `anni-ze.i18n.GettextTranslationProvider`), 47  
`_abc_impl` (Attribut von `annize.i18n.ProvidedTrStr`), 47  
`_abc_impl` (Attribut von `annize.i18n.TrStr`), 43  
`_abc_impl` (Attribut von `anni-ze.i18n.TranslationProvider`), 44  
`_abc_impl` (Attribut von `annize.i18n._FixedTrStr`), 48  
`_abc_impl` (Attribut von `annize.i18n._FormattedTrStr`), 49  
`_abc_impl` (Attribut von `annize.project.ArgumentNode`), 58  
`_abc_impl` (Attribut von `annize.project.FileNode`), 58  
`_abc_impl` (Attribut von `anni-ze.project.IgnoreUnavailableFeatureNode`), 60  
`_abc_impl` (Attribut von `annize.project.Node`), 55  
`_abc_impl` (Attribut von `annize.project.ObjectNode`), 58  
`_abc_impl` (Attribut von `annize.project.ProjectNode`), 57  
`_abc_impl` (Attribut von `annize.project.ReferenceNode`), 59  
`_abc_impl` (Attribut von `anni-ze.project.ScalarValueNode`), 59  
`_abc_impl` (Attribut von `anni-`

`ze.project.feature_loader.DefaultFeatureLoader),` 84  
`_abc_impl (Attribut von anni-`  
`ze.project.feature_loader.FeatureLoader),`  
`_abc_impl (Attribut von anni-`  
`ze.project.file_formats.FileFormat),` 62  
`_abc_impl (Attribut von anni-`  
`ze.project.file_formats.FileFormat.Marshaler),`  
`_abc_impl (Attribut von anni-`  
`ze.project.file_formats.FileFormat.NullMarshaler),`  
`_abc_impl (Attribut von anni-`  
`ze.project.inspector.BasicInspector.TypeInfo),`  
`_abc_impl (Attribut von anni-`  
`ze.project.inspector.BasicInspector._ListTypeInfo),`  
`_abc_impl (Attribut von anni-`  
`ze.project.inspector.BasicInspector._ScalarTypeInfo),`  
`_abc_impl (Attribut von anni-`  
`ze.project.inspector.BasicInspector._UnionTypeInfo),`  
`_abc_impl (Attribut von anni-`  
`ze.project.inspector.FullInspector.CreatableTypeInfo),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.Behavior),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.argument.ArgumentBehavior),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.argument.AssociateArgumentBehavior),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.basket.BasketBehavior),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.block.BlockBehavior),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.feature_unavailable.FeatureUnavailableBehavior),`  
`_abc_impl (Attribut von anni-`  
`ze.project.materializer.behaviors.reference.ReferenceBehavior),`  
`_abc_impl (Attribut von anni-`  
`ze.user_feedback.UserFeedbackController),`  
`_abc_impl (Attribut von anni-`  
`ze.user_feedback.static.StaticUserFeedbackController),`

`_add_controller_to_context() (im Modul anni-`  
`ze.user_feedback),` 83  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.FileNode),` 57  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.IgnoreUnavailableFeatureNode),` 60  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.Node),` 52  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.ObjectNode),` 58  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.ProjectNode),` 56  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.ReferenceNode),` 59  
`_allowed_child_types() (Klassenmethode von anni-`  
`ze.project.ScalarValueNode),` 59  
`_annize_user_interaction_culture() (im Modul`  
`annize.i18n),` 48  
`_clone_early() (Methode von anni-`  
`ze.project.FileNode),` 57  
`_clone_early() (Methode von annize.project.Node),`  
`_clone_early() (Methode von anni-`  
`ze.project.ProjectNode),` 56  
`_clone_late() (Methode von annize.project.Node),` 52  
`_clone_late() (Methode von anni-`  
`ze.project.ProjectNode),` 56  
`_controllers_for_context() (im Modul anni-`  
`ze.user_feedback),` 83  
`_controllers_tuples_for_context() (im Modul`  
`annize.user_feedback),` 83  
`_current_translation_providers_lists() (im`  
`Modul annize.i18n),` 48  
`_early_materialize() (Methode von anni-`  
`ze.project.materializer.core.ProjectMaterializer),`  
`_get_data() (im Modul annize.features.base),` 27  
`_last_resort_culture (in Modul annize.i18n),` 47  
`_materialize_hlp_childobjs() (Methode von anni-`  
`ze.project.materializer.core.ProjectMaterializer),`  
`_node_clone_link() (im Modul anni-`  
`ze.project.materializer),` 64  
`_path() (Methode von anni-`  
`ze.features.files.common.Directory),` 23  
`_path() (Methode von anni-`  
`ze.features.files.common.FsEntry),` 21  
`_path() (Methode von anni-`  
`ze.features.files.common.MachineRootDirectory),`  
`_path() (Methode von anni-`  
`ze.features.files.common.ProjectDirectory),` 24

\_path() (Methode von *annize.fs.Path*), 36  
 \_path() (Methode von *annize.fs.ext.DynamicFile*), 41  
 \_possible\_argument\_names\_for\_child\_in\_parent() (Methode von *annize.project.inspector.BasicInspector*), 74  
 \_property\_changed() (Methode von *annize.project.Node*), 53  
 \_setup\_logging() (im Modul *annize.annize\_cli*), 85  
 \_str\_helper() (Methode von *annize.project.ArgumentNode*), 58  
 \_str\_helper() (Methode von *annize.project.FileNode*), 57  
 \_str\_helper() (Methode von *annize.project.IgnoreUnavailableFeatureNode*), 60  
 \_str\_helper() (Methode von *annize.project.Node*), 53  
 \_str\_helper() (Methode von *annize.project.ObjectNode*), 58  
 \_str\_helper() (Methode von *annize.project.ProjectNode*), 56  
 \_str\_helper() (Methode von *annize.project.ReferenceNode*), 59  
 \_str\_helper() (Methode von *annize.project.ScalarValueNode*), 59  
 \_translate\_from\_clone() (im Modul *annize.project.materializer*), 64  
 \_translation\_for\_culture() (Methode von *annize.i18n.ProvidedTrStr*), 47  
 \_translation\_for\_culture() (Methode von *annize.i18n.TrStr*), 42  
 \_translation\_for\_culture() (Methode von *annize.i18n.FixedTrStr*), 48  
 \_translation\_for\_culture() (Methode von *annize.i18n.FormatedTrStr*), 49  
 \_translation\_provider() (im Modul *annize.features.i18n.common*), 25  
 \_translation\_providers() (im Modul *annize.i18n*), 48  
 add\_translation\_provider() (im Modul *annize.i18n*), 44  
 add\_translations() (Methode von *annize.features.i18n.common.\_ProjectDefinedTranslationProvider*), 24  
 all() (Methode von *annize.project.inspector.FullInspector.ArgumentMatchings*), 79  
 all\_available\_feature\_names() (Methode von *annize.project.feature\_loader.DefaultFeatureLoader*), 72  
 all\_available\_feature\_names() (Methode von *annize.project.feature\_loader.FeatureLoader*), 72  
 all\_file\_format\_names() (im Modul *annize.project.file\_formats*), 62  
 all\_named\_nodes() (Methode von *annize.project.inspector.BasicInspector*), 73  
 allows\_multiple\_args (im Modul *annize.project.inspector.BasicInspector.\_ListTypeInfo* property), 76  
 allows\_multiple\_args (im Modul *annize.project.inspector.BasicInspector.\_ScalarTypeInfo* property), 76  
 allows\_multiple\_args (im Modul *annize.project.inspector.BasicInspector.TypeInfo* property), 75  
 allows\_multiple\_args (im Modul *annize.project.inspector.FullInspector.ArgumentMatching* 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  
 add\_answer() (Methode von *annize.user\_feedback.static.StaticUserFeedbackController*), 84  
 add\_change() (Methode von *annize.project.file\_formats.FileFormat.Marshaler*), 60  
 add\_change() (Methode von *annize.project.file\_formats.FileFormat.NullMarshaler*), 61  
 add\_change\_handler() (Methode von *annize.project.Node*), 51  
 add\_object() (im Modul *annize.flow.run\_context*), 33  
 add\_object() (Methode von *annize.flow.run\_context.RunContext*), 30



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\_unavailable  
     module, 67  
 annize.project.materializer.behaviors.reference\_behavior  
     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 (in *Module annize.project.ProjectNode* property), 55  
 ANNIZE\_CONFIG\_DIRECTORY\_\_NAME (Attribut von *annize.flow.run\_context.RunContext*), 29  
 annize\_user\_interaction\_culture (in *Module annize.i18n*), 49  
 app() (in *Module annize.ui*), 80  
 append\_child() (Methode von *annize.project.Node*), 52  
 append\_to (in *annize.project.ArgumentNode* property), 58  
 arg\_name (in *annize.project.ArgumentNode* property), 58  
 arg\_name (in *annize.project.inspector.FullInspector.ArgumentMatching* property), 78  
 argument\_type\_for\_argument\_node() (Methode von *annize.project.inspector.FullInspector*), 77  
 ArgumentBehavior (Klasse in *annize.project.materializer.behaviors.argument*), 65  
 ArgumentNode (Klasse in *annize.project*), 58  
 AssociateArgumentNodeBehavior (Klasse in *annize.project.materializer.behaviors.argument*), 66

## B

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

## C

changes() (Methode von *annize.project.ProjectNode*), 56  
 child\_node(*annize.project.Node.\_\_ChildrenListChangeEvent* property), 54  
 child\_position (*annize.project.Node.\_\_ChildrenListChangeEvent* property), 54  
 children (*annize.project.Node* property), 51  
 children() (Methode von *annize.fs.Path*), 36  
 ChildrenNotMaterializableError, 70  
 choice\_dialog() (im Modul *annize.user\_feedback*), 84  
 choice\_dialog() (Methode von *annize.user\_feedback.NullUserFeedbackController*), 83  
 choice\_dialog() (Methode von *annize.user\_feedback.static.StaticUserFeedbackController*), 84  
 choice\_dialog() (Methode von *annize.user\_feedback.UserFeedbackController*), 82  
 clone() (Methode von *annize.project.Node*), 52  
 Color (Klasse in *annize.data.color*), 15  
 Commands (Klasse in *annize.annize\_cli*), 85  
 ConcatenatedVersionPatternPart (Klasse in *annize.data.version*), 20  
 content() (im Modul *annize.fs*), 39  
 copy\_to() (Methode von *annize.fs.Path*), 37  
 creatable\_type\_info() (Methode von *annize.project.inspector.FullInspector*), 77  
 creatable\_types\_for\_node\_argument() (Methode von *annize.project.inspector.FullInspector*), 77  
 creatables\_for\_node\_argument() (Methode von *annize.project.inspector.FullInspector*), 77  
 create\_new() (im Modul *annize.project*), 50  
 create\_object() (im Modul *annize.project.materializer.object\_factory*), 71  
 create\_object() (statische Methode von *annize.project.materializer.object\_factory.CreateObjectHelper*), 71  
 ctime() (Methode von *annize.fs.Path*), 36  
 Culture (Klasse in *annize.features.i18n.common*), 25  
 Culture (Klasse in *annize.i18n*), 44  
 Culture.\_TSystemLocaleSetup (Klasse in *annize.i18n*), 46  
 culture\_by\_spec() (im Modul *annize.i18n*), 47  
 culture\_list() (Methode von *annize.i18n.Culture*), 45  
 current() (im Modul *annize.flow.run\_context*), 32  
 current\_culture() (im Modul *annize.i18n*), 46

## D

Data (Klasse in *annize.features.base*), 26  
 DateTime (Klasse in *annize.features.base*), 27

DefaultFeatureLoader (Klasse in *annize.project.feature\_loader*), 72  
 description() (Methode von *annize.project.Node*), 52  
 deserialize\_node() (Methode von *annize.project.file\_formats.FileFormat*), 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 (Klasse in *annize.features.files.common*), 23  
 DirectoryPart (Klasse in *annize.features.files.common*), 22  
 do() (Methode von *annize.annize\_cli.Commands*), 85  
 does\_exclude() (Methode von *annize.features.files.common.Exclude*), 22  
 does\_exclude() (Methode von *annize.features.files.common.ExcludeAllBut*), 22  
 dynamic\_file() (im Modul *annize.fs*), 39  
 DynamicFile (Klasse in *annize.fs.ext*), 40

## E

early\_materialize\_children() (Methode von *annize.project.materializer.core.EarlyNodeMaterialization*), 69  
 early\_node\_context() (Methode von *annize.project.materializer.behaviors.Behavior*), 64  
 early\_node\_context() (Methode von *annize.project.materializer.behaviors.block.BlockBehavior*), 67  
 early\_node\_context() (Methode von *annize.project.materializer.behaviors.feature\_unavailable.FeatureUnavailable*), 67  
 EarlyNodeMaterialization (Klasse in *annize.project.materializer.core*), 68  
 english\_lang\_name (*annize.i18n.Culture* property), 44  
 erroneous\_nodes() (Methode von *annize.project.materializer.MaterializationResult*), 64  
 errors\_for\_node() (Methode von *annize.project.materializer.MaterializationResult*), 64  
 Exclude (Klasse in *annize.features.files.common*), 21  
 ExcludeAllBut (Klasse in *annize.features.files.common*), 22  
 excludes (*annize.features.files.common.Directory* property), 23  
 excludes (*annize.features.files.common.DirectoryPart* property), 23

`explicit_only` (Attribut von `annize.object.config.InnerParameterConfig`), 50  
`explicit_only` (Attribut von `annize.object.config.ParameterConfig`), 50  
`explicit_only()` (im Modul `annize.object`), 49

## F

`FAIL` (Attribut von `annize.project.ReferenceNode.OnUnresolvableAction`), 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` (Klasse in `annize.project.feature_loader`), 71  
`FeatureUnavailableBehavior` (Klasse in `annize.project.materializer.behaviors.feature_unavailable`), 67  
`FeatureUnavailableError`, 60  
`file` (`annize.project.Node` property), 51  
`File` (Klasse in `annize.features.files.common`), 21  
`file_format()` (im Modul `annize.project.file_formats`), 62  
`file_size()` (Methode von `annize.fs.Path`), 37  
`FileFormat` (Klasse in `annize.project.file_formats`), 60  
`FileFormat.Marshaler` (Klasse in `annize.project.file_formats`), 60  
`FileFormat.NullMarshaler` (Klasse in `annize.project.file_formats`), 61  
`FileNode` (Klasse in `annize.project`), 57  
`FileSystemContent` (Klasse in `annize.fs`), 35  
`FirstOrDefault` (Klasse in `annize.features.base`), 27  
`format()` (Methode von `annize.i18n.TrStr`), 42  
`fresh_temp_directory()` (im Modul `annize.fs`), 39  
`FreshTempDirectory` (Klasse in `annize.fs.ext`), 40  
`friendly_join_string_list()` (im Modul `annize.i18n`), 48  
`from_iso_639_1_lang_code()` (statische Methode von `annize.i18n.Culture`), 44  
`FsEntry` (Klasse in `annize.features.files.common`), 21  
`full_name` (`annize.i18n.Culture` property), 45  
`FullInspector` (Klasse in `annize.project.inspector`), 76  
`FullInspector.ArgumentMatching` (Klasse in `annize.project.inspector`), 78  
`FullInspector.ArgumentMatchings` (Klasse in `annize.project.inspector`), 78  
`FullInspector.CreatableInfo` (Klasse in `annize.project.inspector`), 79  
`FullInspector.CreatableTypeInfo` (Klasse in `annize.project.inspector`), 79

## G

`GenerateMOs` (Klasse in `annize.features.i18n.gettext`), 26  
`get_selected_task()` (Methode von `annize.flow.runner.Runner`), 34  
`get_success_state()` (Methode von `annize.flow.runner.Runner`), 35  
`get_tasks()` (Methode von `annize.flow.runner.Runner`), 34  
`GettextTranslationProvider` (Klasse in `annize.i18n`), 46  
`GettextTranslationProvider._NoneTranslations` (Klasse in `annize.i18n`), 47  
`green` (`annize.data.color.Color` property), 16

## H

`homepage_url` (`annize.project.materializer.core.NodeMaterialization` property), 69  
`homepage_url` (`annize.features.base.Data` property), 26  
`homepage_url()` (im Modul `annize.features.base`), 27  
`html_color_spec` (`annize.data.color.Color` property), 16  
`hue` (`annize.data.color.Color` property), 16

## I

`IgnoreUnavailableFeatureNode` (Klasse 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  
`InnerParameterConfig` (Klasse in `annize.object.config`), 50  
`input_dialog()` (im Modul `annize.user_feedback`), 83  
`input_dialog()` (Methode von `annize.user_feedback.NullUserFeedbackController`), 83  
`input_dialog()` (Methode von `annize.user_feedback.static.StaticUserFeedbackController`), 84  
`input_dialog()` (Methode von `annize.user_feedback.UserFeedbackController`), 82  
`insert_child()` (Methode von `annize.project.Node`), 51  
`insert_child()` (Methode von `annize.project.ProjectNode`), 55  
`InternalError`, 70  
`is_advanced` (`annize.features.task.Task` property), 28  
`is_finished()` (Methode von `annize.flow.runner.Runner`), 35

is\_friendly\_name() (im Modul *annize.flow.run\_context*), 33  
 is\_friendly\_name() (Methode von *annize.flow.run\_context.RunContext*), 30  
 is\_optional(*annize.project.inspector.BasicInspector.\_ScalarTypeInfo* property), 76  
 is\_optional(*annize.project.inspector.BasicInspector.TypeInfo* property), 75  
 is\_toplevel\_object() (im Modul *annize.flow.run\_context*), 33  
 is\_toplevel\_object() (Methode von *annize.flow.run\_context.RunContext*), 30  
 is\_valid\_annize\_configuration\_file\_name() (im Modul *annize.project.loader*), 80  
 iso\_639\_1\_language\_code (*annize.i18n.Culture* property), 45  
**K**  
 kwargs (*annize.project.inspector.FullInspector.CreatableInfo* property), 79  
**L**  
 LANGUAGE (Attribut von *annize.i18n.Culture.\_TSystemLocaleSetup*), 46  
 LC\_ALL (Attribut von *annize.i18n.Culture.\_TSystemLocaleSetup*), 46  
 lightness (*annize.data.color.Color* property), 16  
 load() (im Modul *annize.project*), 50  
 load() (statische Methode von *annize.project.ProjectNode*), 56  
 load\_feature() (Methode von *annize.project.feature\_loader.DefaultFeatureLoader*), 72  
 load\_feature() (Methode von *annize.project.feature\_loader.FeatureLoader*), 72  
 load\_file\_node() (Methode von *annize.project.file\_formats.FileFormat*), 61  
 load\_project() (im Modul *annize.project.loader*), 79  
 load\_project() (im Modul *annize.project.loader*), 79  
 long\_description (*annize.features.base.Data* property), 26  
 long\_description() (im Modul *annize.features.base*), 27  
**M**  
 MachineRootDirectory (Klasse in *annize.features.files.common*), 24  
 main() (im Modul *annize.annize\_cli*), 84  
 mark\_object\_as\_toplevel() (Methode von *annize.flow.run\_context.RunContext*), 30  
 marshaler (*annize.project.FileNode* property), 57  
 match\_arguments() (Methode von *annize.project.inspector.FullInspector*), 77  
 matches\_inner\_type() (Methode von *annize.project.inspector.BasicInspector.\_ScalarTypeInfo*), 76  
 matches\_inner\_type() (Methode von *annize.project.inspector.BasicInspector.TypeInfo*), 75  
 matches\_type() (Methode von *annize.project.inspector.BasicInspector.\_ScalarTypeInfo*), 76  
 matches\_type() (Methode von *annize.project.inspector.BasicInspector.\_UnionTypeInfo*), 76  
 matches\_type() (Methode von *annize.project.inspector.BasicInspector.TypeInfo*), 75  
 matching\_by\_arg\_name() (Methode von *annize.project.inspector.FullInspector.ArgumentMatchings*), 78  
 MaterializationResult (Klasse in *annize.project.materializer*), 63  
 materialize() (im Modul *annize.project.materializer*), 64  
 materialize() (Methode von *annize.project.materializer.core.ProjectMaterializer*), 69  
 materialized\_children() (Methode von *annize.project.materializer.core.NodeMaterialization*), 69  
 materialized\_children\_tuples() (Methode von *annize.project.materializer.core.NodeMaterialization*), 69  
 MaterializerError, 60  
 message\_dialog() (im Modul *annize.user\_feedback*), 83  
 message\_dialog() (Methode von *annize.user\_feedback.NullUserFeedbackController*), 83  
 message\_dialog() (Methode von *annize.user\_feedback.static.StaticUserFeedbackController*), 84  
 message\_dialog() (Methode von *annize.user\_feedback.UserFeedbackController*), 82  
 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 (Klasse in annize.fs.ext), 41  
 move\_to() (Methode von annize.fs.Path), 38  
 mtime() (Methode von annize.fs.Path), 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() (Methode von annize.project.file\_formats.FileFormat), 61  
 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 (Klasse in annize.project), 50  
 Node.\_\_ChildrenListChangeEvent (Klasse in annize.project), 54  
 Node.ChangeEvent (Klasse in annize.project), 54  
 Node.ChildAddedEvent (Klasse in annize.project), 54  
 Node.ChildRemovedEvent (Klasse in annize.project), 54  
 Node.PropertyChangedEvent (Klasse in annize.project), 55  
 node\_by\_name() (Methode von annize.project.inspector.BasicInspector), 73  
 node\_context() (Methode von annize.project.materializer.behaviors.argument.ArgumentBehavior), 65  
 node\_context() (Methode von annize.project.materializer.behaviors.argument.AssociateArgumentNodeBehavior), 66  
 node\_context() (Methode von annize.project.materializer.behaviors.basket.BasketBehavior), 66  
 node\_context() (Methode von annize.project.materializer.behaviors.Behavior), 65  
 node\_context() (Methode von annize.project.materializer.behaviors.block.BlockBehavior), 67  
 node\_context() (Methode von annize.project.materializer.behaviors.feature\_unavailable.FeatureUnavailableBehavior), 67  
 node\_context() (Methode von annize.project.materializer.behaviors.reference.ReferenceBehavior), 68  
 NodeMaterialization (Klasse in annize.project.materializer.core), 69  
 nodes (annize.project.inspector.FullInspector.ArgumentMatching property), 78  
 NullUserFeedbackController (Klasse in annize.user\_feedback), 83  
 NumericVersionPatternPart (Klasse in annize.data.version), 19



## O

object\_by\_name() (im Modul *annize.flow.run\_context*), 32

object\_by\_name() (Methode von *annize.flow.run\_context.RunContext*), 29

object\_metadata() (im Modul *annize.flow.run\_context*), 33

object\_metadata() (Methode von *annize.flow.run\_context.RunContext*), 31

object\_name() (im Modul *annize.flow.run\_context*), 32

object\_name() (Methode von *annize.flow.run\_context.RunContext*), 29

object\_names() (im Modul *annize.flow.run\_context*), 32

object\_names() (Methode von *annize.flow.run\_context.RunContext*), 29

ObjectNode (Klasse in *annize.project*), 58

objects\_by\_type() (im Modul *annize.flow.run\_context*), 32

objects\_by\_type() (Methode von *annize.flow.run\_context.RunContext*), 30

objects\_for\_node() (Methode von *annize.project.materializer.MaterializationResult*), 64

old\_value (*annize.project.Node.PropertyChangedEvent* property), 55

on\_unresolvable (*annize.project.ReferenceNode* property), 59

OptionalVersionPatternPart (Klasse in *annize.data.version*), 20

OutOfContextError, 32

## P

parameter\_config() (im Modul *annize.object.config*), 49

parameter\_info() (Methode von *annize.project.inspector.BasicInspector*), 73

ParameterConfig (Klasse in *annize.object.config*), 49

parent (*annize.project.Node* property), 51

parser() (im Modul *annize.annize\_cli*), 84

ParserError, 60

parts (*annize.data.version.VersionPattern* property), 18

parts (*annize.features.files.common.Directory* property), 23

path (*annize.fs.ext.FreshTempDirectory* property), 40

path (*annize.project.FileNode* property), 57

Path (Klasse in *annize.fs*), 36

path() (Methode von *annize.fs.FilesystemContent*), 36

path() (Methode von *annize.fs.Path*), 36

Path.\_TransferHelper (Klasse in *annize.fs*), 38

Path.TransferFilters (Klasse in *annize.fs*), 38

Path.TransferFilters.And (Klasse in *annize.fs*), 38

pattern (*annize.data.version.Version* property), 17

possible\_append\_to\_targets\_for\_node() (Methode von *annize.project.inspector.FullInspector*), 78

possible\_argument\_infos\_for\_child\_in\_parent() (Methode von *annize.project.inspector.BasicInspector*), 74

possible\_argument\_names\_for\_child\_in\_parent() (Methode von *annize.project.inspector.BasicInspector*), 73

possible\_reference\_targets\_for\_node\_argument() (Methode von *annize.project.inspector.FullInspector*), 77

prepare() (Methode von *annize.flow.run\_context.RunContext*), 29

pretty\_project\_name (*annize.features.base.Data* property), 26

pretty\_project\_name() (im Modul *annize.features.base*), 27

problems (*annize.project.materializer.core.NodeMaterialization* property), 69

project (*annize.project.Node* property), 51

project\_annize\_config\_directory() (im Modul *annize.project.loader*), 80

project\_annize\_config\_main\_file() (im Modul *annize.project.loader*), 79

project\_cultures() (im Modul *annize.features.i18n.common*), 25

project\_directory (*annize.features.base.Data* property), 26

project\_directory() (im Modul *annize.features.base*), 27

project\_name (*annize.features.base.Data* property), 26

project\_name() (im Modul *annize.features.base*), 27

project\_root\_directory() (im Modul *annize.project.loader*), 80

ProjectCultures (Klasse in *annize.features.i18n.common*), 25

ProjectDirectory (Klasse in *annize.features.files.common*), 24

ProjectMaterializer (Klasse in *annize.project.materializer.core*), 69

ProjectNode (Klasse in *annize.project*), 55

property\_name (*annize.project.Node.PropertyChangedEvent* property), 55

ProvidedTrStr (Klasse in *annize.i18n*), 47

## R

readme\_pdf() (im Modul *annize.asset.data*), 15

red (*annize.data.color.Color* property), 16

reference\_key (*annize.project.ReferenceNode* property), 59

ReferenceBehavior (Klasse in *annize.project.materializer.behaviors.reference*),

- 68  
 ReferenceNode (Klasse in *annize.project*), 59  
 ReferenceNode.OnUnresolvableAction (Klasse 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() (im Modul *annize.project.file\_formats*), 62  
 relative\_path (annize.features.files.common.FsEntry property), 21  
 remove() (Methode von *annize.fs.Path*), 37  
 remove\_change\_handler() (Methode von *annize.project.Node*), 51  
 remove\_child() (Methode von *annize.project.Node*), 52  
 remove\_child() (Methode von *annize.project.ProjectNode*), 55  
 resolve\_appendtonodes() (im Modul *annize.project.materializer.preprocessors*), 71  
 resolve\_reference\_node() (Methode von *annize.project.inspector.BasicInspector*), 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.Materialization property), 63  
 run\_runner() (Methode von *annize.flow.runner.Runner*), 34  
 RunContext (Klasse in *annize.flow.run\_context*), 28  
 Runner (Klasse in *annize.flow.runner*), 34
- ## S
- sanitized\_project\_name() (im Modul *annize.features.base*), 27  
 saturation (annize.data.color.Color property), 16  
 save() (Methode von *annize.project.ProjectNode*), 55  
 save\_file\_node() (Methode von *annize.project.file\_formats.FileFormat.Marshaler*), 61  
 save\_file\_node() (Methode von *annize.project.file\_formats.FileFormat.NullMarshaler*), 61  
 ScalarValueNode (Klasse 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() (Methode von *annize.data.version.ConcatenatedVersionPatternPart*), 20  
 segments\_tuples\_to\_text() (Methode von *annize.data.version.NumericVersionPatternPart*), 19  
 segments\_tuples\_to\_text() (Methode von *annize.data.version.OptionalVersionPatternPart*), 20  
 segments\_tuples\_to\_text() (Methode von *annize.data.version.SeparatorVersionPatternPart*), 19  
 segments\_tuples\_to\_text() (Methode von *annize.data.version.VersionPattern*), 18  
 segments\_tuples\_to\_text() (Methode von *annize.data.version.VersionPatternPart*), 18  
 segments\_values (annize.data.version.Version property), 17  
 SeparatorVersionPatternPart (Klasse in *annize.data.version*), 19  
 serialize\_node() (Methode von *annize.project.file\_formats.FileFormat*), 62  
 serialize\_node() (Methode von *annize.project.file\_formats.FileFormat.Marshaler*), 61  
 serialize\_node() (Methode von *annize.project.file\_formats.FileFormat.NullMarshaler*), 61  
 set\_materialized\_result() (Methode von *annize.project.materializer.core.NodeMaterialization*), 69

set\_object\_metadata() (im Modul *annize.flow.run\_context*), 33  
 set\_object\_metadata() (Methode von *annize.flow.run\_context.RunContext*), 31  
 set\_object\_name() (im Modul *annize.flow.run\_context*), 32  
 set\_object\_name() (Methode von *annize.flow.run\_context.RunContext*), 29  
 set\_problems() (Methode von *annize.project.materializer.core.NodeMaterialization*), 69  
 set\_selected\_task() (Methode von *annize.flow.runner.Runner*), 34  
 show\_task\_chooser() (Methode von *annize.flow.runner.Runner*), 34  
 show\_task\_execution() (Methode von *annize.flow.runner.Runner*), 34  
 show\_task\_execution\_success() (Methode von *annize.flow.runner.Runner*), 34  
 SKIP (Attribut von *annize.project.ReferenceNode.OnUnresolvableAction*), 59  
 source\_path(*annize.features.files.common.DirectoryPart* property), 23  
 StaticUserFeedbackController (Klasse in *annize.user\_feedback.static*), 84  
 str\_to\_value() (Methode von *annize.data.version.ConcatenatedVersionPatternPart*), 20  
 str\_to\_value() (Methode von *annize.data.version.NumericVersionPatternPart*), 19  
 str\_to\_value() (Methode von *annize.data.version.OptionalVersionPatternPart*), 20  
 str\_to\_value() (Methode von *annize.data.version.SeparatorVersionPatternPart*), 19  
 str\_to\_value() (Methode von *annize.data.version.VersionPatternPart*), 18  
 String (Klasse in *annize.features.i18n.common*), 25  
 string\_name(*annize.i18n.ProvidedTrStr* property), 47  
 studio() (Methode von *annize.annize\_cli.Commands*), 85  
 summary(*annize.features.base.Data* property), 26  
 summary() (im Modul *annize.features.base*), 27

## T

target\_node(*annize.project.Node.ChangeEvent* property), 54  
 Task (Klasse in *annize.features.task*), 28  
 temp\_clone() (Methode von *annize.fs.Path*), 37  
 text(*annize.data.version.Version* property), 17  
 text\_to\_segments\_tuples() (Methode von *annize.data.version.VersionPattern*), 18  
 TextSource (Klasse in *annize.features.i18n.gettext*), 26  
 tr() (im Modul *annize.i18n*), 46  
 tr() (statische Methode von *annize.i18n.TrStr*), 42  
 transfer\_action\_copy() (statische Methode von *annize.fs.Path.\_TransferHelper*), 38  
 transfer\_action\_move() (statische Methode von *annize.fs.Path.\_TransferHelper*), 39  
 transfer\_to() (statische Methode von *annize.fs.Path.\_TransferHelper*), 38  
 translate() (im Modul *annize.i18n*), 43  
 translate() (Methode von *annize.features.i18n.common.\_ProjectDefinedTranslationProvider*), 24  
 translate() (Methode von *annize.i18n.GettextTranslationProvider*), 46  
 translate() (Methode von *annize.i18n.TranslationProvider*), 43  
 translate() (Methode von *annize.i18n.TrStr*), 42  
 TranslationProvider (Klasse in *annize.i18n*), 43  
 TranslationUnavailableError, 48  
 TrStr (Klasse in *annize.i18n*), 42  
 trstr() (im Modul *annize.i18n*), 43  
 try\_get\_materialization\_for\_node() (Methode von *annize.project.materializer.core.NodeMaterialization*), 69  
 TTransferFilter (Attribut von *annize.fs.Path*), 37  
 type(*annize.project.inspector.BasicInspector.\_ScalarTypeInfo* property), 76  
 type(*annize.project.inspector.BasicInspector.TypeInfo* property), 75  
 type\_documentation() (Methode von *annize.project.inspector.BasicInspector*), 74  
 type\_info(*annize.project.inspector.FullInspector.CreatableTypeInfo* property), 79  
 type\_info() (Methode von *annize.project.inspector.BasicInspector*), 72  
 type\_name(*annize.project.ObjectNode* property), 58  
 type\_short\_name(*annize.project.inspector.FullInspector.CreatableTypeInfo* property), 79

## U

undo\_changes() (Methode von *annize.project.ProjectNode*), 56  
 UnresolvableReferenceError, 60  
 UnsatisfiableUserFeedbackAttemptError, 83  
 unspecified\_culture (im Modul *annize.i18n*), 47  
 UpdatePOs (Klasse in *annize.features.i18n.gettext*), 26  
 UserFeedbackController (Klasse in *annize.user\_feedback*), 82



## V

`value` (*annize.project.ScalarValueNode* property), 59  
`Version` (*Klasse in annize.data.version*), 17  
`VersionPattern` (*Klasse in annize.data.version*), 17  
`VersionPatternPart` (*Klasse in annize.data.version*), 18

## W

`wait_finished()` (*Methode von annize.flow.runner.Runner*), 35  
`with_modified()` (*Methode von annize.data.color.Color*), 16  
`write_file()` (*Methode von annize.fs.Path*), 37