stable :

AndiEcker

Mar 19, 2024

CONTENTS

welcome to the documentation of the portions (app/service modules and sub-packages) of this freely extendable ae namespace (**PEP 420**).

CHAPTER

ONE

FEATURES AND USE-CASES

the portions of this namespace are simplifying your Python application or service in the areas/domains:

```
* data processing/validation
* file handling
* i18n (localization)
* configuration settings
* console
* logging
* database access
* networking
* multi-platform/-OS
* context help
* app tours
* user preferences (font size, color, theming, ...)
* QR codes
* sideloading
```

CHAPTER

TWO

SCREENSHOTS OF APPS DEVELOPED WITH AE PORTIONS

Fig. 2.1: GlslTest app demo

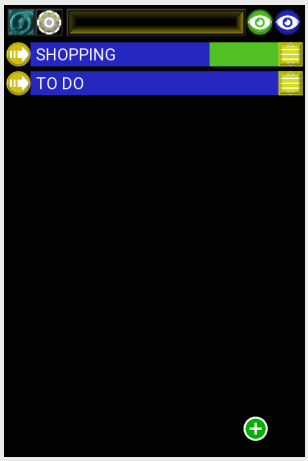

Fig. 2.2: kivy lisz app root list

Fig. 2.3: fruits sub-list

Fig. 2.4: using light theme

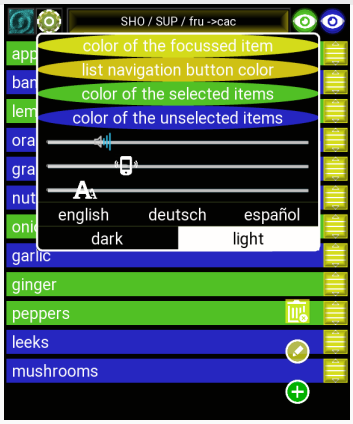

Fig. 2.5: lisz user preferences

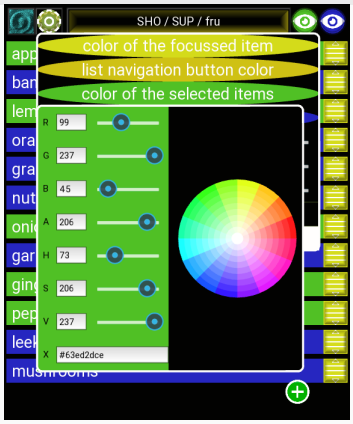

Fig. 2.6: kivy lisz color editor

Fig. 2.7: bigger font size

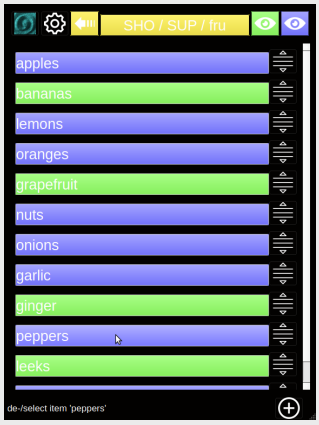

Fig. 2.8: enaml/qt lisz app

Fig. 2.9: light themed in landscape

CHAPTER

THREE

CODE MAINTENANCE GUIDELINES

3.1 portions code requirements

- pure python
- fully typed (**PEP 526**)
- fully *documented*
- 100 % test coverage
- multi thread save
- code checks (using pylint and flake8)

3.2 design pattern and software principles

- DRY - don't repeat yourself
- KIS - keep it simple

3.3 contributing

we want to make it as easy and fun as possible for you to contribute to this project.

3.3.1 reporting bugs

before you create a new issue, please check to see if you are using the latest version of this project; the bug may already be resolved.

also search for similar problems in the issue tracker; it may already be an identified problem.

include as much information as possible into the issue description, at least:

1. version numbers (of Python and any involved packages).
2. small self-contained code example that reproduces the bug.
3. steps to reproduce the error.
4. any traceback/error/log messages shown.

3.3.2 requesting new features

1. on the git repository host server create new issue, providing a clear and detailed explanation of the feature you want and why it's important to add.
2. if you are able to implement the feature yourself (refer to the *contribution steps* section below).

3.3.3 contribution steps

thanks for your contribution – we'll get your merge request reviewed. you could also review other merge requests, just like other developers will review yours and comment on them. based on the comments, you should address them. once the reviewers approve, the maintainers will merge.

before you start make sure you have a GitLab account.

contribution can be done either with the `git-repo-manager tool` or directly by using the `git` command and the `Gitlab` server.

using the git repository manager *grm*

1. fork and clone the repository of this project to your computer

   in your console change the working directory to your project's parent folder. then run the following command with the `new_feature_or_fix` part replaced by an appropriate branch name, describing shortly your contribution:

   ```
   grm -b new_feature_or_fix fork ae-group/ae_ae
   ```

   Note: the `grm fork` action will also add the forked repository as the remote `upstream` to your local repository.

   ---

   after the repository fork you change your current working directory to the new working tree root folder, created by the `grm fork` action, and execute the `grm renew` action. this will prepare a new package version of the project and upgrade the project files created from templates to its latest version.

2. code and check

   now use your favorite IDE/Editor to implement the new feature or code the bug fix. don't forget to amend the project with new unit and integrity tests, and ensure they pass, by executin from time to time the `grm check` action.

3. publish your changes

   before you initiate a push/merge request against the Gitlab server, execute the `grm prepare` action, which will create, with the help of the `git diff` command, a *.commit_msg.txt* file in the working tree root of your project, containing a short summary in the first line followed with a blank line and a list of the project files that got added, changed or deleted.

   Hint: the *.commit_msg.txt* file can be amended by any text editor before you run the `grm commit` action. for changes initiated by an issue please include the issue number (in the format `fixes #<issue-number>`) into this file. you may use `Markdown` syntax in this file for simple styling.

   ---

   to finally commit and upload your changes run the following three grm actions in the root folder of your project:

```
grm commit
grm push
grm request
```

the `grm commit` command is first executing a `grm check` action to do a finally check of the project resources and to run the unit and integrity tests. if all these checks pass then a new git commit will be created, including your changes to the project. `grm push``will then push the commit to your ``origin` remote repository (your fork) and `grm request` will finally create a bew merge/pull request against the `upstream` remote repository (the forked one).

Hint: to complete the workflow a maintainer of the project has to execute the `grm release` action. this will merge your changes into the main branch *develop* of the `upstream` repository and then release a new version of the project onto PyPI.

---

more detailed information of the features of the `grm` tool are available within the grm user manual.

using *git* and *Gitlab*

alternatively to the `grm` tool you could directly use the git command suite and the Gitlab website to achieve the same (with a lot more of typing and fiddling ;-):

1. fork the upstream repository into your user account.
2. clone your forked repo as `origin` remote to your computer, and add an `upstream` remote for the destination repo by running the following commands in the console of your local machine:

```
git clone https://gitlab.com/<YourGitLabUserName>/ae_ae.git
git remote add upstream https://gitlab.com/ae-group/ae_ae.git
```

3. checkout out a new local feature branch and update it to the latest version of the `develop` branch:

```
git checkout -b <new_feature_or_fix_branch_name> develop
git pull --rebase upstream develop
```

   please keep your code clean by staying current with the `develop` branch, where code will be merged. if you find another bug, please fix it in a separated branch instead.

4. push the branch to your fork. treat it as a backup:

```
git push origin <new_feature_or_fix_branch_name>
```

5. code

   implement the new feature or the bug fix; include tests, and ensure they pass.

6. check

   run the basic code style and typing checks locally (pylint, mypy and flake8) before you commit.

7. commit

   for every commit please write a short summary in the first line followed with a blank line and then more detailed descriptions of the change. for bug fixes please include any issue number (in the format #nnn) in your summary:

```
git commit -m "issue #123: put change summary here (can be a issue title)"
```

> **Note:** **never leave the commit message blank!** provide a detailed, clear, and complete description of your changes!

8. publish your changes (prepare a Merge Request)

   before submitting a merge request, update your branch to the latest code:

   ```
   git pull --rebase upstream develop
   ```

   if you have made many commits, we ask you to squash them into atomic units of work. most issues should have one commit only, especially bug fixes, which makes them easier to back port:

   ```
   git checkout develop
   git pull --rebase upstream develop
   git checkout <new_feature_or_fix_branch_name>
   git rebase -i develop
   ```

   push changes to your fork:

   ```
   git push -f
   ```

9. issue/make a GitLab Merge Request:
   - navigate to your fork where you just pushed to
   - click *Merge Request*
   - in the branch field write your feature branch name (this is filled with your default branch name)
   - click *Update Commit Range*
   - ensure the changes you implemented are included in the *Commits* tab
   - ensure that the *Files Changed* tab incorporate all of your changes
   - fill in some details about your potential patch including a meaningful title
   - click *New merge request*.

3.3.4 release to PyPI

the release of a new/changed project will automatically be initiated by the GitLab CI, using the two protected vars `PYPI_USERNAME` and `PYPI_PASSWORD` (marked as masked) from the users group of this namespace, in order to provide the user name and password of the maintainers PyPI account (on Gitlab.com at Settings/CI_CD/Variables).

3.3.5 useful links and resources

- General GitLab documentation
- GitLab workflow documentation
- grm (git repository manager) module `project repository` and user manual

3.4 create new namespace

a **PEP 420** namespace splits the codebase of a library or framework into multiple project repositories, called portions (of the namespace).

Hint: the *aedev* namespace is providing the grm tool to create and maintain any namespace and its portions.

the id of a new namespace consists of letters only and has to be available on PYPI. the group-name name gets by default generated from the namespace name plus the suffix `'-group'`, so best choose an id that results in a group name that is available on your repository server.

3.5 register a new namespace portion

follow the steps underneath to add and register a new module as portion onto the *ae* namespace:

1. open a console window and change the current directory to the parent directory of your projects root folders.
2. choose a not-existing/unique name for the new portion (referred as <*portion-name*> in the next steps).
3. run `grm --namespace=ae --project=<portion_name> new-module` to register the portion name within the namespace, to create a new project folder *ae_<portion-name>* (providing initial project files created from templates) and to get a pre-configured git repository (with the remote already set and the initial files unstaged, to be extended, staged and finally committed).
4. run `cd ae_<portion-name>` to change the current to the working tree root of the new portion project.
5. run pyenv local venv_name (or any other similar tool) to create/prepare a local virtual environment.
6. fans of TDD are then coding unit tests in the prepared test module *test_ae_<portion-name>.py*, situated within the *tests* sub-folder of your new code project folder.
7. extend the file <portion_name>.py situated in the *ae* sub-folder to implement the new portion.
8. run `grm check-integrity` to run the linting and unit tests (if they fail go one or two steps back).
9. run `grm prepare`, then amend the commit message within the file *.commit_msg.txt*, then run `grm commit` and `grm push` to commit and upload your new portion to your personal remote/server repository fork, and finally run `grm request` to request the merge/pull into the forked/upstream repository in the users group *ae-group* (at https://gitlab.com/ae-group).

the registration of a new portion to the ae namespace has to be done by a namespace maintainer.

registered portions will automatically be included into the *ae namespace documentation*, available at ReadTheDocs.

CHAPTER

FOUR

REGISTERED NAMESPACE PACKAGE PORTIONS

the following list contains all registered portions of the ae namespace, plus additional modules of each portion.

Hint: a not on the ordering: portions with no dependencies are at the begin of the following list. the portions that are depending on other portions of the ae namespace are listed more to the end.

ae.base	basic constants, helper functions and context manager
ae.deep	easy handling of deeply nested data structures
ae.django_utils	helpers for django projects
ae.droid	android constants and helper functions
ae.notify	send notifications via email, telegram or whatsapp
ae.valid	data validation helper functions
ae.files	generic file object helpers
ae.paths	generic file path helpers
ae.core	application core constants, helper functions and base classes
ae.lockname	named threading locks
ae.dynamicod	dynamic execution of code blocks and expressions
ae.i18n	internationalization / localization helpers
ae.parse_date	parse date strings more flexible and less strict
ae.literal	literal type detection and evaluation
ae.progress	display progress of long running processes
ae.updater	application environment updater
ae.console	console application environment
ae.sys_core	dynamic system configuration, initialization and connection
ae.sys_data	external system data structures
ae.sys_core_sh	SiHOT PMS system core xml interface
ae.sys_data_sh	Sihot system data interface
ae.db_core	database connection and data manipulation base classes
ae.db_ora	database system core layer to access Oracle databases
ae.db_pg	postgres database layer
ae.transfer_service	transfer client and server services
ae.sideloading_server	sideloading server
ae.gui_app	base class for python applications with a graphical user interface
ae.gui_help	main app base class with context help for flow and app state changes
ae.kivy_glsl	add glsl shaders to your kivy widget

continues on next page

Table 4.1 – continued from previous page

ae.kivy_dyn_chi	dynamic children mix-in for kivy container widgets
ae.kivy_relief_canvas	inner/outer elliptic/square reliefs for any kivy widget
ae.kivy	core application classes and widgets for GUIApp-conform Kivy apps
ae.kivy.i18n	ae.kivy.i18n module
ae.kivy.widgets	ae.kivy.widgets module
ae.kivy.apps	ae.kivy.apps module
ae.kivy.behaviors	ae.kivy.behaviors module
ae.kivy.tours	ae.kivy.tours module
ae.kivy_auto_width	automatic width mix-in classes for kivy widgets
ae.kivy_file_chooser	extended kivy file chooser widget
ae.kivy_iterable_displayer	iterable displayer widget
ae.kivy_qr_displayer	qr code displayer widget
ae.kivy_sideloading	kivy mixin and widgets to integrate a sideloading server in your app
ae.kivy_user_prefs	user preferences widgets for your kivy app
ae.lisz_app_data	lisz demo app data handling
ae.enaml_app	enaml application widgets, helper functions and classes
ae.enaml_app.functions	enaml application helper functions.

4.1 ae.base

4.1.1 basic constants, helper functions and context manager

this module is pure python, has no external dependencies, and is providing base constants, common helper functions and context managers.

base constants

ISO format strings for *date* and *datetime* values are provided by the constants *DATE_ISO* and *DATE_TIME_ISO*.

the *UNSET* constant is useful in cases where *None* is a valid data value and another special value is needed to specify that e.g. an argument or attribute has no (valid) value or did not get specified/passed.

default values to compile file and folder names for a package or an app project are provided by the constants: *DOCS_FOLDER*, *TESTS_FOLDER*, *TEMPLATES_FOLDER*, *BUILD_CONFIG_FILE*, *PACKAGE_INCLUDE_FILES_PREFIX*, *PY_EXT*, *PY_INIT*, *PY_MAIN*, *CFG_EXT* and *INI_EXT*.

the constants `PACKAGE_NAME`, `PACKAGE_DOMAIN` and `PERMISSIONS` are mainly used for apps running on mobile devices. to avoid redundancies, these values get loaded from the *build config file* - if it exists in the current working directory.

base helper functions

to write more compact and readable code for the most common file I/O operations, the helper functions *read_file()* and *write_file()* are wrapping Python's built-in open() function and its context manager.

the function *duplicates()* returns the duplicates of an iterable type.

norm_line_sep() is converting any combination of line separators of a string to a single new-line character.

norm_name() converts any string into a name that can be used e.g. as file name or as method/attribute name.

to normalize a file path, in order to remove ., .. placeholders, to resolve symbolic links or to make it relative or absolute, call the function *norm_path()*.

camel_to_snake() and *snake_to_camel()* providing name conversions of class and method names.

to encode unicode strings to other codecs the functions *force_encoding()* and *to_ascii()* can be used.

the *round_traditional()* function get provided by this module for traditional rounding of float values. the function signature is fully compatible to Python's round() function.

the function *instantiate_config_parser()* ensures that the ConfigParser instance is correctly configured, e.g. to support case-sensitive config variable names and to use ExtendedInterpolation for the interpolation argument.

app_name_guess() guesses the name of o running Python application from the application environment, with the help of *build_config_variable_values()*, which determines config-variable-values from the build spec file of an app project.

operating system constants and helpers

the string *os_platform* provides the OS where your app is running, extending Python's sys.platform() for mobile platforms like Android and iOS.

os_host_name(), *os_local_ip()* and *os_user_name()* are determining machine and user information from the OS.

use *env_str()* to determine the value of an OS environment variable with automatic variable name conversion. other helper functions provided by this namespace portion to determine the values of the most important system environment variables for your application are *sys_env_dict()* and *sys_env_text()*.

android-specific constants and helper functions

some helper functions of this module are provided to be used for the Android OS.

the helper function *start_app_service()* is starting a service in its own, separate thread. with the function *request_app_permissions()* you can ensure that all your Android permissions will be requested. the module *ae.kivy.apps* does this automatically on app startup. on other platforms than Android it will have no effect to call these functions.

Note: importing this module on an Android OS, it is monkey patching the shutil module to prevent crashes.

links to other android code and service examples and documentation:

- **`https://python-for-android.readthedocs.io/en/latest/`__**
- **`https://github.com/kivy/python-for-android/tree/develop/pythonforandroid/recipes/android/src/android`__**
- **`https://github.com/tshirtman/kivy_service_osc/blob/master/src/main.py`__**

- `https://blog.kivy.org/2014/01/building-a-background-application-on-android-with-kivy/`__
- `https://github.com/Android-for-Python/Android-for-Python-Users`__
- `https://github.com/Android-for-Python/INDEX-of-Examples`__

big thanks to Robert Flatt for his investigations, findings and documentations to code and build Kivy apps for the Android OS, and to Gabriel Pettier for his service osc example.

generic context manager

the context manager *in_wd()* allows to switch the current working directory temporarily. the following example demonstrates a typical usage, together with a temporary path, created with the help of Pythons `TemporaryDirectory` class:

```
with tempfile.TemporaryDirectory() as tmp_dir, in_wd(tmp_dir):
    # within the context the tmp_dir is set as the current working directory
    assert os.getcwd() == tmp_dir
# current working directory set back to the original path and the temporary directory␣
↪got removed
```

call stack inspection

module_attr() dynamically determines a reference to an attribute (variable, function, class, ...) in a module.

module_name(), *stack_frames()*, *stack_var()* and *stack_vars()* are inspecting the call stack frames to determine e.g. variable values of the callers of a function/method.

Hint: the `AppBase` class uses these helper functions to determine the `version` and `title` of an application, if these values are not specified in the instance initializer.

---

another useful helper function provided by this portion to inspect and debug your code is *full_stack_trace()*.

Module Attributes

DOCS_FOLDER	project documentation root folder name
TESTS_FOLDER	name of project folder to store unit/integration tests
TEMPLATES_FOLDER	template folder name, used in template and namespace root projects to maintain and provide common file templates
BUILD_CONFIG_FILE	gui app build config file
PACKAGE_INCLUDE_FILES_PREFIX	file/folder names prefix included into setup package_data/ae_updater
PY_CACHE_FOLDER	python cache folder name
PY_EXT	file extension for modules and hooks
PY_INIT	init-module file name of a python package
PY_MAIN	main-module file name of a python executable
CFG_EXT	CFG config file extension
INI_EXT	INI config file extension
DATE_ISO	ISO string format for date values (e.g.
DATE_TIME_ISO	ISO string format for datetime values
DEF_ENCODE_ERRORS	default encode error handling for UnicodeEncodeErrors
DEF_ENCODING	encoding for *force_encoding()* that will always work independent from destination (console, file sys, ...).
NAME_PARTS_SEP	name parts separator character, e.g.
SKIPPED_MODULES	skipped modules used as default by *module_name()*, *stack_var()* and *stack_vars()*
UNSET	pseudo value used for attributes/arguments if *None* is needed as a valid value
os_platform	operating system / platform string (see *_os_platform()*).

Functions

app_name_guess()	guess/try to determine the name of the currently running app (w/o assessing not yet initialized app instance).
build_config_variable_values(*names_defaults)	determine build config variable values from the `buildozer.spec` file in the current directory.
camel_to_snake(name)	convert name from CamelCase to snake_case.
deep_dict_update(data, update)	update the optionally nested data dict in-place with the items and sub-items from the update dict.
dummy_function(*_args, **_kwargs)	null function accepting any arguments and returning None.
duplicates(values)	determine all duplicates in the iterable specified in the `values` argument.
env_str(name[, convert_name])	determine the string value of an OS environment variable, optionally preventing invalid variable name.
force_encoding(text[, encoding, errors])	force/ensure the encoding of text (str or bytes) without any UnicodeDecodeError/UnicodeEncodeError.
full_stack_trace(ex)	get full stack trace from an exception.
import_module(import_name[, path])	search, import and execute a Python module dynamically without adding it to sys.modules.

continues on next page

Table 4.2 – continued from previous page

in_wd(new_cwd)	context manager to temporary switch the current working directory / cwd.
instantiate_config_parser()	instantiate and prepare config file parser.
main_file_paths_parts(portion_name)	determine tuple of supported main/version file name path part tuples.
module_attr(import_name[, attr_name])	determine dynamically a reference to a module or to any attribute (variable/func/class) declared in the module.
module_file_path([local_object])	determine the absolute path of the module from which this function got called.
module_name(*skip_modules[, depth])	find the first module in the call stack that is *not* in *skip_modules*.
norm_line_sep(text)	convert any combination of line separators in the *text* arg to new-line characters.
norm_name(name[, allow_num_prefix])	normalize name to start with a letter/alphabetic/underscore and to contain only alphanumeric/underscore chars.
norm_path(path[, make_absolute, ...])	normalize path, replacing ../. parts or the tilde character (for home folder) and transform to relative/abs.
now_str([sep])	return the current timestamp as string (to use as suffix for file and variable/attribute names).
os_host_name()	determine the operating system host/machine name.
os_local_ip()	determine ip address of this system/machine in the local network (LAN or WLAN).
os_user_name()	determine the operating system username.
project_main_file(import_name[, project_path])	determine the main module file path of a project package, containing the project __version__ module variable.
read_file(file_path[, extra_mode, encoding, ...])	returning content of the text/binary file specified by file_path argument as string.
request_app_permissions(*_args, **_kwargs)	null function accepting any arguments and returning None.
round_traditional(num_value[, num_digits])	round numeric value traditional.
snake_to_camel(name[, back_convertible])	convert name from snake_case to CamelCase.
stack_frames([depth])	generator returning the call stack frame from the level given in *depth*.
stack_var(name, *skip_modules[, scope, depth])	determine variable value in calling stack/frames.
stack_vars(*skip_modules[, find_name, ...])	determine all global and local variables in a calling stack/frames.
start_app_service(*_args, **_kwargs)	null function accepting any arguments and returning None.
sys_env_dict()	returns dict with python system run-time environment values.
sys_env_text([ind_ch, ind_len, key_ch, ...])	compile formatted text block with system environment info.
to_ascii(unicode_str)	converts unicode string into ascii representation.
write_file(file_path, content[, extra_mode, ...])	(over)write the file specified by *file_path* with text or binary/bytes content.

Classes

UnsetType()	(singleton) UNSET (type) object class.

DOCS_FOLDER = 'docs'

project documentation root folder name

TESTS_FOLDER = 'tests'

name of project folder to store unit/integration tests

TEMPLATES_FOLDER = 'templates'

template folder name, used in template and namespace root projects to maintain and provide common file templates

BUILD_CONFIG_FILE = 'buildozer.spec'

gui app build config file

PACKAGE_INCLUDE_FILES_PREFIX = 'ae_'

file/folder names prefix included into setup package_data/ae_updater

PY_CACHE_FOLDER = '__pycache__'

python cache folder name

PY_EXT = '.py'

file extension for modules and hooks

PY_INIT = '__init__.py'

init-module file name of a python package

PY_MAIN = '__main__.py'

main-module file name of a python executable

CFG_EXT = '.cfg'

CFG config file extension

INI_EXT = '.ini'

INI config file extension

DATE_ISO = '%Y-%m-%d'

ISO string format for date values (e.g. in config files/variables)

DATE_TIME_ISO = '%Y-%m-%d %H:%M:%S.%f'

ISO string format for datetime values

DEF_ENCODE_ERRORS = 'backslashreplace'

default encode error handling for UnicodeEncodeErrors

DEF_ENCODING = 'ascii'

encoding for *force_encoding()* that will always work independent from destination (console, file sys, ...).

NAME_PARTS_SEP = '_'

name parts separator character, e.g. for *norm_name()*

SKIPPED_MODULES = ('ae.base', 'ae.paths', 'ae.dynamicod', 'ae.core', 'ae.console', 'ae.gui_app', 'ae.gui_help', 'ae.kivy', 'ae.kivy.apps', 'ae.kivy.behaviors', 'ae.kivy.i18n', 'ae.kivy.tours', 'ae.kivy.widgets', 'ae.enaml_app', 'ae.beeware_app', 'ae.pyglet_app', 'ae.pygobject_app', 'ae.dabo_app', 'ae.qpython_app', 'ae.appjar_app', 'importlib._bootstrap', 'importlib._bootstrap_external')

skipped modules used as default by *module_name()*, *stack_var()* and *stack_vars()*

class UnsetType

Bases: object

(singleton) UNSET (type) object class.

__bool__()

ensure to be evaluated as False, like None.

__len__()

ensure to be evaluated as empty.

UNSET = <ae.base.UnsetType object>

pseudo value used for attributes/arguments if *None* is needed as a valid value

app_name_guess()

guess/try to determine the name of the currently running app (w/o assessing not yet initialized app instance).

Return type
str

Returns
application name/id or "unguessable" if not guessable.

build_config_variable_values(**names_defaults*, *section='app'*)

determine build config variable values from the `buildozer.spec` file in the current directory.

Parameters

- **names_defaults** (Tuple[str, Any]) – tuple of tuples of build config variable names and default values.
- **section** (str) – name of the spec file section, using 'app' as default.

Return type
Tuple[Any, ...]

Returns
tuple of build config variable values (using the passed default value if not specified in the *BUILD_CONFIG_FILE* spec file or if the spec file does not exist in cwd).

camel_to_snake(*name*)

convert name from CamelCase to snake_case.

Parameters
name (str) – name string in CamelCaseFormat.

Return type
str

Returns
name in snake_case_format.

deep_dict_update(*data*, *update*)

update the optionally nested data dict in-place with the items and sub-items from the update dict.

Parameters

- **data** (dict) – dict to be updated/extended. non-existing keys of dict-sub-items will be added.
- **update** (dict) – dict with the [sub-]items to update in the data dict.

Hint: the module/portion *ae.deep* is providing more deep update helper functions.

dummy_function(**_args*, ***_kwargs*)

null function accepting any arguments and returning None.

Parameters

- **_args** – ignored positional arguments.
- **_kwargs** – ignored keyword arguments.

Returns
always None.

duplicates(*values*)

determine all duplicates in the iterable specified in the values argument.

inspired by Ritesh Kumars answer to https://stackoverflow.com/questions/9835762.

Parameters
values (Iterable) – iterable (list, tuple, str, ...) to search for duplicate items.

Return type
list

Returns
list of the duplicate items found (can contain the same duplicate multiple times).

env_str(*name*, *convert_name=False*)

determine the string value of an OS environment variable, optionally preventing invalid variable name.

Parameters

- **name** (str) – name of an OS environment variable.
- **convert_name** (bool) – pass True to prevent invalid variable names by converting CamelCase names into SNAKE_CASE, lower-case into upper-case and all non-alpha-numeric characters into underscore characters.

Return type
Optional[str]

Returns
string value of OS environment variable if found, else None.

force_encoding(*text*, *encoding='ascii'*, *errors='backslashreplace'*)

force/ensure the encoding of text (str or bytes) without any UnicodeDecodeError/UnicodeEncodeError.

Parameters

- **text** (Union[str, bytes]) – text as str/bytes.
- **encoding** (str) – encoding (def= *DEF_ENCODING*).

- **errors**¶ (str) – encode error handling (def= *DEF_ENCODE_ERRORS*).

Return type
str

Returns
text as str (with all characters checked/converted/replaced to be encode-able).

full_stack_trace(*ex*)

get full stack trace from an exception.

Parameters
ex¶ (Exception) – exception instance.

Return type
str

Returns
str with stack trace info.

import_module(*import_name*, *path=<ae.base.UnsetType object>*)

search, import and execute a Python module dynamically without adding it to sys.modules.

Parameters

- **import_name**¶ (str) – dot-name of the module to import.
- **path**¶ (Union[str, *UnsetType*, None]) – optional file path of the module to import. if this arg is not specified or has the default value (*UNSET*), then the path will be determined from the import name. specify None to prevent the module search.

Return type
Optional[ModuleType]

Returns
a reference to the loaded module or None if module could not be imported.

instantiate_config_parser()

instantiate and prepare config file parser.

Return type
ConfigParser

in_wd(*new_cwd*)

context manager to temporary switch the current working directory / cwd.

Parameters
new_cwd¶ (str) – path to the directory to switch to (within the context/with block). an empty string gets interpreted as the current working directory.

Return type
Generator[None, None, None]

main_file_paths_parts(*portion_name*)

determine tuple of supported main/version file name path part tuples.

Parameters
portion_name¶ (str) – portion or package name.

Return type
Tuple[Tuple[str, ...], ...]

Returns
tuple of tuples of main/version file name path parts.

module_attr(*import_name*, *attr_name=''*)

determine dynamically a reference to a module or to any attribute (variable/func/class) declared in the module.

Parameters

- **import_name** (str) – import-/dot-name of the distribution/module/package to load/import.
- **attr_name** (str) – name of the attribute declared within the module. do not specify or pass an empty string to get/return a reference to the imported module instance.

Return type
Optional[Any]

Returns
module instance or module attribute value or None if module not found or UNSET if module attribute doesn't exist.

Note: a previously not imported module will *not* be added to *sys.modules* by this function.

---

module_file_path(*local_object=None*)

determine the absolute path of the module from which this function got called.

Parameters
local_object (Optional[Callable]) – optional local module, class, method, function, traceback, frame, or code object of the calling module (passing *lambda: 0* also works). omit to use instead the *__file__* module variable (which will not work if the module is frozen by `py2exe` or `PyInstaller`).

Return type
str

Returns
module path (inclusive module file name) or empty string if path not found/determinable.

module_name(**skip_modules*, *depth=0*)

find the first module in the call stack that is *not* in *skip_modules*.

Parameters

- **skip_modules** (str) – module names to skip (def=this ae.core module).
- **depth** (int) – the calling level from which on to search. the default value 0 refers the frame and the module of the caller of this function. pass 1 or an even higher value if you want to get the module name of a function/method in a deeper level in the call stack.

Return type
Optional[str]

Returns
the module name of the call stack level specified by *depth*.

norm_line_sep(*text*)

convert any combination of line separators in the *text* arg to new-line characters.

Parameters
text (str) – string containing any combination of line separators ('\r\n' or '\r').

Return type
str

Returns
normalized/converted string with only new-line ('\n') line separator characters.

norm_name(*name*, *allow_num_prefix=False*)

normalize name to start with a letter/alphabetic/underscore and to contain only alphanumeric/underscore chars.

Parameters

- **name**¶ (str) – any string to be converted into a valid variable/method/file/... name.
- **allow_num_prefix**¶ (bool) – pass True to allow leading digits in the returned normalized name.

Return type
str

Returns
cleaned/normalized/converted name string (e.g. for a variable-/method-/file-name).

norm_path(*path*, *make_absolute=True*, *remove_base_path=''*, *remove_dots=True*, *resolve_sym_links=True*)

normalize path, replacing ../. parts or the tilde character (for home folder) and transform to relative/abs.

Parameters

- **path**¶ (str) – path string to normalize/transform.
- **make_absolute**¶ (bool) – pass False to not convert path to an absolute path.
- **remove_base_path**¶ (str) – pass a valid base path to return a relative path, even if the argument values of *make_absolute* or *resolve_sym_links* are *True*.
- **remove_dots**¶ (bool) – pass False to not replace/remove the . and .. placeholders.
- **resolve_sym_links**¶ (bool) – pass False to not resolve symbolic links, passing True implies a *True* value also for the *make_absolute* argument.

Return type
str

Returns
normalized path string: absolute if *remove_base_path* is empty and either *make_absolute* or *resolve_sym_links* is *True*; relative if *remove_base_path* is a base path of *path* or if *path* got specified as relative path and neither *make_absolute* nor *resolve_sym_links* is *True*.

Hint: the *normalize()* function additionally replaces *PATH_PLACEHOLDERS*.

---

now_str(*sep=''*)

return the current timestamp as string (to use as suffix for file and variable/attribute names).

Parameters
sep¶ (str) – optional prefix and separator character (separating date from time and in time part the seconds from the microseconds).

Return type
str

Returns
timestamp as string (length=20 + 3 * len(sep)).

os_host_name()

determine the operating system host/machine name.

Return type
str

Returns
machine name string.

os_local_ip()

determine ip address of this system/machine in the local network (LAN or WLAN).

inspired by answers of SO users @dml and @fatal_error to the question: https://stackoverflow.com/questions/166506.

Return type
str

Returns
ip address of this machine in the local network (WLAN or LAN/ethernet) or empty string if this machine is not connected to any network.

_os_platform()

determine the operating system where this code is running (used to initialize the *os_platform* variable).

Return type
str

Returns

operating system (extension) as string:

- *'android'* for all Android systems.
- *'cygwin'* for MS Windows with an installed Cygwin extension.
- *'darwin'* for all Apple Mac OS X systems.
- *'freebsd'* for all other BSD-based unix systems.
- *'ios'* for all Apple iOS systems.
- *'linux'* for all other unix systems (like Arch, Debian/Ubuntu, Suse, ...).
- *'win32'* for MS Windows systems (w/o the Cygwin extension).

os_platform = 'linux'

operating system / platform string (see *_os_platform()*).

this string value gets determined for most of the operating systems with the help of Python's sys.platform() function and additionally detects the operating systems iOS and Android (not supported by Python).

os_user_name()

determine the operating system username.

Return type
str

Returns
username string.

project_main_file(*import_name*, *project_path=''*)

determine the main module file path of a project package, containing the project __version__ module variable.

Parameters

- **import_name**¶ (str) – import name of the module/package (including namespace prefixes for namespace packages).
- **project_path**¶ (str) – optional path where the project of the package/module is situated. not needed if the current working directory is the root folder of either the import_name project or of a sister project (under the same project parent folder).

Return type
str

Returns
absolute file path/name of main module or empty string if no main/version file found.

read_file(*file_path*, *extra_mode=''*, *encoding=None*, *error_handling='ignore'*)

returning content of the text/binary file specified by file_path argument as string.

Parameters

- **file_path**¶ (str) – file path/name to load into a string or a bytes array.
- **extra_mode**¶ (str) – extra open mode flag characters appended to "r" onto open() mode argument. pass "b" to read the content of a binary file returned as bytes array. in binary mode the argument passed in *error_handling* will be ignored.
- **encoding**¶ (Optional[str]) – encoding used to load and convert/interpret the file content.
- **error_handling**¶ (str) – for files opened in text mode pass *'strict'* or *None* to return *None* (instead of an empty string) for the cases where either a decoding *ValueError* exception or any *OSError*, *FileNotFoundError* or *PermissionError* exception got raised. the default value *'ignore'* will ignore any decoding errors (missing some characters) and will return an empty string on any file/os exception. this parameter will be ignored if the *extra_mode* argument contains the 'b' character (to read the file content as binary/bytes-array).

Return type
Union[str, bytes]

Returns
file content string or bytes array.

Raises

- **FileNotFoundError** – if file does not exist.
- **OSError** – if *file_path* is misspelled or contains invalid characters.
- **PermissionError** – if current OS user account lacks permissions to read the file content.
- **ValueError** – on decoding errors.

round_traditional(*num_value*, *num_digits=0*)

round numeric value traditional.

needed because python round() is working differently, e.g. round(0.075, 2) == 0.07 instead of 0.08 inspired by https://stackoverflow.com/questions/31818050/python-2-7-round-number-to-nearest-integer.

Parameters

- **num_value**¶ (float) – float value to be round.
- **num_digits**¶ (int) – number of digits to be round (def=0 - rounds to an integer value).

Return type
float

Returns
rounded value.

snake_to_camel(*name*, *back_convertible=False*)

convert name from snake_case to CamelCase.

Parameters

- **name**¶ (str) – name string composed of parts separated by an underscore character (*NAME_PARTS_SEP*).
- **back_convertible**¶ (bool) – pass *True* to get the first character of the returned name in lower-case if the snake name has no leading underscore character (and to allow the conversion between snake and camel case without information loss).

Return type
str

Returns
name in camel case.

stack_frames(*depth=1*)

generator returning the call stack frame from the level given in *depth*.

Parameters
depth¶ (int) – the stack level to start; the first returned frame by this generator. the default value (1) refers the next deeper stack frame, respectively the one of the caller of this function. pass 2 or a higher value if you want to start with an even deeper frame/level.

Return type
Generator

Returns
generated frames of the call stack.

stack_var(*name*, **skip_modules*, *scope=''*, *depth=1*)

determine variable value in calling stack/frames.

Parameters

- **name**¶ (str) – variable name to search in the calling stack frames.
- **skip_modules**¶ (str) – module names to skip (def=see *SKIPPED_MODULES* module constant).
- **scope**¶ (str) – pass 'locals' to only check for local variables (ignoring globals) or 'globals' to only check for global variables (ignoring locals). the default value (an empty string) will not restrict the scope, returning either a local or global value.
- **depth**¶ (int) – the calling level from which on to search. the default value (1) refers the next deeper stack frame, which is the caller of the function. pass 2 or an even higher value if you want to start the variable search from a deeper level in the call stack.

Return type
Optional[Any]

Returns
the variable value of a deeper level within the call stack or UNSET if the variable was not found.

stack_vars(**skip_modules*, *find_name=''*, *min_depth=1*, *max_depth=0*, *scope=''*)

determine all global and local variables in a calling stack/frames.

Parameters

- **skip_modules** (str) – module names to skip (def=see *SKIPPED_MODULES* module constant).
- **find_name** (str) – if passed then the returned stack frame must contain a variable with the passed name.
- **scope** (str) – scope to search the variable name passed via *find_name*. pass 'locals' to only search for local variables (ignoring globals) or 'globals' to only check for global variables (ignoring locals). passing an empty string will find the variable within either locals and globals.
- **min_depth** (int) – the call stack level from which on to search. the default value (1) refers the next deeper stack frame, respectively to the caller of this function. pass 2 or a higher value if you want to get the variables from a deeper level in the call stack.
- **max_depth** (int) – the maximum depth in the call stack from which to return the variables. if the specified argument is not zero and no *skip_modules* are specified then the first deeper stack frame that is not within the default *SKIPPED_MODULES* will be returned. if this argument and *find_name* get not passed then the variables of the top stack frame will be returned.

Return type
Tuple[Dict[str, Any], Dict[str, Any], int]

Returns
tuple of the global and local variable dicts and the depth in the call stack.

sys_env_dict()

returns dict with python system run-time environment values.

Return type
Dict[str, Any]

Returns
python system run-time environment values like python_ver, argv, cwd, executable, frozen and bundle_dir (if bundled with pyinstaller).

Hint: see also https://pyinstaller.readthedocs.io/en/stable/runtime-information.html

---

sys_env_text(*ind_ch=' '*, *ind_len=12*, *key_ch='='*, *key_len=15*, *extra_sys_env_dict=None*)

compile formatted text block with system environment info.

Parameters

- **ind_ch** (str) – indent character (default=" ").
- **ind_len** (int) – indent depths (default=12 characters).
- **key_ch** (str) – key-value separator character (default="=").
- **key_len** (int) – key-name minimum length (default=15 characters).
- **extra_sys_env_dict** (Optional[Dict[str, str]]) – dict with additional system info items.

Return type
str

Returns
text block with system environment info.

to_ascii(*unicode_str*)

converts unicode string into ascii representation.

useful for fuzzy string compare; inspired by MiniQuark's answer in: https://stackoverflow.com/questions/517923/what-is-the-best-way-to-remove-accents-in-a-python-unicode-string

Parameters
unicode_str¶ (str) – string to convert.

Return type
str

Returns
converted string (replaced accents, diacritics, ... into normal ascii characters).

write_file(*file_path*, *content*, *extra_mode=''*, *encoding=None*)

(over)write the file specified by *file_path* with text or binary/bytes content.

Parameters

- **file_path**¶ (str) – file path/name to write the passed content into (overwriting any previous content!).
- **content**¶ (Union[str, bytes]) – new file content either passed as string or list of line strings (will be concatenated with the line separator of the current OS: os.linesep).
- **extra_mode**¶ (str) – open mode flag characters. passed unchanged to the *mode* argument of open() if this argument starts with 'a', else this argument value will be appended to 'w'.
- **encoding**¶ (Optional[str]) – encoding used to write/convert/interpret the file content to write.

Raises

- **FileExistsError** – if file exists already and is write-protected.
- **FileNotFoundError** – if parts of the file path do not exist.
- **OSError** – if *file_path* is misspelled or contains invalid characters.
- **PermissionError** – if current OS user account lacks permissions to read the file content.
- **ValueError** – on decoding errors.

request_app_permissions(**_args*, ***_kwargs*)

null function accepting any arguments and returning None.

Parameters

- **_args**¶ – ignored positional arguments.
- **_kwargs**¶ – ignored keyword arguments.

Returns
always None.

start_app_service(**_args*, ***_kwargs*)

null function accepting any arguments and returning None.

Parameters

- **_args**¶ – ignored positional arguments.
- **_kwargs**¶ – ignored keyword arguments.

Returns
always None.

4.2 ae.deep

4.2.1 easy handling of deeply nested data structures

this ae namespace portion is pure python, depends only on the Python runtime and the *ae.base* portion, and provides functions for to read, update and delete values of deep data structures. more helper function to prepare and convert data structures between different systems are available in the *ae.sys_data* module.

the root and node objects of deep data structures consisting of sequences (like list, tuple, ...), mappings (dict, ...) and data (class) objects. the leaf data objects are mostly simple types like int, float or string.

deep data structure example

the following deep data structure is composed of the data class `Person`, a member list and two dictionaries:

```
>>> from dataclasses import dataclass
>>> @dataclass
... class Person:
...     first_name: str
...     hobbies: List[str]
```

```
>>> member_hobbies = [
...         "dancing",
...         "music",            # ...
...         ]
```

```
>>> member_peter = Person(
...     first_name="Peter",
...     hobbies=member_hobbies,
...     # ...
...     )
```

```
>>> member_list = [
...     member_peter,           # ...
...     ]
```

```
>>> club_data = {
...     'city': "Madrid",
...     'members': member_list, # ...
...     }
```

```
>>> clubs_mapping = {
...     'fun-club': club_data,  # ...
...     }
```

putting the above data structures together, results in a deep data structure, in where `clubs_mapping` represents the root object.

the nodes of this deep data structure get referenced by `club_data`, `member_list`, `member_peter` and `member_hobbies`.

the fields `city`, `first_name` and the items `0` and `1` of `member_hobbies` (referencing the values `"dancing"` and `"music"`) are finally representing the leafs of this data structure:

```
>>> clubs_mapping == {
...     'fun-club': {
...         'city': "Madrid",
...         'members': [
...             Person(
...                 first_name="Peter",
...                 hobbies=[
...                     "dancing",
...                     "music",
...                 ]
...             ),
...         ]
...     }
... }
True
```

referencing a deep data object

there are two types of paths to reference the data items within a deep data structure: `object key lists` and `key path strings`.

to get any node object or leaf value within a deep data structure, referenced by a key path string, call the functions *key_path_object()*, which expects a data structure in its first argument *obj* and a key path string in its *key_path* second argument.

in the following example, the function *key_path_object()* determines the first name object from the `member_list` data node:

```
>>> key_path_object(member_list, '0.first_name')
'Peter'
```

to determine the same object via an object key list, use the function *key_list_object()*:

```
>>> key_list_object([(member_list, 0),
...                  (member_peter, 'first_name')])
'Peter'
```

use the function *key_path_string()* to convert an object key list into a key path string. the following example determines the same `first_name` data leaf object with an object key list:

```
>>> key_path = key_path_string([(member_list, 0),
...                             (member_peter, 'first_name')])
>>> print(repr(key_path))
'0.first_name'
>>> key_path_object(member_list, key_path)
'Peter'
```

e.g. the more deep/complex key path string `'fun-club.members.0.first_name.4'`, references the 5th character of the leaf object "Peter", this time from the root node of the example data structure (`clubs_mapping`):

```
>>> key_path_object(clubs_mapping, 'fun-club.members.0.first_name.4')
'r'
```

the same char object, referenced above with a key path string, can also be referenced with an object key list, with the help of the function *key_path_string()*:

```
>>> key_path_string([
...     (clubs_mapping, 'fun-club'),    # clubs_mapping['fun-club'] == club_data
...     (club_data, 'members'),         # club_data['members'] == member_list
...     (member_list, 0),               # member_list[0] == member_peter
...     (member_peter, 'first_name'),   # member_peter.first_name == "Peter"
...     ("Peter", 4),                   # "Peter"[4] == "r"
...     ])
'fun-club.members.0.first_name.4'
```

helpers to examine deep data structures

the *deep_search()* function allows to scan and inspect all the elements of any deep data structure. *deep_search()* can also be very useful for discovering internals of the Python language/libraries or to debug and test deep and complex data structures.

object_items() is another useful helper function which is returning a list of key-value pairs of any type of data node object.

helpers to change data in deep data structures

use the function *replace_object()* to change/replace a single node or leaf object within a deep data structure. alternatively you could use *key_path_object()*, by passing the new value as additional argument to it.

for multiple/bulk changes use the function *deep_replace()*, which is traversing/scanning the entire data structure.

the function *deep_update()* merges two deep data structures.

to wipe any node/leaf from a deep data structure use the function *pop_object()*, which returns the old/removed node, item or attribute value. another option to remove objects from a data structure is to use *deep_update()* with data:*~ae.base.UNSET* values in its *updating_obj* argument.

more details you find in the respective docstring of these functions.

Module Attributes

DataLeafTypesType	list/tuple of types of deep data leaves
KeyType	index/attribute types of deep data structures
KeyFilterCallable	callable to filter item key/index and attribute names
ObjKeysType	object key list of tuples of: object, key/index/attribute
ObjCheckCallable	*deep_replace()*/*deep_search()* parameter type
MutableDataTypes	deep data structure node types
DEFAULT_LEAF_TYPES	default leaf types of deep data structures
KEY_PATH_SEPARATORS	separator characters in key path string

Functions

deep_replace(obj, replace_with[, ...])	replace values (bottom up) within the passed (deeply nested) data structure.
deep_search(obj, found[, leaf_types, ...])	search key and/or value within the passed (deeply nested) data object structure.
deep_update(obj, updating_obj[, leaf_types, ...])	merge the `updating_obj` data structure into the similar structured node object `obj`.
key_filter_default(index_or_attr)	default key filter callable, returning True to filter out key/index/attribute id/name.
key_list_object(obj_keys)	determine object in a deep nested data structure via an object key list.
key_path_object(obj, key_path[, new_value])	determine object in a deep nested data structure via a key path string, and optionally assign a new value to it.
key_path_string(obj_keys)	convert obj keys path into deep object key path string.
key_value(key_str)	convert key string (mapping key, sequence index, obj attribute) into its value (str, int, float, tuple, ..).
next_key_of_path(key_path)	parse key_path to determine the next item key/index, the path separator and the rest of the key path.
object_item_value(obj, key[, default_value])	determine value of a data object item/attribute.
object_items(obj[, leaf_types, key_filter])	determine the items of a data object, with mapping keys, sequence indexes or attribute names and its values.
pop_object(obj_keys)	delete sub-attribute/item of a data object.
replace_object(obj_keys, new_value)	set sub-attribute/item with a mutable parent object to a new value within a deeply nested object/data structure.
value_filter_default(value, obj_keys)	default item value filter callable, returning True to filter out values to be scanned deeper.

DataLeafTypesType

list/tuple of types of deep data leaves

alias of `Tuple`[`Type`, ...]

KeyType

index/attribute types of deep data structures

alias of `Union`[`str`, `int`, `tuple`]

KeyFilterCallable

callable to filter item key/index and attribute names

alias of `Callable`[[`Union`[`str`, `int`, `tuple`]], `bool`]

ObjKeysType

object key list of tuples of: object, key/index/attribute

alias of `List`[`Tuple`[`Any`, `Any`]]

ObjCheckCallable

deep_replace()/*deep_search()* parameter type

alias of `Callable`[[`List`[`Tuple`[`Any`, `Any`]], `Any`, `Any`], `Any`]

MutableDataTypes = (<class 'collections.abc.MutableMapping'>, <class 'collections.abc.MutableSequence'>)

deep data structure node types

DEFAULT_LEAF_TYPES = (<class 'bytes'>, <class 'int'>, <class 'float'>, <class 'set'>, <class 'str'>, <class 'type'>)

default leaf types of deep data structures

KEY_PATH_SEPARATORS = ('.', '[', ']')

separator characters in key path string

key_filter_default(*index_or_attr*)

default key filter callable, returning True to filter out key/index/attribute id/name.

this function is the default value for the *key_filter* parameter of the deep scanning and traversing functions *deep_replace()*, *deep_search()*, *deep_update()* and *object_items()*.

if you are using your own filter callable you could call this function from it to prevent endless recursion, especially if your deep data structures contains circular- or self-referencing objects, like e.g.: :rtype: bool

- self- or doubly-linked data structures (e.g. `kivy.app.App.proxy_ref` (self-linked) or `ae.kivy.apps.KivyMainApp.framework_app` and *ae.kivy.apps.FrameworkApp.main_app*).
- back-linked data structures, like e.g. the `parent` property in Kivy widget trees.
- magic (two leading underscore characters) or internal (one leading underscore) attributes (e.g. via WindowSDL._WindowBase__instance).

value_filter_default(*value*, *obj_keys*)

default item value filter callable, returning True to filter out values to be scanned deeper.

this function is the default value for the *value_filter* parameter of the deep scanning and traversing functions *deep_replace()*, *deep_search()* and *deep_update()*.

by using your own filter callable make sure to prevent endless processing or recursion, especially if your deep data structures contains circular- or self-referencing objects. e.g. some data value types have to be excluded from to be deeper processed to prevent RecursionError (endless recursion, e.g. on str values because str[i] is str, on int because int.denominator is int).

Return type
bool

deep_replace(*obj*, *replace_with*, *leaf_types=(<class 'bytes'>, <class 'int'>, <class 'float'>, <class 'set'>, <class 'str'>, <class 'type'>)*, *key_filter=<function key_filter_default>*, *value_filter=<function value_filter_default>*, *obj_keys=None*)

replace values (bottom up) within the passed (deeply nested) data structure.

Parameters

- **obj** (Any) – mutable sequence or mapping data structure to be deep searched and replaced. can contain any combination of deep nested data objects. mutable node objects (e.g. dict/list) as well as the immutable types not included in *leaf_types* will be recursively deep searched (top down) by passing their items one by one to the callback function specified by *replace_with*.
- **replace_with** (Callable[[List[Tuple[Any, Any]], Any, Any], Any]) – called for each item with the 3 arguments object key list, key in parent data-structure, and the object/value. any return value other than *UNSET* will be used to overwrite the node/leaf object in the data-structure.
- **leaf_types** (Tuple[Type, ...]) – tuple of leaf types to skip from to be searched deeper. the default value of this parameter is specified in the modul constant *DEFAULT_LEAF_TYPES*.

- **key_filter** (Callable[[Union[str, int, tuple]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for item-key/item-index/attribute-name to be filtered out. by default all attribute/key names starting with an underscore character will be filtered out (see default callable *key_filter_default()*).
- **value_filter** (Callable[[Any, List[Tuple[Any, Any]]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for items/attributes values to be filtered out. by default empty values, excluded values (see EXCLUDED_VALUE_TYPES) and already scanned objects will be filtered out (see default callable *value_filter_default()*).
- **obj_keys** (Optional[List[Tuple[Any, Any]]]) – used (internally only) to pass the parent data-struct path in recursive calls.

Return type
int

Returns
the number of levels of the first mutable data objects above the changed data object, or 0 of the changed data object is mutable.

Raises
ValueError if no mutable parent object is in data structure (obj).

Raises
AttributeError e.g. if obj is of type int, and *int* is missing in leaf_types.

Note: make sure to prevent overwrites on internal objects of the Python runtime, on some of them the Python interpreter could even crash (e.g. with: exit code 134 (interrupted by signal 6: SIGABRT)).

---

deep_search(*obj*, *found*, *leaf_types=(<class 'bytes'>, <class 'int'>, <class 'float'>, <class 'set'>, <class 'str'>, <class 'type'>)*, *key_filter=<function key_filter_default>*, *value_filter=<function value_filter_default>*, *obj_keys=None*)

search key and/or value within the passed (deeply nested) data object structure.

Parameters

- **obj** (Any) – root object to start the top-down deep search from, which can contain any combination of deep nested elements/objects. for each sub-element the callable passed into *found* will be executed. if the callable returns True then the data path, the key and the value will be stored in a tuple and added to the search result list (finally returned to the caller of this function).

  for iterable objects of type dict/tuple/list, the sub-items will be searched, as well as the attributes determined via the Python dir() function. to reduce the number of items/attributes to be searched use the parameters *leaf_types* and/or *key_filter*.
- **found** (Callable[[List[Tuple[Any, Any]], Any, Any], Any]) – called for each item with 3 arguments (data-struct-path, key in data-structure, value), and if the return value is True then the data/object path, the last key and value will be added as a new item to the returned list.
- **leaf_types** (Tuple[Type, ...]) – tuple of leaf types to skip from to be searched deeper. the default value of this parameter is specified in the modul constant *DEFAULT_LEAF_TYPES*.
- **key_filter** (Callable[[Union[str, int, tuple]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for item-key/item-index/attribute-name to be filtered out. by default all attribute/key names starting with an underscore character will be filtered out (see default callable *key_filter_default()*).

- **value_filter**¶ (Callable[[Any, List[Tuple[Any, Any]]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for items/attributes values to be filtered out. by default empty values, excluded values (see EXCLUDED_VALUE_TYPES) and already scanned objects will be filtered out (see default callable *value_filter_default()*).
- **obj_keys**¶ (Optional[List[Tuple[Any, Any]]]) – used (internally only) to pass the parent data-struct path in recursive calls.

Return type
List[Tuple[List[Tuple[Any, Any]], Any, Any]]

Returns
list of tuples (data-struct-path, key, value); one tuple for each found item within the passed *obj* argument. an empty list will be returned if no item got found.

deep_update(*obj*, *updating_obj*, *leaf_types=(<class 'bytes'>, <class 'int'>, <class 'float'>, <class 'set'>, <class 'str'>, <class 'type'>)*, *key_filter=<function key_filter_default>*, *value_filter=<function value_filter_default>*, *obj_keys=None*)

merge the updating_obj data structure into the similar structured node object obj.

Parameters

- **obj**¶ (Any) – deep data object to update.
- **updating_obj**¶ (Any) – data structure similar structured like the obj argument with update values. a UNSET value will delete the item from the obj argument.
- **leaf_types**¶ (Tuple[Type, ...]) – tuple of leaf types to skip from to be searched deeper. the default value of this parameter is specified in the modul constant *DEFAULT_LEAF_TYPES*.
- **key_filter**¶ (Callable[[Union[str, int, tuple]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for item-key/item-index/attribute-name to be filtered out. by default all attribute/key names starting with an underscore character will be filtered out (see default callable *key_filter_default()*).
- **value_filter**¶ (Callable[[Any, List[Tuple[Any, Any]]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for items/attributes values to be filtered out. by default empty values, excluded values (see EXCLUDED_VALUE_TYPES) and already scanned objects will be filtered out (see default callable *value_filter_default()*).
- **obj_keys**¶ (Optional[List[Tuple[Any, Any]]]) – used (internally only) to pass the parent data-struct path in recursive calls.

key_list_object(*obj_keys*)

determine object in a deep nested data structure via an object key list.

Parameters
obj_keys¶ (List[Tuple[Any, Any]]) – object key list.

Return type
Any

Returns
recalculated object referenced by the first object and the keys of obj_keys or *~ae.base.UNSET* if not found.

Raises
TypeError if key does not match the object type in any item of obj_keys. ValueError if obj_keys is not of type *ObjKeysType*.

Hint: to include changes on immutable data structures, the returned object value gets recalculated, starting from the first object (`obj_keys`[0][0]), going deeper via the key only (while ignoring all other child objects in the object key list specified by :paramref:`obj_keys`).

key_path_object(*obj*, *key_path*, *new_value=<ae.base.UnsetType object>*)

determine object in a deep nested data structure via a key path string, and optionally assign a new value to it.

Parameters

- **obj** (Any) – initial data object to search in (and its sub-objects).
- **key_path** (str) – composed key string containing dict keys, tuple/list/str indexes and object attribute names, separated by a dot character, like shown in the following examples:

```
>>> class AClass:
...     str_attr_name_a = "a_attr_val"
...     dict_attr = {'a_str_key': 3, 999: "value_with_int_key", '999
↪': "..str_key"}
```

```
>>> class BClass:
...     str_attr_name_b = "b_b_b_b_b"
...     b_obj = AClass()
```

```
>>> b = BClass()
```

```
>>> assert key_path_object(b, 'str_attr_name_b') == "b_b_b_b_b"
>>> assert key_path_object(b, 'b_obj.str_attr_name_a') == "a_attr_val
↪"
>>> assert key_path_object(b, 'b_obj.str_attr_name_a.5') == "r"  #␣
↪6th chr of a_attr_val
>>> assert key_path_object(b, 'b_obj.dict_attr.a_str_key') == 3
```

the item key or index value of lists and dictionaries can alternatively be specified in Python syntax, enclosed in [and]:

```
>>> assert key_path_object(b, 'b_obj.dict_attr["a_str_key"]') == 3
>>> assert key_path_object(b, 'b_obj.dict_attr[\'a_str_key\']') == 3
>>> assert key_path_object(b, 'b_obj.dict_attr[999]') == "value_with_
↪int_key"
>>> assert key_path_object(b, 'b_obj.dict_attr["999"]') == "..str_key
↪"
```

only dict key strings that are not can be misinterpreted as number can be specified without the high commas (enclosing the key string), like e.g.:

```
>>> assert key_path_object(b, 'b_obj.dict_attr[a_str_key]') == 3
```

- **new_value** (Optional[Any]) – optional new value - replacing the found object. the old object value will be returned.

Note: immutable objects, like tuples, that are embedding in `obj` will be automatically updated/replaced up in the data tree structure until a mutable object (list, dict or object) get

found.

Return type
Any

Returns
specified object/value (the old value if *new_value* got passed) or *UNSET* if not found/exists (key path string is invalid).

key_path_string(*obj_keys*)

convert obj keys path into deep object key path string.

Parameters
obj_keys (List[Tuple[Any, Any]]) – object key list to convert.

Return type
str

Returns
key path string of the object keys path specified by the obj_keys argument.

key_value(*key_str*)

convert key string (mapping key, sequence index, obj attribute) into its value (str, int, float, tuple, ..).

Return type
Any

next_key_of_path(*key_path*)

parse key_path to determine the next item key/index, the path separator and the rest of the key path.

Parameters
key_path (str) – data object key/index path string to parse.

Return type
Tuple[Any, str, str]

Returns
tuple of key/index, the separator character and the (unparsed) rest of the key path (possibly an empty string).

Raises
IndexError if the argument of key_path is an empty string.

object_items(*obj*, *leaf_types=(<class 'bytes'>, <class 'int'>, <class 'float'>, <class 'set'>, <class 'str'>, <class 'type'>)*, *key_filter=<function key_filter_default>*)

determine the items of a data object, with mapping keys, sequence indexes or attribute names and its values.

Parameters

- **obj** (Any) – data structure/node object (list, dict, set, tuple, data object, ...).
- **leaf_types** (Tuple[Type, ...]) – tuple of leaf types to skip from to be searched deeper. the default value of this parameter is specified in the modul constant *DEFAULT_LEAF_TYPES*.
- **key_filter** (Callable[[Union[str, int, tuple]], bool]) – called for each sub-item/-attribute of the data structure specified by obj. return True for item-key/item-index/attribute-name to be filtered out. by default all attribute/key names starting with an underscore character will be filtered out (see default callable *key_filter_default()*).

Return type
List[Tuple[Any, Any]]

Returns
items view of the data object specified in the `obj` argument.

`object_item_value`(*obj*, *key*, *default_value=<ae.base.UnsetType object>*)

determine value of a data object item/attribute.

Parameters

- **obj** (Any) – data structure object to get item/attribute value from.
- **key** (Any) – mapping key, attribute name or sequence index of the item/attribute.
- **default_value** (Any) – default value to return if the item/attribute does not exist in `obj`.

Return type
Any

Returns
data object item/attribute value or the `default_value` if not found.

`pop_object`(*obj_keys*)

delete sub-attribute/item of a data object.

Parameters
obj_keys (List[Tuple[Any, Any]]) – list of (object, key) tuples identifying an element within a deeply nested data structure or object hierarchy. the root of the data/object structure is the object at list index 0 and the element to be deleted is identified by the object and key in the last list item of this argument.

the referenced data structure can contain even immutable node objects (like tuples) which will be accordingly changed/replaced if affected/needed. for that at least one object/element above the immutable object in the deep data structure has to be mutable, else a ValueError will be raised.

Return type
List[Any]

Returns
list of deleted values or UNSET if not found.

Raises
ValueError if no immutable parent object got found or if the `obj_keys' is empty. `IndexError if the one of the specified indexes in :paramref:`obj_keys` does not exist.

`replace_object`(*obj_keys*, *new_value*)

set sub-attribute/item with a mutable parent object to a new value within a deeply nested object/data structure.

Parameters

- **obj_keys** (List[Tuple[Any, Any]]) – list of (object, key) tuples identifying an element within a deeply nested data structure or object hierarchy. the root of the data/object structure is the object at list index 0 and the element to be changed is identified by the object and key in the last list item of this argument.

  the referenced data structure can contain immutable data node objects (like tuples) which will be accordingly changed/replaced if affected/needed.

  at least one object/element, situated above of replaced data object within the deep data structure, has to be mutable, else a ValueError will be raised.

- **new_value** (Any) – value to be assigned to the element referenced by the last list item of the argument in *obj_keys*.

Return type
int

Returns
the number of levels of the first mutable data objects above the changed data object, or 0 of the changed data object is mutable.

Raises
ValueError if no immutable parent object got found or if the object key list in the `obj_keys` argument is empty.

4.3 ae.django_utils

4.3.1 helpers for django projects

this module is providing helper functions for your django projects.

Functions

generic_language(lang_code)	remove the country specific part - if exists - from the passed language code.
requested_language()	determine the currently requested short translation language code, fallback to english/'en' if unset.
set_url_part(url[, fragment, query])	add or replace fragment and/or query string of url.

generic_language(*lang_code*)

remove the country specific part - if exists - from the passed language code.

Parameters
lang_code (str) – lower-case language code with optional hyphen and country specific part.

Return type
str

Returns
stripped/short language code (e.g. return “en” if lang_code == “en-gb”)

requested_language()

determine the currently requested short translation language code, fallback to english/’en’ if unset.

Return type
str

Returns
short (without country specific part) and lower-case language code

set_url_part(*url*, *fragment=''*, *query=''*, ***set_part_values*)

add or replace fragment and/or query string of url.

Parameters

- **url** (str) – url to complete by adding and/or updating parts of it.
- **fragment** (str) – id of the fragment to add/replace.
- **query** (str) – query string to add/replace.

- **set_part_values**¶ – other parts like ‘scheme’, ‘netloc’, ‘path’, ‘params’ (see :func:urllib.parse.urlparse).

Return type
str

Returns
url with added/replaced parts.

4.4 ae.droid

4.4.1 android constants and helper functions

to include this ae namespace portion into your project add the following import statement into your main module (main.py) of your application project:

```
import ae.droid
```

this import will ensure that all your permissions will be requested on app startup. on other platforms than Android it will have no effect.

big thanks to Robert Flatt for his investigations, findings and documentations to code and build Kivy apps for the Android OS, and to Gabriel Pettier for his service osc example.

links to android service examples and documentation:

- https://github.com/tshirtman/kivy_service_osc/blob/master/src/main.py
- https://python-for-android.readthedocs.io/en/latest/services/#arbitrary-scripts-services
- https://blog.kivy.org/2014/01/building-a-background-application-on-android-with-kivy/

4.5 ae.notify

4.5.1 send notifications via email, telegram or whatsapp

this pure python module depends mainly on the Standard Python Libraries `email` and `smtplib` and the external `requests` module.

an instance of the *Notifications* has to be created for each notification sender in your app, providing the sender’s credentials for each used notification channel (service).

the notification channels and the receiver(s) can be specified individually for each notification message to send.

Module Attributes

DEF_ENC_PORT	standard SMTP port
DEF_ENC_SERVICE_NAME	standard SMTP service name
SSL_ENC_PORT	port to use SMTP via SSL
SSL_ENC_SERVICE_NAME	service name in *Notifications.smtp_server_uri* of SMTP via SSL
TLS_ENC_PORT	port to use SMTP via TLS
TLS_ENC_SERVICE_NAME	service name in *Notifications.smtp_server_uri* of SMTP via TLS
TELEGRAM_MESSAGE_MAX_LEN	maximum length of Telegram notification message body
WHATSAPP_MESSAGE_MAX_LEN	maximum length of Whatsapp notification message body

Classes

Notifications([smtp_server_uri, mail_from, ...])	a single instance of this class can be used to handle all notifications of an app/service.

DEF_ENC_PORT = 25

standard SMTP port

DEF_ENC_SERVICE_NAME = 'smtp'

standard SMTP service name

SSL_ENC_PORT = 465

port to use SMTP via SSL

SSL_ENC_SERVICE_NAME = 'smtps'

service name in *Notifications.smtp_server_uri* of SMTP via SSL

TLS_ENC_PORT = 587

port to use SMTP via TLS

TLS_ENC_SERVICE_NAME = 'smtpTLS'

service name in *Notifications.smtp_server_uri* of SMTP via TLS

TELEGRAM_MESSAGE_MAX_LEN = 4096

maximum length of Telegram notification message body

WHATSAPP_MESSAGE_MAX_LEN = 65536

maximum length of Whatsapp notification message body

_body_mime_type_conversion(*msg_body*, *mime_type*)

convert content of notification message body.

Parameters

- **msg_body** (str) – message body string.
- **mime_type** (str) – mime type to convert to, if it has the "to" prefix in front of the resulting mime type.

Return type
Tuple[str, str]

Returns
tuple of converted message body and resulting mime type (removing the "to" prefix).

class Notifications(*smtp_server_uri=''*, *mail_from=''*, *local_mail_host=''*, *telegram_token=''*, *whatsapp_token=''*, *whatsapp_sender=''*)

Bases: object

a single instance of this class can be used to handle all notifications of an app/service.

__init__(*smtp_server_uri=''*, *mail_from=''*, *local_mail_host=''*, *telegram_token=''*, *whatsapp_token=''*, *whatsapp_sender=''*)

initialize one or more different services for a sender of multiple notifications to individual receivers.

Parameters

- **smtp_server_uri** (str) – host and optional port and user credentials of email SMTP server to use, in the format [service://{]}{[}user{[}:password{]}@{]}mail_server_host{[}:mail_server_port]. default SMTP ports: 25/DEF_ENC_PORT, port 587/TSL_ENC_PORT for E-SMTP/TLS or 465/SSL_ENC_PORT for smtps/SSL.
- **mail_from** (str) – email sender address.
- **local_mail_host** (str) – FQDN of the local email host in the SMTP HELO/EHLO command.
- **telegram_token** (str) – token for the Telegram cloud API obtained from the @BotFather bot.
- **whatsapp_token** (str) – token for the WhatsApp cloud API obtained from the developer portal.
- **whatsapp_sender** (str) – sender phone number id for the WhatsApp cloud API obtained from the developer portal.

send_notification(*msg_body*, *receiver*, *subject=''*, *mime_type='to_html'*)

send a notification message with optional subject to receiver via the specified service.

Parameters

- **msg_body** (str) – message body. line breaks are converted (br-tag <-> newline character) in accordance with mime_type.
- **receiver** (str) – receiver address in the format `service:address=name`, where `service` is mailto, telegram or whatsapp, `address` is an email address, a chat id or phone number and `name` is the name of the receiving person.
- **subject** (str) – optional subject text. added to the top of the msg_body for messenger services, separated by an empty line. if not specified or specified as empty string or as a single space character, then it will not be added to the top of the message body.
- **mime_type** (str) – mime type ('html' or 'plain'), and optional conversion to it (if starts with 'to').

Return type
str

Returns
error message on error or empty string if notification got send successfully.

send_email(*msg_body*, *address*, *subject*, *name*, *mime_type='to_html'*)

send email to the passed address.

Parameters

- **msg_body** (str) – message body text. for new lines use newline char in plain and
 in html mime_type.
- **address** (str) – email address of the receiver.
- **subject** (str) – email subject text.
- **name** (str) – name of the receiver.
- **mime_type** (str) – mime type ('html' or 'plain'), and optional conversion to it (if starts with 'to').

Return type
str

Returns
error message on error or empty string if notification email got send successfully.

send_telegram(*msg_body*, *chat_id*, *name*, *mime_type='to_html'*)

send message to the passed telegram chat id.

Parameters

- **msg_body** (str) – message body text. in 'html' mime_type message texts are only a few tags support by Telegram (see https://core.telegram.org/bots/api#html-style). on top of that you can also include the following tags, which will be either converted or removed: * br: will be converted into a new line character. * div: will be removed.
- **chat_id** (str) – chat id of the telegram receiver or group.
- **name** (str) – name of the receiver.
- **mime_type** (str) – mime type ('html' or 'plain'), and optional conversion to it (if starts with 'to').

Return type
str

Returns
error message on error or empty string if notification got send successfully.

Hint: see https://www.heise.de/select/ct/2023/8/2231816070982959290 for useful bots and tips. use https://telemetr.io/, https://lyzem.com/ or https://tgstat.com/ to search/find public channels/groups.

---

send_whatsapp(*msg_body*, *receiver_id*, *name*, *mime_type='to_html'*)

send message to the passed WhatsApp user/group.

Parameters

- **msg_body** (str) – message body text. for new lines use newline char in plain and
 in html mime_type.
- **receiver_id** (str) – phone number with country code (and a leading '+') of the WhatsApp receiver or the id of the WA group (last part of the URL to invite somebody into the group).
- **name** (str) – name of the receiver.

- **mime_type** (str) – mime type ('html' or 'plain'), and optional conversion to it (if starts with 'to'). recognized/converted html tags are b, br, i and pre.

Return type
str

Returns
error message on error or empty string if notification got send successfully.

using WA Business API (see: https://developers.facebook.com/docs/whatsapp/on-premises/reference/messages and https://github.com/Neurotech-HQ/heyoo/blob/58ad576c3dfaf05bad5f342bc8614cf0ba02e6ae/heyoo/__init__.py#L43) has the restriction that the receiver has first to send a message to the sender to get a window of 24 hours. and using pyWhatKit's webbrowser-based approach will not work on PythonAnywhere because web.whatsapp.com is not in their whitelist (https://www.pythonanywhere.com/whitelist/)

4.6 ae.valid

4.6.1 data validation helper functions

this module is pure Python and has no dependencies.

the two slightly bigger helper functions provided by this namespace portion are *correct_email()* and *correct_phone()*, which are useful to check if a string contains a valid email address or phone number.

they also allow you to automatically correct an email address or a phone number to a valid format. more sophisticated helpers for the validation of email addresses, phone numbers and post addresses are available in the `ae.validation` namespace portion.

Functions

correct_email(email[, changed, removed])	check and correct email address from a user input (removing all comments).
correct_phone(phone[, changed, removed, ...])	check and correct phone number from a user input (removing all invalid characters including spaces).

correct_email(*email*, *changed=False*, *removed=None*)

check and correct email address from a user input (removing all comments).

special conversions that are not returned as changed/corrected are: the domain part of an email will be corrected to lowercase characters, additionally emails with all letters in uppercase will be converted into lowercase.

regular expressions are not working for all edge cases (see the answer to this SO question: https://stackoverflow.com/questions/201323/using-a-regular-expression-to-validate-an-email-address) because RFC822 is very complex (even the reg expression recommended by RFC 5322 is not complete; there is also a more readable form given in the informational RFC 3696). additionally a regular expression does not allow corrections. therefore this function is using a procedural approach (using recommendations from RFC 822 and https://en.wikipedia.org/wiki/Email_address).

Parameters

- **email** (str) – email address to check and correct.
- **changed** (bool) – optional flag if email address got changed (before calling this function) - will be returned unchanged if email did not get corrected.

- **removed** (Optional[List[str]]) – optional list declared by caller to pass back all the removed characters including the index in the format "<index>:<removed_character(s)>".

Return type
Tuple[str, bool]

Returns
tuple of (possibly corrected email address, flag if email got changed/corrected).

correct_phone(*phone*, *changed=False*, *removed=None*, *keep_1st_hyphen=False*)

check and correct phone number from a user input (removing all invalid characters including spaces).

Parameters

- **phone** (str) – phone number to check and correct.
- **changed** (bool) – optional flag if phone got changed (before calling this function) - will be returned unchanged if phone did not get corrected.
- **removed** (Optional[List[str]]) – optional list declared by caller to pass back all the removed characters including the index in the format "<index>:<removed_character(s)>".
- **keep_1st_hyphen** (bool) – pass True to keep at least the first occurring hyphen character.

Return type
Tuple[str, bool]

Returns
tuple of (possibly corrected phone number, flag if phone got changed/corrected).

4.7 ae.files

4.7.1 generic file object helpers

this namespace portion is pure Python providing helpers for file object and content managing. it only depends on the *ae.base* namespace portion.

Hint: more helper functions to manage directory/folder structures are provided by the *ae.paths* portion.

the helper function *copy_bytes()* provides recoverable copies of binary files and file streams, with progress callbacks for each copied bytes chunk/buffer.

file_lines() and *read_file_text()* are helpers to read/load text file contents.

the function *write_file_text()* stores a string to a text file.

the helper function *file_transfer_progress()* puts the amount of transferred bytes in a short and user readable format, to be displayed as progress string in a file transfer progress.

RegisteredFile and *CachedFile* encapsulate and optionally cache the contents of a file within a file object. instances of these classes are compatible with the file objects provided by Python's pathlib module. but also pure path strings can be used as file objects (see also the *FileObject* type).

all these types of file objects are supported by the files register class *FilesRegister* from the *ae.paths* portion.

registered file

a registered file object represents a single file on your file system and can be instantiated from one of the classes *RegisteredFile* or *CachedFile* provided by this module/portion:

```
from ae.files import RegisteredFile

rf = RegisteredFile('path/to/the/file_name.extension')

assert str(rf) == 'path/to/the/file_name.extension'
assert rf.path == 'path/to/the/file_name.extension'
assert rf.stem == 'file_name'
assert rf.ext == '.extension'
assert rf.properties == {}
```

file properties will be automatically attached to each file object instance with the instance attribute *properties*. in the last example it results in an empty dictionary because the *path* of this file object does not contain folder names with an underscore character.

file properties

file property names and values are automatically determined from the names of their sub-folders, specified in the *path* attribute. every sub-folder name containing an underscore character in the format <property-name>_<value> will be interpreted as a file property:

```
rf = RegisteredFile('property1_69/property2_3.69/property3_whatever/file_name.ext')
assert rf.properties['property1'] == 69
assert rf.properties['property2'] == 3.69
assert rf.properties['property3'] == 'whatever'
```

the property types *int*, *float* and *string* are recognized and converted into a property value. boolean values can be specified as `1` and `0` integers.

cached file

a cached file created from the *CachedFile* behaves like a *registered file* and additionally provides the possibility to cache parts or the whole file content as well as the file pointer of the opened file:

```
cf = CachedFile('integer_69/float_3.69/string_whatever/file_name.ext')

assert str(cf) == 'integer_69/float_3.69/string_whatever/file_name.ext'
assert cf.path == 'integer_69/float_3.69/string_whatever/file_name.ext'
assert cf.stem == 'file_name'
assert cf.ext == '.ext'
assert cf.properties['integer'] == 69
assert cf.properties['float'] == 3.69
assert cf.properties['string'] == 'whatever'
```

pn instantiation of the *CachedFile* file object the default file object loader function *_default_object_loader()* will be used, which opens a file stream via Python's `open()` built-in. alternatively you can specify a specific file object loader with the *object_loader* parameter or by assigning a callable directly to the `object_loader` attribute:

```
cf = CachedFile('integer_69/float_3.69/string_whatever/file_name.ext',
                object_loader=lambda cached_file_obj: my_open_method(cached_file_obj.
↪path))
```

the cached file object is accessible via the *loaded_object* attribute of the cached file object instance:

```
assert isinstance(cf.loaded_object, TextIOWrapper)
cf.loaded_object.seek(...)
cf.loaded_object.read(...)

cf.loaded_object.close()
```

Module Attributes

FileObject	file object type, e.g.
PropertyType	types of file property values
PropertiesType	dict of file properties
FilenameOrStream	file name or file stream pointer

Functions

copy_bytes(src_file, dst_file, *[, ...])	recoverable copy of a file or stream (file-like object), optionally with progress callbacks.
file_lines(file_path[, encoding])	returning lines of the text file specified by file_path argument as tuple.
file_transfer_progress(transferred_bytes[, ...])	return string to display the transfer progress of transferred bytes in short and user readable format.
read_file_text(file_path[, encoding, ...])	returning content of the text file specified by file_path argument as string.
write_file_text(text_or_lines, file_path[, ...])	write the passed text string or list of line strings into the text file specified by file_path argument.

Classes

CachedFile(file_path[, object_loader, ...])	represents a cacheables registered file object - see also *cached file* examples.
RegisteredFile(file_path, **kwargs)	represents a single file - see also *registered file* examples.

FileObject

file object type, e.g. a file path str or any class or callable where the returned instance/value is either a string or an object with a *stem* attribute (holding the file name w/o extension), like e.g. *CachedFile*, *RegisteredFile*, pathlib.Path or pathlib.PurePath.

alias of Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]

PropertyType

types of file property values

alias of Union[int, float, str]

PropertiesType

dict of file properties

alias of Dict[str, Union[int, float, str]]

FilenameOrStream

file name or file stream pointer

alias of Union[str, BinaryIO]

copy_bytes(*src_file*, *dst_file*, ***, *transferred_bytes=0*, *total_bytes=0*, *buf_size=16384*, *overwrite=False*, *move_file=False*, *recoverable=False*, *errors=None*, *progress_func=<function dummy_function>*, ***progress_kwargs*)

recoverable copy of a file or stream (file-like object), optionally with progress callbacks.

Parameters

- **src_file** (Union[str, BinaryIO]) – source file name or opened stream (file-like) object. if passing a non-seekable stream together with a non-zero value in *transferred_bytes* then the source stream has to be set to the correct position before you call this function. if passing any source stream then also the total file/stream size has to be passed into the *total_bytes* parameter. source file streams does also not support a True value in the *move_file* argument.
- **dst_file** (Union[str, BinaryIO]) – destination file name or opened stream (file-like) object. recoverable copies and copies with a True value in the *overwrite* argument are not allowed; always use a destination file name if you need a recoverable/overwriting copy.
- **transferred_bytes** (int) – file offset at which the copy process starts. if not passed for recoverable copies, then *copy_bytes* will determine this value from the file length of the destination file.
- **total_bytes** (int) – source file size in bytes (needed only if *src_file* is a stream).
- **buf_size** (int) – size of copy buffer/chunk in bytes (that get copied before each progress callback).
- **overwrite** (bool) – pass True to allow to overwrite of destination file. if the destination file exists already then this function will return an error (when this argument get not passed or is False).
- **move_file** (bool) – pass True to delete source file on complete copying (only works if source is a stream).
- **recoverable** (bool) – pass True to allow recoverable file copy (only working if source is a stream).
- **errors** (Optional[List[str]]) – pass empty list to get a list of detailed error messages.
- **progress_func** (Callable) – optional callback to dispatch or break/cancel the copy progress for large files. if the callback returns a non-empty value it will be interpreted as cancel reason, the copy process will be stopped and an error will be returned.
- **progress_kwargs** – optional additional kwargs passed to the progress function. the kwargs *total_bytes* and *transferred_bytes* will be updated before the callback.

Return type
str

Returns
destination file name/stream as string or empty string on error.

Hint: this function is extending the compatible Python functions shutil.copyfileobj(), shutil.copyfile(), shutil.copy(), shutil.copy2() and http.server.SimpleHTTPRequestHandler.copyfile() with recoverability and a progress callback. it can also be used as argument for the copy_function parameter of e.g. shutil.copytree() and shutil.move().

---

file_lines(*file_path*, *encoding=None*)

returning lines of the text file specified by file_path argument as tuple.

Parameters

- **file_path** (str) – file path/name to parse/load.
- **encoding** (Optional[str]) – encoding used to load and convert/interpret the file content.

Return type
Tuple[str, ...]

Returns
tuple of the lines found in the specified file or empty tuple if the file could not be found or opened.

file_transfer_progress(*transferred_bytes*, *total_bytes=0*, *decimal_places=3*)

return string to display the transfer progress of transferred bytes in short and user readable format.

Parameters

- **transferred_bytes** (int) – number of transferred bytes.
- **total_bytes** (int) – number of total bytes.
- **decimal_places** (int) – number of decimal places (should be between 0 and 3).

Return type
str

Returns
formatted string to display progress of currently running transfer.

read_file_text(*file_path*, *encoding=None*, *error_handling='ignore'*)

returning content of the text file specified by file_path argument as string.

Parameters

- **file_path** (str) – file path/name to load into a string.
- **encoding** (Optional[str]) – encoding used to load and convert/interpret the file content.
- **error_handling** (str) – pass *'strict'* or *None* to return *None* (instead of an empty string) for the cases where either a decoding *ValueError* exception or any *OSError*, *FileNotFoundError* or *PermissionError* exception got raised. the default value *'ignore'* will ignore any decoding errors (missing some characters) and will return an empty string on any file/os exception.

Return type
Optional[str]

Returns
file content string. if the file could not be decoded, found or opened, then return empty string or None (None only if *'strict'* got passed to the *error_handling* parameter).

write_file_text(*text_or_lines*, *file_path*, *encoding=None*)

write the passed text string or list of line strings into the text file specified by file_path argument.

Parameters

- **text_or_lines** (Union[str, List[str], Tuple[str]]) – new file content either passed as string or list of line strings (will be concatenated with the line separator of the current OS: os.linesep).
- **file_path** (str) – file path/name to write the passed content into (overwriting any previous content!).
- **encoding** (Optional[str]) – encoding used to write/convert/interpret the file content to write.

Return type
bool

Returns
True if the content got written to the file, False on any file/OS error.

class RegisteredFile(*file_path*, ***kwargs*)

Bases: object

represents a single file - see also *registered file* examples.

__init__(*file_path*, ***kwargs*)

initialize registered file_obj instance.

Parameters

- **file_path** (str) – file path string.
- **kwargs** – not supported, only there to have compatibility to *CachedFile* to detect invalid kwargs.

path: str

file path

stem: str

file basename without extension

ext: str

file name extension

properties: Dict[str, Union[int, float, str]]

file properties

__eq__(*other*)

allow equality checks.

Parameters
other (Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]) – other file object to compare this instance with.

Return type
bool

Returns
True if both objects are of this type and contain a file with the same path, else False.

`__repr__`()
for config var storage and eval recovery.

Returns
evaluable/recoverable representation of this object.

`__str__`()
return file path.

Returns
file path string of this file object.

`add_property`(*property_name*, *str_value*)
add a property to this file object instance.

Parameters
- **property_name** (str) – stem of the property to add.
- **str_value** (str) – literal of the property value (int/float/str type will be detected).

`__hash__ = None`

`_default_object_loader`(*file_obj*)
file object loader that is opening the file and keeping the handle of the opened file.

Parameters
file_obj (Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]) – file object (path string or obj with *path* attribute holding the complete file path).

Returns
file handle to the opened file.

`class CachedFile`(*file_path*, *object_loader=<function _default_object_loader>*, *late_loading=True*)
Bases: *RegisteredFile*

represents a cacheables registered file object - see also *cached file* examples.

`__init__`(*file_path*, *object_loader=<function _default_object_loader>*, *late_loading=True*)
create cached file object instance.

Parameters
- **file_path** (str) – path string of the file.
- **object_loader** (Callable[[*CachedFile*], Any]) – callable converting the file_obj into a cached object (available via *loaded_object*).
- **late_loading** (bool) – pass False to convert/load file_obj cache early, directly at instantiation.

path: str
file path

stem: str
file basename without extension

ext: str
file name extension

properties: Dict[str, Union[int, float, str]]

file properties

property loaded_object: Any

loaded object class instance property.

Returns

loaded and cached file object.

4.8 ae.paths

4.8.1 generic file path helpers

this pure python namespace portion is providing useful *path helper functions* as well as *generic system paths* for most platforms, like e.g.:

- android OS
- iOS
- linux
- macOS
- Windows

the only external hard dependency of this module are the ae namespace portions *ae.base* and *ae.files*. optional dependencies are:

- on android OS the PyPi package *jnius*, needed by the functions *user_data_path()* and *user_docs_path()*.
- the *plyer* PyPi package, needed by the function *add_common_storage_paths()*.

path helper functions

the generator functions *coll_files()*, *coll_folders()* and *coll_items()* are building the fundament for most of the file and folder collection functionality, provided by this module. for example the function *path_files()* is using these generators to determine the files within a folder structure that are matching the specified wildcards and *path part placeholders*. similarly the function *path_folders()* for folders, and *path_items()* to collect both, file and folder names.

use the functions *copy_files()* and *move_files()* to duplicate and move multiple files or entire file path trees. these two functions are based on *copy_file()* and *move_file()*. the functions *copy_tree()* and *move_tree()* providing an alternative way to copy or move entire directory trees.

the helper function *normalize()* converts path strings containing path placeholders into regular path strings, resolving symbolic links, or is converting paths string from absolute paths to relative paths and vice versa.

to determine if the path of a file or folder is matching a glob-like path pattern/mask with wildcards, the functions *path_match()* and *paths_match()* can be used. useful specially for cases where you don't have direct access to the file system.

file paths for series of files, e.g. for logging, can be determined via the *series_file_name()* function.

the function *skip_py_cache_files()* can be used in path file collections to skip the files situated in the Python cache folder (*PY_CACHE_FOLDER* respectively `__pycache__`).

generic system paths

generic system paths are determined by the following helper functions:

- *app_data_path()*: application data path.
- *app_docs_path()*: application documents path.
- *user_data_path()*: user data path.
- *user_docs_path()*: user documents path.

these system paths together with additional generic paths like e.g. the current working directory, storage paths provided by the *plyer* package, or user and application paths, are provided as *path placeholders*, which get stored within the *PATH_PLACEHOLDERS* dict by calling the function *add_common_storage_paths()*.

path_name() and *placeholder_path()* are converting regular path strings or parts of it into path placeholders.

file/folder collection and classification

more complex collections of files and folder paths, and the grouping of them, can be done with the classes *Collector*, described in the underneath section *collecting files*, and *FilesRegister*, described in the section *files register*.

use the *Collector class* for temporary quick file path searches on your local file systems as well as on remote servers/hosts. one implementation example is e.g. the method `deployed_code_files()` of the `aedev.pythonanywhere` module.

the class *FilesRegister* helps you to create and cache file path registers permanently, to quickly find at any time the best fitting match for a requested purpose. for example the *gui_app* module is using it to dynamically select image/font/audio/... resource files depending on the current user preferences, hardware and/or software environment.

collecting files

to collect file names in the current working directory create an instance of the *Collector* class and call its *collect()* method with a file or folder path, which can contain wildcards:

```
.. code-block:: python
```

from ae.paths import Collector coll = Collector() coll.collect('*.png') image_files_list = coll.files

after that a list containing the found file names can then be retrieved from the *files* attribute.

collect() can be called multiple times to accumulate and extend the *files* list:

```
.. code-block:: python
```

coll = Collector() coll.collect('*.png') coll.collect('*.jpg') image_files_list = coll.files

multiple calls of *Collector.collect()* can be joined into one code line, because it is returning its instance. the following statement is equivalent to the last example:

```
.. code-block:: python
```

image_files_list = Collector().collect('*.png').collect('*.jpg').files

by specifying the ** wildcard entire folder trees can be scanned. the following example is collection all the files, including the hidden ones, in the folder tree under the current working directory:

```
.. code-block:: python
```

test_files = Collector().collect('**/*').collect('/.*').files

Hint: the second call of the *Collector.collect()* method in this example has to be done only if you are using Python's `glob.glob()` as the searcher callback, which excludes hidden files (with a leading dot) from to match the * wildcard.

Collector is by default only returning files. to also collect folder paths in a deep folder tree, you have to pass the collecting generator function to the optional *item_collector* parameter of the *Collector* class. the accumulated files and folders can then be retrieved from their respective instance attributes *files*, *paths* and *selected*:

```
.. code-block:: python
```

coll = Collector(item_collector=coll_items) coll.collect(...) ... files = coll.files folders = coll.paths file_and_folder_items = coll.selected

the found files are provided by the *Collector.files* instance attribute. found folders will be separately collected within the *Collector* instance attribute *paths*. the *selected* attribute contains all found files and folders in a single list.

collect from multiple locations

in a single call of *collect()*, providing the method parameters *append* or *select*, you can scan multiple combinations of path prefixes and suffixes, which can both contain wildcards and folder names, whereas the suffixes containing also parts of the file names to search for.

Hint: the wildcards *, ** and *?* are allowed in the prefixes as well as in suffixes.

the resulting file paths are relative or absolute, depending on if the specified prefix(es) containing absolute or relative paths.

in the following example determines the relative paths of all folders directly underneath the current working directory with a name that contains the string *'xxx'* or is starting with *'yyy'* or is ending with *'zzz'*:

```
.. code-block:: python
```

coll = Collector(item_collector=coll_folders) coll.collect('', append=('*xxx*', 'yyy*', '*zzz')) folders = coll.paths

Hint: replace empty string in the first argument of *collect()* with '{cwd}' to get absolute paths.

the following example is collecting the absolute paths of files with the name *xxx.cfg* from the first found location/folder, starting to search in the current working directory, then in the folder above the application data folder, and finally in a folder with the name of the main application underneath the user data folder:

```
.. code-block:: python
```

coll = Collector() coll.collect("{cwd}", "{app}/..", "{usr}/{main_app_name}", append="xxx.cfg") found_files = coll.files

to set or change the generic path placeholder parts values, e.g. of the main application name (*{main_app_name}*) and the application data path (*{app}*), you simply specify their corresponding values as kwargs in the construction of the *Collector* instance:

```
.. code-block:: python
```

coll = Collector(main_app_name=..., app=...)

additionally you can specify any other path placeholders that will be automatically used and replaced by the *Collector* instance:

```
.. code-block:: python
```

coll = Collector(any_other_placeholder=...)

by default only the found file(s)/folder(s) of the first combination will be collected. to collect all files instead, pass an empty tuple to the method argument *only_first_of* of :meth:`~Collector.collect':

```
.. code-block:: python
```

coll.collect(..., append=..., [select=...,] only_first_of=())

add one of the strings *'prefix'*, *'append'* or *'select'* to the *only_first_of* tuple argument to collect only the files/folders of the first combination of the specified prefixes, append-suffixes and select-suffixes.

by using the *select* argument the found files and folders will additionally be collected in the *Collector* instance attribute *selected*.

combinations collected via the *select* argument that are not existing will be logged. the results are provided by the instance attributes *failed*, *prefix_failed* and *suffix_failed*.

files register

a files register is an instance of the *FilesRegister*, providing property based file collection and selection, which is e.g. used by the *ae.gui_app* ae namespace portion to find and select resource files like icon/image or sound files.

files can be collected from various places by a single instance of the class *FilesRegister*:

```
.. code-block:: python
```

from ae.paths import FilesRegister

file_reg = FilesRegister('first/path/to/collect') file_reg.add_paths('second/path/to/collect/files/from')

registered_file = file_reg.find_file('file_name')

in this example the *FilesRegister* instance collects all files that are existing in any sub-folders underneath the two provided paths. then the *find_file()* method will return a file object of type *RegisteredFile* of the last collected file with the stem (base name w/o extension) *'file_name'*.

multiple files with the same stem can be collected and registered e.g. with different formats, to be selected by the app by their different properties, which are specified in the folder names underneath the collected paths. assuming your application is providing an icon image in two sizes, provided within the following folder structure, situated in the current working directory:

```
resources/
    size_72/
        app_icon.jpg
```

(continues on next page)

(continued from previous page)

```
    size_150/
        app_icon.png
```

when you then create an instance of *FilesRegister* both image files from the *resources* folder will get registered, interpreting the sub-folder names (*size_**) as properties or attributes for the registered files, where *size* will result as the property name and the string after the underscore as the property value.

to retrieve the paths of the application image file with the size 72, call the *find_file()* method:

```
.. code-block:: python
```

file_reg = FilesRegister('resources') app_icon_image_path = file_reg.find_file('app_icon', dict(size=72))

as a shortcut you can alternatively call the object directly (leaving *.find_file* away):

```
.. code-block:: python
```

app_icon_image_path = file_reg('app_icon', dict(size=150))

the resulting file path in *app_icon_image_path* will be *"resources/size_72/app_icon.jpg"* in the penultimate example and *"resources/size_150/app_icon.png"* in the last example.

an instance of *FilesRegister* (*file_reg*) behaves like a dict object, where the item key is the file stem ('app_icon') and the item value is a list of instances of *RegisteredFile*. both files in the resources folder are provided as one dict item:

```
.. code-block:: python
```

file_reg = FilesRegister('resources') assert 'app_icon' in file_reg assert len(file_reg) == 1 assert len(file_reg['app_icon']) == 2 assert isinstance(file_reg['app_icon'][0], RegisteredFile)

for more complex selections you can use callables passed into the *property_matcher* and *file_sorter* arguments of *find_file()*.

Module Attributes

APPEND_TO_END_OF_FILE_LIST	special flag default value for the *first_index* argument of the *add_** methods of *FilesRegister* to append new file objects to the end of the name's register file object list.
INSERT_AT_BEGIN_OF_FILE_LIST	special flag default value for the *first_index* argument of the *add_** methods of *FilesRegister* to insert new file objects always at the begin of the name's register file object list.
COLLECTED_FOLDER	item type value for a folder|directory|node
CollArgType	argument type passed to most callable arguments of *coll_items()* except of *searcher*
CollCreatorReturnType	type of the return value of the *creator* callable, which will be returned to the caller
CollYieldType	type of collected item, which is either *COLLECTED_FOLDER* for a folder or the file extension for a file
CollYieldItems	type of collected item iterator yielding tuples of (CollYieldType, item[_path])
SearcherRetType	type of the return value of the callable *searcher* argument of *coll_items()*
SearcherType	type of the callable *searcher* argument of *coll_items()*
copy_file(src, dst, *[, follow_symlinks])	Copy data and metadata.
copy_tree(src, dst[, symlinks, ignore, ...])	Recursively copy a directory tree and return the destination directory.
move_file(src, dst[, copy_function])	Recursively move a file or directory to another location.
move_tree(src, dst[, copy_function])	Recursively move a file or directory to another location.
PATH_PLACEHOLDERS	placeholders of user-, os- and app-specific system paths and file name parts

Functions

add_common_storage_paths()	add common storage paths to *PATH_PLACEHOLDERS* depending on the operating system (OS).
app_data_path()	determine the os-specific absolute path of the {app} directory where user app data can be stored.
app_docs_path()	determine the os-specific absolute path of the {ado} directory where user documents app are stored.
coll_files(file_mask[, file_class])	determine existing file(s) underneath the folder specified by *file_mask*.
coll_folders(folder_mask[, folder_class])	determine existing folder(s) underneath the folder specified by *folder_mask*.
coll_item_type(item_path)	classify path item to be either file, folder or non-existent/excluded/skipped.
coll_items(item_mask[, searcher, selector, ...])	determine path-/file-like item(s) specified with optional wildcards by `item_mask`.
copy_files(src_folder, dst_folder[, ...])	copy files from src_folder into optionally created dst_folder, optionally overwriting destination files.
move_files(src_folder, dst_folder[, overwrite])	move files from src_folder into optionally created dst_folder, optionally overwriting destination files.
normalize(path[, make_absolute, ...])	normalize/transform path replacing *PATH_PLACEHOLDERS* and the tilde character (for home folder).
path_files(file_mask[, file_class])	determine existing file(s) underneath the folder specified by `file_mask`.
path_folders(folder_mask[, folder_class])	determine existing folder(s) underneath the folder specified by `folder_mask`.
path_items(item_mask[, selector, creator])	determine existing file/folder item(s) underneath the folder specified by `item_mask`.
path_join(*parts)	join path parts preventing trailing path separator if last part is empty string.
path_match(path, mask)	return True if the specified path matches the specified path mask/pattern.
path_name(path)	determine placeholder key name of the specified path.
paths_match(paths, masks)	filter the paths matching at least one of the specified glob-like wildcard masks.
placeholder_key(path)	determine *PATH_PLACEHOLDERS* key of specified path.
placeholder_path(path)	replace begin of path string with the longest prefix found in *PATH_PLACEHOLDERS*.
series_file_name(file_path[, digits, ...])	determine non-existent series file name with a unique series index.
skip_py_cache_files(file_path)	file exclude callback for the files under Python's cache folders.
user_data_path()	determine the os-specific absolute path of the {usr} directory where user data can be stored.
user_docs_path()	determine the os-specific absolute path of the {doc} directory where the user is storing the personal documents.

Classes

Collector([item_collector])	file/folder collector class
FilesRegister(*add_path_args[, ...])	file register catalog - see also *files register* examples.

APPEND_TO_END_OF_FILE_LIST = 9223372036854775807

special flag default value for the *first_index* argument of the *add_** methods of *FilesRegister* to append new file objects to the end of the name's register file object list.

INSERT_AT_BEGIN_OF_FILE_LIST = -9223372036854775807

special flag default value for the *first_index* argument of the *add_** methods of *FilesRegister* to insert new file objects always at the begin of the name's register file object list.

COLLECTED_FOLDER = None

item type value for a folder|directory|node

CollArgType = typing.Any

argument type passed to most callable arguments of *coll_items()* except of *searcher*

CollCreatorReturnType = typing.Any

type of the return value of the *creator* callable, which will be returned to the caller

CollYieldType

type of collected item, which is either *COLLECTED_FOLDER* for a folder or the file extension for a file

alias of Optional[str]

CollYieldItems

type of collected item iterator yielding tuples of (CollYieldType, item[_path])

alias of Iterable[tuple[Optional[str], Any]]

SearcherRetType

type of the return value of the callable *searcher* argument of *coll_items()*

alias of Iterable[Any]

SearcherType

type of the callable *searcher* argument of *coll_items()*

alias of Callable[[str], Iterable[Any]]

coll_item_type(*item_path*)

classify path item to be either file, folder or non-existent/excluded/skipped.

Parameters
item_path (str) – file/folder path string.

Return type
Optional[str]

Returns
COLLECTED_FOLDER for folders, the file extension for files or None if not found.

coll_items(*item_mask*, *searcher=functools.partial(<function glob>*, *recursive=True)*, *selector=<class 'str'>*, *type_detector=<function coll_item_type>*, *creator=<class 'str'>*, ***creator_kwargs*)

determine path-/file-like item(s) specified with optional wildcards by item_mask.

Parameters

- **item_mask**¶ (str) – file path mask with optional glob() wildcards, the '~' shortcut for the home folder path and any *path placeholders*, which is specifying the files/folders to collect. use the '**' glob as path to include also items from sub-folders deeper than one level.
- **searcher**¶ (Callable[[str], Iterable[Any]]) – callable to convert|resolve path with optional wildcards in item_mask into multiple item file/folder path strings.
- **type_detector**¶ (Callable[[Any], Optional[str]]) – callable to typify/classify a found item to be stored in the first tuple items of the returned list. if not passed then path_item_type() will be used.
- **selector**¶ (Callable[[Any], Union[bool, Any]]) – called with each found file/folder name to check if it has to be added to the returned list. the default argument (str) results in returning every file/folder found by glob().
- **creator**¶ (Callable[[Any], Any]) – each found file/folder will be passed as argument to this class/callable and the instance/return-value will be appended as an item to the returned item list. if not passed then the *str* class will be used, which means that the items of the returned list will be strings of the file/folder path and name. passing a class, like e.g. *ae.files.CachedFile*, *ae.files.CachedFile* or pathlib.Path, will create instances of this class. alternatively you can pass a callable which will be called on each found file/folder. in this case the return value of the callable will be inserted in the related item of the returned list. silly mypy does not support Union[Type[Any], Callable[[str, KwArg()], Any]].
- **creator_kwargs**¶ – additional/optional kwargs passed onto the used item_class apart from the item name.

Return type
Iterable[tuple[Optional[str], Any]]

Returns
iterator/generator yielding found and selected file system items as instances of the item creator class (passed as creator argument, defaulting to the *str* class).

coll_files(*file_mask*, *file_class=<class 'str'>*, ***file_kwargs*)

determine existing file(s) underneath the folder specified by *file_mask*.

Parameters

- **file_mask**¶ (str) – glob file mask (with optional glob wildcards and *PATH_PLACEHOLDERS*) specifying the files to collect (by default including the sub-folders).
- **file_class**¶ (Union[Type[Any], Callable]) – factory used for the returned list items (see *coll_items.creator*). silly mypy does not support Union[Type[Any], Callable[[str, KwArg()], Any]].
- **file_kwargs**¶ – additional/optional kwargs apart from file name passed onto the used item_class.

Return type
Iterable[tuple[Optional[str], Any]]

Returns
list of files of the class specified by *file_mask*.

coll_folders(*folder_mask*, *folder_class=<class 'str'>*, ***folder_kwargs*)

determine existing folder(s) underneath the folder specified by *folder_mask*.

Parameters

- **folder_mask**¶ (str) – glob folder mask (with optional glob wildcards and *PATH_PLACEHOLDERS*) specifying the folders to collect (by default including the sub-folders).
- **folder_class**¶ (Union[Type[Any], Callable]) – class or factory used for the returned list items (see *creator*). silly mypy does not support Union[Type[Any], Callable[[str, KwArg()], Any]].
- **folder_kwargs**¶ – additional/optional kwargs apart from file name passed onto the used item_class.

Return type
Iterable[tuple[Optional[str], Any]]

Returns
list of folders of the class specified by *folder_mask*.

add_common_storage_paths()

add common storage paths to *PATH_PLACEHOLDERS* depending on the operating system (OS).

the following storage paths are provided by the *plyer* PyPi package (not all of them are available in each OS):

- *application*: user application directory.
- *documents*: user documents directory.
- *downloads*: user downloads directory.
- *external_storage*: external storage root directory.
- *home*: user home directory.
- *music*: user music directory.
- *pictures*: user pictures directory.
- *root*: root directory of the operating system partition.
- *sdcard*: SD card root directory (only available in Android if sdcard is inserted).
- *videos*: user videos directory.

additionally storage paths that are only available on certain OS (inspired by the method *get_drives*, implemented in https://github.com/kivy-garden/filebrowser/blob/master/kivy_garden/filebrowser/__init__.py):

- *Linux*: external storage devices/media mounted underneath the system partition root in /mnt or /media.
- *Apple Mac OsX or iOS*: external storage devices/media mounted underneath the system partition root in /Volume.
- *MS Windows*: additional drives mapped as the drive partition name.

app_data_path()

determine the os-specific absolute path of the {app} directory where user app data can be stored.

Hint: use *app_docs_path()* instead to get a more public path to the user.

Return type
str

Returns
path string of the user app data folder.

app_docs_path()

determine the os-specific absolute path of the {ado} directory where user documents app are stored.

Hint: use *app_data_path()* instead to get a more hidden path to the user.

Return type
str

Returns
path string of the user documents app folder.

copy_file(*src*, *dst*, *, *follow_symlinks=True*)

alias for shutil.copy2() (compatible to shutil.copy(), shutil.copyfile() and *ae.files.copy_bytes()*.

copy_tree(*src*, *dst*, *symlinks=False*, *ignore=None*, *copy_function=<function copy2>*, *ignore_dangling_symlinks=False*, *dirs_exist_ok=False*)

alias for shutil.copytree().

move_file(*src*, *dst*, *copy_function=<function copy2>*)

alias for shutil.move() (see also *move_tree()*).

move_tree(*src*, *dst*, *copy_function=<function copy2>*)

another alias for shutil.move() (see also *move_file()*).

copy_files(*src_folder*, *dst_folder*, *overwrite=False*, *copier=<function copy2>*)

copy files from src_folder into optionally created dst_folder, optionally overwriting destination files.

Parameters

- **src_folder** (str) – path to source folder/directory where the files get copied from. placeholders in *PATH_PLACEHOLDERS* will be recognized and substituted.
- **dst_folder** (str) – path to destination folder/directory where the files get copied to. all placeholders in *PATH_PLACEHOLDERS* are recognized and will be substituted.
- **overwrite** (bool) – pass True to overwrite existing files in the destination folder/directory. on False the files will only get copied if they not exist in the destination.
- **copier** (Callable) – copy/move function with src_file and dst_file parameters, returning file path/name.

Return type
List[str]

Returns
list of copied files, with their destination path.

move_files(*src_folder*, *dst_folder*, *overwrite=False*)

move files from src_folder into optionally created dst_folder, optionally overwriting destination files.

Parameters

- **src_folder** (str) – path to source folder/directory where the files get moved from. placeholders in *PATH_PLACEHOLDERS* will be recognized and substituted. please note that the source folders itself will neither be moved nor removed (but will be empty after the operation finished).

- **dst_folder**¶ (str) – path to destination folder/directory where the files get moved to. all placeholders in *PATH_PLACEHOLDERS* are recognized and will be substituted.
- **overwrite**¶ (bool) – pass True to overwrite existing files in the destination folder/directory. on False the files will only get moved if they not exist in the destination.

Return type
List[str]

Returns
list of moved files, with their destination path.

normalize(*path*, *make_absolute=True*, *remove_base_path=''*, *remove_dots=True*, *resolve_sym_links=True*)

normalize/transform path replacing *PATH_PLACEHOLDERS* and the tilde character (for home folder).

Parameters

- **path**¶ (str) – path string to normalize/transform.
- **make_absolute**¶ (bool) – pass False to not convert path to an absolute path.
- **remove_base_path**¶ (str) – pass a valid base path to return a relative path, even if the argument values of *make_absolute* or *resolve_sym_links* are *True*.
- **remove_dots**¶ (bool) – pass False to not replace/remove the . and .. placeholders.
- **resolve_sym_links**¶ (bool) – pass False to not resolve symbolic links, passing True implies a *True* value also for the *make_absolute* argument.

Return type
str

Returns
normalized path string: absolute if *remove_base_path* is empty and either *make_absolute* or *resolve_sym_links* is *True*; relative if *remove_base_path* is a base path of *path* or if *path* got passed as relative path and neither *make_absolute* nor *resolve_sym_links* is *True*.

path_files(*file_mask*, *file_class=<class 'str'>*, ***file_kwargs*)

determine existing file(s) underneath the folder specified by `file_mask`.

Parameters

- **file_mask**¶ (str) – glob file mask (with optional glob wildcards and *PATH_PLACEHOLDERS*) specifying the files to collect (by default including the sub-folders).
- **file_class**¶ (Union[Type[Any], Callable]) – factory used for the returned list items (see *path_items.creator*). silly mypy does not support Union[Type[Any], Callable[[str, KwArg()], Any]].
- **file_kwargs**¶ – additional/optional kwargs apart from file name passed onto the used item_class.

Return type
List[Any]

Returns
list of files of the class specified by `file_mask`.

path_folders(*folder_mask*, *folder_class=<class 'str'>*, ***folder_kwargs*)

determine existing folder(s) underneath the folder specified by `folder_mask`.

Parameters

- **folder_mask** (str) – glob folder mask (with optional glob wildcards and *PATH_PLACEHOLDERS*) specifying the folders to collect (by default including the sub-folders).
- **folder_class** (Union[Type[Any], Callable]) – class or factory used for the returned list items (see *creator*). silly mypy does not support Union[Type[Any], Callable[[str, KwArg()], Any]].
- **folder_kwargs** – additional/optional kwargs apart from file name passed onto the used item_class.

Return type
List[Any]

Returns
list of folders of the class specified by `folder_mask`.

path_items(*item_mask*, *selector=<class 'str'>*, *creator=<class 'str'>*, ***creator_kwargs*)

determine existing file/folder item(s) underneath the folder specified by `item_mask`.

Parameters

- **item_mask** (str) – file path mask (with optional glob wildcards and *PATH_PLACEHOLDERS*) specifying the files/folders to collect.
- **selector** (Callable[[str], Any]) – called with each found file/folder name to check if it has to be added to the returned list. the default argument (str) results in returning every file/folder found by glob().
- **creator** (Union[Type[Any], Callable]) – each found file/folder will be passed as argument to this class/callable and the instance/return-value will be appended as an item to the returned item list. if not passed then the *str* class will be used, which means that the items of the returned list will be strings of the file/folder path and name. passing a class, like e.g. *ae.files.CachedFile*, *ae.files.CachedFile* or `pathlib.Path`, will create instances of this class. alternatively you can pass a callable which will be called on each found file/folder. in this case the return value of the callable will be inserted in the related item of the returned list. silly mypy does not support Union[Type[Any], Callable[[str, KwArg()], Any]].
- **creator_kwargs** – additional/optional kwargs passed onto the used item_class apart from the item name.

Return type
List[Any]

Returns
list of found and selected items of the item class (`item_mask`).

path_join(**parts*)

join path parts preventing trailing path separator if last part is empty string.

Parameters
parts (str) – path parts to join.

Return type
str

Returns
joined path string.

Hint: although `os.path.join()` is implemented in C, this function is faster:: .. code-block:: python

import os import timeit from ae.paths import path_join paths_secs = timeit.timeit(‘path_join(“test”, “sub_test”, “sub_sub_test”)’, globals=globals()) os_secs = timeit.timeit(‘os.path.join(“test”, “sub_test”, “sub_sub_test”)’, globals=globals()) assert paths_secs < os_secs

even if you import `os.path.join()` without the namespace prefixes, like this:: .. code-block:: python

from os.path import join as os_join os_secs = timeit.timeit(‘os_join(“test”, “sub_test”, “sub_sub_test”)’, globals=globals()) assert paths_secs < os_secs

---

_path_match_replacement = re.compile('(//|**/|*|\\\\\\?|\\\\\\[*\\\\\\]|\\\\\\[\\\\\\?\\\\\\]|\\\\\\[!|\\\\\\[|\\\\\\])')**

pre-compiled regular expression for *path_match()*, inspired by the great SO answer of Pugsley (see https://stackoverflow.com/questions/27726545/63212852#63212852)

path_match(*path*, *mask*)

return True if the specified path matches the specified path mask/pattern.

Parameters

- **path** (str) – path string to match.
- **mask** (str) – path mask/pattern including glob-like wildcards.

Return type
bool

Returns
True if the path specified by `path` matches the mask/pattern specified by the `mask` argument.

path_name(*path*)

determine placeholder key name of the specified path.

Parameters
path (str) – path string to determine name of (can contain placeholders).

Return type
str

Returns
name (respectively dict key in *PATH_PLACEHOLDERS*) of the found path or empty string if not found.

paths_match(*paths*, *masks*)

filter the paths matching at least one of the specified glob-like wildcard masks.

Parameters

- **paths** (Sequence[str]) – sequence of path strings to be checked if they match at least one pattern/mask, specified by the `masks` argument.
- **masks** (Sequence[str]) – sequence of path masks/pattern with glob-like wildcards.

Return type
Iterable[str]

Returns
list of the paths specified by `paths` that match at least one mask, specified by the `masks` argument.

placeholder_key(*path*)

determine *PATH_PLACEHOLDERS* key of specified path.

Parameters
path¶ (str) – path string starting with a *PATH_PLACEHOLDERS* path prefix.

Return type
str

Returns
placeholder key (if found as path prefix), else empty string.

placeholder_path(*path*)

replace begin of path string with the longest prefix found in *PATH_PLACEHOLDERS*.

Parameters
path¶ (str) – path string (optionally including sub-folders and file name).

Return type
str

Returns
path string with replaced placeholder prefix (if found).

series_file_name(*file_path*, *digits=2*, *marker=' '*, *create=False*)

determine non-existent series file name with a unique series index.

Parameters

- **file_path**¶ (str) – file path and name (optional with extension).
- **digits**¶ (int) – number of digits used for the series index.
- **marker**¶ (str) – marker that will be put at the end of the file name and before the series index.
- **create**¶ (bool) – pass True to create the file (to reserve the series index).

Return type
str

Returns
file path extended with unique/new series index.

skip_py_cache_files(*file_path*)

file exclude callback for the files under Python's cache folders.

Parameters
file_path¶ (str) – path to file to check for exclusion, relative to the project root folder.

Return type
bool

Returns
True if the file specified in `file_path` has to excluded, else False.

user_data_path()

determine the os-specific absolute path of the {usr} directory where user data can be stored.

Hint: this path is not accessible on Android devices, use *user_docs_path()* instead to get a more public path to the user.

Return type
str

Returns
path string of the user data folder.

`user_docs_path()`

determine the os-specific absolute path of the {doc} directory where the user is storing the personal documents.

Hint: use *user_data_path()* instead to get a more hidden user data.

Return type
str

Returns
path string of the user documents folder.

`PATH_PLACEHOLDERS = {'ado': '/home/docs/Documents/docs', 'app': '/home/docs/.config/docs', 'app_name': 'docs', 'cwd': '/home/docs/checkouts/readthedocs.org/user_builds/ae/checkouts/stable/docs', 'doc': '/home/docs/Documents', 'usr': '/home/docs/.config'}`

placeholders of user-, os- and app-specific system paths and file name parts

`class Collector(item_collector=<function coll_files>, **placeholders)`

Bases: object

file/folder collector class

`__init__(item_collector=<function coll_files>, **placeholders)`

create new file/folder/item collector instance with individual (extended or overriding) placeholders.

Parameters

- **item_collector** (Callable[[str], Iterable[tuple[Optional[str], Any]]]) – callable to determine the item type to collect. the default is the *coll_files()* function. pass e.g. *coll_folders()* to collect only folders or *coll_items()* to collect both (files and folders). overload the arguments of these functions (with partial) to adapt/change their default arguments.
- **placeholders** – all other kwargs are placeholders with their names:replacements as keys:values. the placeholders provided by *PATH_PLACEHOLDERS* are available too (but will be overwritten by these arguments).

`paths: List[Any]`

list of found/collected folders

`files: List[Any]`

list of found/collected files

`selected: List[Any]`

list of found/collected file/folder item instances

`failed`

number of not found select-combinations

`prefix_failed: Dict[str, int]`

not found select-combinations count for each prefix

suffix_failed: Dict[str, int]

not found select-combinations count for each suffix

placeholders

path part placeholders of this Collector instance

check_add(*item_mask*, *select=False*)

check if item mask match file/folder(s) and if yes append accordingly to collecting instance lists.

Parameters

- **item_mask** (str) – file/folder mask, optionally including wildcards in the glob.glob format.
- **select** (bool) – pass True to additionally add found files/folders into *selected*.

Return type
bool

Returns
True if at least one file/folder got found/added, else False.

_collect_appends(*prefix*, *appends*, *only_first_of*)

_collect_selects(*prefix*, *selects*, *only_first_of*)

collect(**prefixes*, *append=()*, *select=()*, *only_first_of=('append', 'prefix', 'select')*)

collect additional files/folders by combining the given prefixes with all the given append/select suffixes.

Note: all arguments of this method can either be passed either as tuples, or for a single value as string.

Parameters

- **prefixes** (str) – tuple of file/folder paths to be used as prefix.
- **append** (Union[str, Tuple[str, ...]]) – tuple of file/folder names to be used as append suffix.
- **select** (Union[str, Tuple[str, ...]]) – tuple of file/folder names to be used as select suffix. in contrary to append is logging not found prefixes combinations in the instance attributes *failed*, *prefix_failed* and *suffix_failed*.
- **only_first_of** (Union[str, Tuple[str, ...]]) – tuple with the strings *'prefix'*, *'append'* or *'select'* or one of these strings. if it contains the string *'prefix'* then only the files/folders of the first combination will be collected. if it contains *'append'* then only the files/folders of the first combination done with the suffixes passed into the *append* argument will be collected. if it contains *'select'* then only the files/folders of the first combination done with the suffixes passed into the *select* argument will be collected. pass empty tuple to collect all combinations.

Return type
Collector

each of the passed *prefixes* will be combined with the suffixes specified in *append* and in *select*. the resulting file/folder paths that are exist, will then be added to the appropriate instance attribute, either *files* for a file or *paths* for a folder.

additionally the existing file/folder paths from the combinations of *prefixes* and *select* will be added in the *selected* list attribute.

Hint: more details and some examples are available in the doc string of this *module*.

property error_message: str

returns an error message if an error occurred.

Returns

error message string if collection failure/error occurred, else an empty string.

class FilesRegister(**add_path_args*, *property_matcher=None*, *file_sorter=None*, ***add_path_kwargs*)

Bases: dict

file register catalog - see also *files register* examples.

__init__(**add_path_args*, *property_matcher=None*, *file_sorter=None*, ***add_path_kwargs*)

create files register instance.

this method gets redirected with *add_path_args* and *add_path_kwargs* arguments to *add_paths()*.

Parameters

- **add_path_args** – if passed then *add_paths()* will be called with this args tuple.
- **property_matcher** (Optional[Callable[[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]], bool]]) – property matcher callable, used as default value by *find_file()* if not passed there.
- **file_sorter** (Optional[Callable[[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]], Any]]) – file sorter callable, used as default value by *find_file()* if not passed there.
- **add_path_kwargs** – passed onto call of *add_paths()* if the *add_path_args* got provided by the caller.

__call__(**find_args*, ***find_kwargs*)

add_path_args and kwargs will be completely redirected to *find_file()*.

Return type

Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any, None]

add_file(*file_obj*, *first_index=9223372036854775807*)

add a single file to the list of this dict mapped by the file-name/stem as dict key.

Parameters

- **file_obj** (Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]) – either file path string or any object with a *stem* attribute.
- **first_index** (int) – pass list index -n-1..n-1 to insert *file_obj* in the name's list. values greater than n (==len(file_list)) will append the file_obj to the end of the file object list and values less than n-1 will insert the file_obj to the start of the file.

add_files(*files*, *first_index=9223372036854775807*)

add files from another *FilesRegister* instance.

Parameters

- **files** (Iterable[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]]) – iterable with file objects to be added.

- **first_index** (int) – pass list index -n-1..n-1 to insert the first file_obj in each name's register list. values greater than n (==len(file_list)) will append the file_obj to the end of the file object list. the order of the added items will be unchanged if this value is greater or equal to zero. negative values will add the items from *files* in reversed order and **after** the item specified by this index value (so passing -1 will append the items to the end in reversed order, while passing -(n+1) will insert them at the beginning in reversed order).

Return type
List[str]

Returns
list of paths of the added files.

add_paths(**file_path_masks*, *first_index=9223372036854775807*, *file_class=<class 'ae.files.RegisteredFile'>*, ***init_kwargs*)

add files found in the folder(s) specified by the *file_path_masks* args.

Parameters

- **file_path_masks** (str) – file path masks (with optional wildcards and *PATH_PLACEHOLDERS*) specifying the files to collect (by default including the sub-folders).
- **first_index** (int) – pass list index -n-1..n-1 to insert the first file_obj in each name's register list. values greater than n (==len(file_list)) will append the file_obj to the end of the file object list. the order of the added items will be unchanged if this value is greater or equal to zero. negative values will add the found items in reversed order and **after** the item specified by this index value (so passing -1 will append the items to the end in reversed order, while passing -(n+1) will insert them at the beginning in reversed order).
- **file_class** (Type[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]]) – the used file object class (see `FileObject`). each found file object will be passed to the class constructor (callable) and added to the list which is an item of this dict.
- **init_kwargs** – additional/optional kwargs passed onto the used *file_class*. pass e.g. the object_loader to use, if *file_class* is `CachedFile` (instead of the default: `RegisteredFile`).

Return type
List[str]

Returns
list of paths of the added files.

add_register(*files_register*, *first_index=9223372036854775807*)

add files from another *FilesRegister* instance.

Parameters

- **files_register** (*FilesRegister*) – files register instance containing the file_obj to be added.
- **first_index** (int) – pass list index -n-1..n-1 to insert the first file_obj in each name's register list. values greater than n (==len(file_list)) will append the file_obj to the end of the file object list. the order of the added items will be unchanged if this value is greater or equal to zero. negative values will add the found items in reversed order and **after** the item specified by this index value (so passing -1 will append the items to the end in reversed order, while passing -(n+1) will insert them at the beginning in reversed order).

Return type
List[str]

Returns
list of paths of the added files.

find_file(*name*, *properties=None*, *property_matcher=None*, *file_sorter=None*)

find file_obj in this register via properties, property matcher callables and/or file sorter.

Parameters

- **name** (str) – file name (stem without extension) to find.
- **properties** (Optional[Dict[str, Union[int, float, str]]]) – properties to select the correct file.
- **property_matcher** (Optional[Callable[[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]], bool]]) – callable to match the correct file.
- **file_sorter** (Optional[Callable[[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]], Any]]) – callable to sort resulting match results.

Return type
Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any, None]

Returns
registered/cached file object of the first found/correct file.

reclassify(*file_class=<class 'ae.files.CachedFile'>*, ***init_kwargs*)

re-instantiate all name's file registers items to instances of the class *file_class*.

Parameters

- **file_class** (Type[Union[str, *RegisteredFile*, *CachedFile*, Path, PurePath, Any]]) – the new file object class (see *FileObject*). each found file object will be passed to the class constructor (callable) and the return value will then replace the file object in the file list.
- **init_kwargs** – additional/optional kwargs passed onto the used file_class. pass e.g. the object_loader to use, if *file_class* is CachedFile (the default file object class).

4.9 ae.core

4.9.1 application core constants, helper functions and base classes

this module declares practical constants, tiny helper functions and app base classes, which are reducing the code of your application (and of other ae namespace modules/portions).

core constants

there are three debug level constants: *DEBUG_LEVEL_DISABLED*, *DEBUG_LEVEL_ENABLED* and *DEBUG_LEVEL_VERBOSE*. short names for all debug level constants are provided by the dict *DEBUG_LEVELS*. the debug level of your application can be either set in your code or optionally data-driven externally (using the *config files* or *config options* of the module *ae.console*).

to use the python logging module in conjunction with this module the constant *LOGGING_LEVELS* is providing a mapping between the debug levels and the python logging levels.

the encoding of strings into byte-strings (to output them to the console/stdout or to file contents) can be tricky sometimes. to not lose any logging output because of invalid characters this module will automatically handle any

UnicodeEncodeError exception for you. invalid characters will then automatically be converted to the default encoding (specified by *DEF_ENCODING*) with the default error handling method specified by *DEF_ENCODE_ERRORS* (both defined in the *ae.base* namespace portion/module).

core helper functions

the *print_out()* function, which is fully compatible to pythons print(), is using the encode helpers *force_encoding()* and *to_ascii()* to auto-correct invalid characters.

the function *hide_dup_line_prefix()* is very practical if you want to remove or hide redundant line prefixes in your log files, to make them better readable.

application base classes

the classes *AppBase* and *SubApp* are applying logging and debugging features to your application. create in your application one instance of *AppBase* to represent the main application task. if your application needs a separate logging/debugging configuration for sub-threads or sub-tasks then create an *SubApp* instance for each of these sub-apps.

sub-apps are very flexible and not tied to any fix use-case. they can be created e.g. for each sub-task or application thread. you could also create a *SubApp* instance for each of your external systems, like a database server or to connect your application onto different test environments or to your live/production system (e.g. for system comparison and maintenance).

both application classes are automatically catching and handling any exceptions and run-time errors: only if any critical exception/error cannot be handled then the *shutdown()* method will make sure that all sub-apps and threads get terminated and joined. additionally all print-out buffers will be flushed to include all the info of the critical error (the last debug and error messages) into the standard error/output and into any activated log files.

basic usage of an application base class

at the top of your python application main file/module create an instance of the class *AppBase*:

```
""" docstring at the top of the main module of your application """
from ae.core import AppBase

__version__ = '1.2.3'

ca = AppBase()
```

in the above example the *AppBase* instance will automatically use the docstring title of the module as application title and the string in the module variable __version___ as application version. to overwrite these defaults pass your application title and version string via the arguments *app_title* and *app_version* to the instantiation of *AppBase*:

```
ca = AppBase(app_title="title of this app instance", app_version='3.2.1')
```

other automatically initialized instance attributes of *AppBase* are documented underneath in the *class docstring*. they include e.g. the *date and time when the instance got created*, the *name/id of this application instance* or the *application path*.

application class hierarchy

for most use cases you will not instantiate from *AppBase* directly - instead you will instantiate one of the extended application classes that are inherited from this base class.

the class *ConsoleApp* e.g. inherits from *AppBase* and is adding configuration options and variables to it. so in your console application it is recommended to directly use instances of *ConsoleApp* instead of *AppBase*.

for applications with an GUI use instead one of the classes *KivyMainApp*, *EnamlMainApp* or `TogaMainApp`.

application logging

print-outs are an essential tool for the debugging and logging of your application at run-time. in python the print-outs are done with the `print()` function or with the python `logging` module. these print-outs get per default send to the standard output and error streams of your OS and so displayed on your system console/shell. the *print_out()* function and the *print_out()* method of this *core* module are adding two more sophisticated ways for print-outs to the console/log-files.

using *AppBase* is making the logging much easier and also ensures that print-outs of any imported library or package will be included within your log files. this is done by redirecting the standard output and error streams to your log files with the help of the *_PrintingReplicator* class.

head-less server applications like web servers are mostly not allowed to use the standard output streams. for some these applications you could redirect the standard output and error streams to a log file by using the OS redirection character (>):

```
python your_application.py >log_std_out.log 2>log_std_err.log
```

but because most web servers doesn't allow you to use this redirection, you can alternatively specify the *suppress_stdout* parameter as `True` in the instantiation of an *AppBase* instance. additionally you can call the *init_logging()* method to activate a log file. after that all print-outs of your application and libraries will only appear in your log file.

also in complex applications, where huge print-outs to the console can get lost easily, you want to use a log file instead. but even a single log file can get messy to read, especially for multithreading server applications. for that *SubApp* is allowing you to create for each thread a separate sub-app instance with its own log file.

using this module ensures that any crashes or freezes happening in your application will be fully logged. apart from the gracefully handling of `UnicodeEncodeError` exceptions, the `Python faulthandler` will be enabled automatically to catch system errors and to dump a traceback of them to the console and any activated log file.

activate ae log file

ae log files are text files using by default the encoding of your OS console/shell. to activate the redirection of your applications print-outs into an ae log file for a *AppBase* instance you simply specify the file name of the log file in the *init_logging()* method call:

```
app = AppBase()
app.init_logging(log_file_name='my_log_file.log')
```

activate ae logging features

for multi-threaded applications include the thread-id of the printing thread automatically into your log files by passing a `True` value to the *multi_threading* argument. to additionally also suppress any print-outs to the standard output/error streams pass `True` to the *suppress_stdout* argument:

```
app = AppBase(multi_threading=True, suppress_stdout=True)
app.init_logging(log_file_name='my_log_file.log')
```

the ae log files provided by this module are automatically rotating if the size of a log file succeeds the value in MBytes defined in the *LOG_FILE_MAX_SIZE*. to adapt this value to your needs you can specify the maximum log file size in MBytes with the argument *log_file_size_max* in your call of *init_logging()*:

```
app.init_logging(log_file_name='my_log_file.log', log_file_size_max=9.)
```

by using the *ConsoleApp* class instead of *AppBase* you can alternatively store the logging configuration of your application within a *configuration variable* or a *configuration option*. the order of precedence to find the appropriate logging configuration of each app instance is documented *here*.

using python logging module

if you prefer to use instead the python logging module for the print-outs of your application, then pass a `python logging configuration dictionary` with the individual configuration of your logging handlers, files and loggers to the *py_logging_params* argument of the *init_logging()* method:

```
app.init_logging(py_logging_params=my_py_logging_config_dict)
```

passing the python logging configuration dictionary to one of the *AppBase* instances created by your application will automatically disable the ae log file of this instance.

application debugging

to use the debug features of *core* you simply have to import the needed *debug level constant* to pass it at instantiation of your *AppBase* or *SubApp* class to the *debug_level* argument:

```
app = AppBase(..., debug_level= :data:`DEBUG_LEVEL_ENABLED`)     # same for␣
↪:class:`SubApp`
```

by passing *DEBUG_LEVEL_ENABLED* the print-outs (and log file contents) will be more detailed, and even more verbose if you use instead the debug level *DEBUG_LEVEL_VERBOSE*.

the debug level can be changed at any time in your application code by directly assigning the new debug level to the *debug_level* property. if you prefer to change the debug levels dynamically, then use the *ConsoleApp* instead of *AppBase*, because *ConsoleApp* provides this property as a *configuration file variable* and *commend line option*. this way you can specify *the actual debug level* without the need to change (and re-build) your application code.

Module Attributes

DEBUG_LEVEL_DISABLED	lowest debug level - only display logging levels ERROR/CRITICAL.
DEBUG_LEVEL_ENABLED	minimum debugging info - display logging levels WARNING or higher.
DEBUG_LEVEL_VERBOSE	verbose debug info - display logging levels INFO/DEBUG or higher.
DEBUG_LEVELS	numeric ids and names of all supported debug levels.
LOGGING_LEVELS	association between ae debug levels and python logging levels.
HIDDEN_CREDENTIALS	credential keys that are hidden in print/repr output (not if verbose)
MAX_NUM_LOG_FILES	maximum number of *ae log files*
LOG_FILE_MAX_SIZE	max.
LOG_FILE_IDX_WIDTH	width of rotating log file index within log file name; adding +3 to ensure index range up to factor 10^3.
ori_std_out	original sys.stdout on app startup
ori_std_err	original sys.stderr on app startup
log_file_lock	log file rotation multi-threading lock
po(*objects[, sep, end, file, flush, ...])	alias of function *print_out()*
APP_KEY_SEP	separator character used in *app_key* of *AppBase* instance
app_inst_lock	app instantiation multi-threading lock

Functions

activate_multi_threading()	activate multi-threading for all app instances (normally done at main app startup).
hide_dup_line_prefix(last_line, current_line)	replace duplicate characters at the start of two strings with spaces.
logger_late_init()	check if logging modules got initialized already and if not then do it now.
main_app_instance()	determine the main instance of the *AppBase* in the current running application.
po(*objects[, sep, end, file, flush, ...])	alias of function *print_out()*
print_out(*objects[, sep, end, file, flush, ...])	universal/unbreakable print function - replacement for the built-in python function print().
registered_app_names()	determine the app names of all registered/running applications.

Classes

AppBase([app_title, app_name, app_version, ...])	provides easy logging and debugging for your application.
SubApp([app_title, app_name, app_version, ...])	separate/additional sub-app/thread/task with own/individual logging/debug configuration.

DEBUG_LEVEL_DISABLED: int = 0

lowest debug level - only display logging levels ERROR/CRITICAL.

DEBUG_LEVEL_ENABLED: int = 1

minimum debugging info - display logging levels WARNING or higher.

DEBUG_LEVEL_VERBOSE: int = 2

verbose debug info - display logging levels INFO/DEBUG or higher.

DEBUG_LEVELS: Dict[int, str] = {0: 'disabled', 1: 'enabled', 2: 'verbose'}

numeric ids and names of all supported debug levels.

LOGGING_LEVELS: Dict[int, int] = {0: 30, 1: 20, 2: 10}

association between ae debug levels and python logging levels.

HIDDEN_CREDENTIALS = ('password', 'token')

credential keys that are hidden in print/repr output (not if verbose)

hide_dup_line_prefix(*last_line*, *current_line*)

replace duplicate characters at the start of two strings with spaces.

Parameters

- **last_line** (str) – last line string (e.g. the last line of text/log file).
- **current_line** (str) – current line string.

Return type
str

Returns
current line string but duplicate characters at the beginning are replaced by space chars.

MAX_NUM_LOG_FILES: int = 69

maximum number of *ae log files*

LOG_FILE_MAX_SIZE: int = 15

max. size in MB of rotating *ae log files*

LOG_FILE_IDX_WIDTH: int = 5

width of rotating log file index within log file name; adding +3 to ensure index range up to factor 10^3.

ori_std_out: TextIO = <_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>

original sys.stdout on app startup

ori_std_err: TextIO = <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>

original sys.stderr on app startup

log_file_lock: RLock = <unlocked _thread.RLock object owner=0 count=0>

log file rotation multi-threading lock

_LOGGER = None

python logger for this module gets lazy/late initialized and only if requested by caller

logger_late_init()

check if logging modules got initialized already and if not then do it now.

_MULTI_THREADING_ACTIVATED: bool = False

flag if threading is used in your application

activate_multi_threading()

activate multi-threading for all app instances (normally done at main app startup).

_deactivate_multi_threading()

disable multi threading (needed to reset app environment in unit testing).

print_out(**objects*, *sep=' '*, *end='\\n'*, *file=None*, *flush=False*, *encode_errors_def='backslashreplace'*, *logger=None*, *app=None*, ***kwargs*)

universal/unbreakable print function - replacement for the built-in python function print().

Parameters

- **objects** – tuple of objects to be printed. if the first object is a string that starts with a \r character then the print-out will be only sent to the standard output (and will not be added to any active log files - see also *end* argument).
- **sep** (str) – separator character between each printed object/string (def=" ").
- **end** (str) – finalizing character added to the end of this print-out (def="\n"). pass \r to suppress the print-out into *ae log file* or to any activated python logger - useful for console/shell processing animation (see tcp.TcpServer.run()).
- **file** (Optional[TextIO]) – output stream object to be printed to (def=None which will use standard output streams). if given then the redirection to all active log files and python logging loggers will be disabled (even if the *logger* argument is specified).
- **flush** (bool) – flush stream after printing (def=False).
- **encode_errors_def** (str) – default error handling to encode (def=:data:*DEF_ENCODE_ERRORS*).
- **logger** (Optional[Logger]) – used logger to output *objects* (def=None). ignored if the *file* argument gets specified/passed.
- **app** (Optional[*AppBase*]) – the app instance from where this print-out got initiated.
- **kwargs** – catch unsupported kwargs for debugging (all items will be printed to all the activated logging/output streams).

this function is silently handling and autocorrecting string encode errors for output/log streams which are not supporting unicode. any instance of *AppBase* is providing this function as a method with the *same name*). it is recommended to call/use this instance method instead of this function.

in multithreading applications this function prevents dismembered/fluttered print-outs from different threads.

Note: this function has an alias named *po()*.

po(**objects*, *sep=' '*, *end='\\n'*, *file=None*, *flush=False*, *encode_errors_def='backslashreplace'*, *logger=None*, *app=None*, ***kwargs*)

alias of function *print_out()*

APP_KEY_SEP: str = '@'

separator character used in *app_key* of *AppBase* instance

_APP_INSTANCES: WeakValueDictionary[str, AppBase] = <WeakValueDictionary>

dict that is weakly holding references to all *AppBase* instances created at run time.

gets automatically initialized in *AppBase.__init__()* to allow log file split/rotation and debug_level access at application thread or module level.

the first created *AppBase* instance is called the main app instance. *_MAIN_APP_INST_KEY* stores the dict key of the main instance.

_MAIN_APP_INST_KEY: str = ''

key in *_APP_INSTANCES* of main *AppBase* instance

app_inst_lock: RLock = <unlocked _thread.RLock object owner=0 count=0>

app instantiation multi-threading lock

main_app_instance()

determine the main instance of the *AppBase* in the current running application.

Return type
Optional[*AppBase*]

Returns
main/first-instantiated *AppBase* instance or None (if app is not fully initialized yet).

registered_app_names()

determine the app names of all registered/running applications.

Return type
List[str]

_register_app_instance(*app*)

register new *AppBase* instance in *_APP_INSTANCES*.

Parameters
app (*AppBase*) – *AppBase* instance to register

_unregister_app_instance(*app_key*)

unregister/remove *AppBase* instance from within *_APP_INSTANCES*.

Parameters
app_key (str) – app key of the instance to remove.

Return type
Optional[*AppBase*]

Returns
removed *AppBase* instance.

_shut_down_sub_app_instances(*timeout=None*)

shut down all *SubApp* instances.

Parameters
timeout (Optional[float]) – timeout float value in seconds used for the *SubApp* shutdowns and for the acquisition of the threading locks of *the ae log file* and the *app instances*.

class _PrintingReplicator(*sys_out_obj=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*)

Bases: object

replacement of standard/error stream replicating print-outs to all active logging streams (log files/buffers).

__init__(*sys_out_obj=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='utf-8'>*)

initialise a new T-stream-object

Parameters
sys_out_obj (TextIO) – standard output/error stream to be replicated (def=sys.stdout)

write(*any_str*)

write string to ae logging and standard output streams.

automatically suppressing UnicodeEncodeErrors if console/shell or log file has different encoding by forcing re-encoding with DEF_ENCODE_ERRORS.

Parameters
any_str (Union[str, bytes]) – string or bytes to output.

Return type
None

__getattr__(*attr*)

get attribute value from standard output stream.

Parameters
attr (str) – name of the attribute to retrieve/return.

Return type
Any

Returns
value of the attribute.

_APP_THREADS: WeakValueDictionary[int, Thread] = <WeakValueDictionary>

weak dict to keep the references of all application threads. added to prevent the joining of unit testing threads in the test teardown (resetting app environment).

_register_app_thread()

add new app thread to _APP_THREADS if not already added.

_join_app_threads(*timeout=None*)

join/finish all app threads and finally deactivate multi-threading.

Parameters
timeout (Optional[float]) – timeout float value in seconds for thread joining (def=None - block/no-timeout).

Note: this function has to be called by the main app instance only.

---

class AppBase(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*)

Bases: object

provides easy logging and debugging for your application.

most applications only need a single instance of this class; apps with threads could create separate instances for each thread.

instance Attributes (ordered alphabetically - ignoring underscore characters):

- *app_key* id/key of this application instance.
- *app_name* basename (without the file name extension) of the executable.

- *app_path* file path of app executable.
- *app_title* application title/description.
- *app_version* application version (set via the *AppBase.app_version* argument).
- *debug_level* debug level of this instance.
- *_last_log_line_prefix* last ae log file line prefix that got print-out to the log of this app instance.
- *_log_buf_stream* ae log file buffer stream.
- *_log_file_index* index of the current rotation ae log file backup.
- *_log_file_name* path and file name of the ae log file.
- *_log_file_size_max* maximum size in MBytes of an ae log file.
- *_log_file_stream* ae log file TextIO output stream.
- *_log_with_timestamp* log timestamp line prefix if True or a non-empty strftime compatible format string.
- *py_log_params* python logging config dictionary.
- *_nul_std_out* null stream used to prevent print-outs to `standard output`.
- *_shut_down* flag set to True if this application instance got already shutdown.
- *startup_beg* datetime of begin of the instantiation/startup of this app instance.
- *startup_end* datetime of end of the instantiation/startup of this application instance.
- *suppress_stdout* flag set to True if this application does not print to stdout/console.
- *sys_env_id* system environment id of this application instance.

_last_log_line_prefix: `str = ''`
prefix of the last printed log line

_log_buf_stream: `Optional[StringIO] = None`
log file buffer stream instance

_log_file_stream: `Optional[TextIO] = None`
log file stream instance

_log_file_index: `int = 0`
log file index (for rotating logs)

_log_file_size_max: `float = 15`
maximum log file size in MBytes (rotating log files)

_log_file_name: `str = ''`
log file name

_log_with_timestamp: `Union[bool, str] = False`
True of strftime format string to enable timestamp

_nul_std_out: `Optional[TextIO] = None`
logging null stream

py_log_params: `Dict[str, Any] = {}`
dict of config parameters for py logging

_shut_down: `bool = False`
True if this app instance got shut down already

__init__(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*)

initialize a new *AppBase* instance.

Parameters

- **app_title** (str) – application title/description setting the attribute *app_title*. if not specified then the docstring of your app's main module will be used (see *example*).
- **app_name** (str) – application instance name to set the attribute *app_name*. if not specified then base name of the main module file name will be used.
- **app_version** (str) – application version string to set the attribute *app_version*. if not specified then value of a global variable with the name *__version__* will be used (if declared in the actual call stack).
- **sys_env_id** (str) – system environment id to set the instance attribute *sys_env_id*. the default value of this argument is an empty string.
- **debug_level** (int) – default debug level to set the instance attribute *debug_level*. the default value of this argument is *DEBUG_LEVEL_DISABLED*.
- **multi_threading** (bool) – pass True if instance is used in multi-threading app.
- **suppress_stdout** (bool) – pass True (for wsgi apps) to prevent any python print outputs to stdout.

startup_beg: datetime

begin of app startup datetime

app_path: str

path to folder of your main app code file

app_title: str = ''

title/description of this app instance

app_name: str = ''

name of this app instance

app_version: str = ''

version of this app instance

_debug_level: int = 2

debug level of this app instance

sys_env_id: str = ''

system environment id of this app instance

suppress_stdout: bool = True

flag to suppress prints to stdout

startup_end: Optional[datetime] = None

end datetime of the application startup

__del__()

deallocate this app instance by calling *AppBase.shutdown()*.

property **active_log_stream:** StringIO | TextIO | None

check if ae logging is active and if yes then return the currently used log stream (read-only property).

Returns
log file or buf stream if logging is activated, else None.

property app_key: str

determine the key of this application class instance (read-only property).

Returns
application key string.

property debug_level: int

debug level property:

Getter
return the current debug level of this app instance.

Setter
change the debug level of this app instance.

property debug: bool

True if app is in debug mode.

property verbose: bool

True if app is in verbose debug mode.

call_method(*callback*, **args*, ***kwargs*)

call passed callable/method with the passed args, catching and logging exceptions preventing app exit.

Parameters

- **callback** (Union[Callable, str]) – either a callable or the name of the main app method of this instance to call.
- **args** – args passed to the main app method to be called.
- **kwargs** – kwargs passed to the main app method to be called.

Return type
Any

Returns
return value of the called method or None if method throws exception/does not exist.

init_logging(*py_logging_params=None*, *log_file_name=''*, *log_file_size_max=15*, *log_with_timestamp=False*, *disable_buffering=False*)

initialize logging system.

Parameters

- **py_logging_params** (Optional[Dict[str, Any]]) – config dict for python logging configuration. if this dict is not empty then python logging is configured with the given options in this dict and all the other kwargs are ignored.
- **log_file_name** (str) – default log file name for ae logging (def=” - ae logging disabled).
- **log_file_size_max** (float) – max. size in MB of ae log file (def=LOG_FILE_MAX_SIZE).
- **log_with_timestamp** (Union[bool, str]) – add timestamp prefix to each log line if True or a non-empty strftime compatible format string.
- **disable_buffering** (bool) – pass True to disable ae log buffering at app startup.

log files and config values will be initialized as late as possible in *log_file_check()*, e.g. indirectly triggered by a request to a config variable via `_parse_args()` (like *logFile*).

log_line_prefix()

compile prefix of log print-out line for this *AppBase* instance.

the line prefix consists of (depending on the individual values of either a module variable or of an attribute this app instance):

- *_MULTI_THREADING_ACTIVATED*: if True then the thread id gets printed surrounded with angle brackets (< and >), right aligned and space padded to minimal 6 characters.
- *sys_env_id*: if not empty then printed surrounded with curly brackets ({ and }), left aligned and space padded to minimal 4 characters.
- *_log_with_timestamp*: if (a) True or (b) a non-empty string then the system time (determined with `now()`) gets printed in the format specified either by the (a) the *DATE_TIME_ISO* constant or (b) by the string in this attribute.

this method is using the instance attribute *_last_log_line_prefix* to keep a copy of the last printed log line prefix to prevent the printout of duplicate characters in consecutive log lines.

Return type
`str`

Returns
log file line prefix string including one space as separator character at the end.

log_file_check(*curr_stream=None*)

check and possibly correct log file status and the passed currently used stream.

Parameters
curr_stream¶ (`Optional`[`TextIO`]) – currently used stream.

Return type
`Optional`[`TextIO`]

Returns
stream passed into *curr_stream* or new/redirected stream of *curr_stream* or None if *curr_stream* is None.

for already opened log files check if the ae log file is big enough and if yes then do a file rotation. if log file is not opened but log file name got already set, then check if log startup buffer is active and if yes then create log file, pass log buffer content to log file and close the log buffer.

print_out(**objects*, *file=None*, ***kwargs*)

app-instance-specific print-outs.

Parameters

- **objects**¶ – objects to be printed out.
- **file**¶ (`Optional`[`TextIO`]) – output stream object to be printed to (def=None). passing None on a main app instance will print the objects to the standard output and any active log files, but on a *SubApp* instance with an active log file the print-out will get redirected exclusively/only to log file of this *SubApp* instance.
- **kwargs**¶ – all the other supported kwargs of this method are documented *at the print_out() function of this module*.

this method has an alias named *po()*

po(**objects*, *file=None*, ***kwargs*)

alias of method *print_out()*

debug_out(**objects*, *minimum_debug_level=1*, ***kwargs*)

special debug version of builtin print() function.

this method will print out the passed objects only if the *current debug level* of this app instance is higher than the value passed into the *minimum_debug_level* argument. in this case the print-out will be delegated onto the *print_out()*.

Parameters

- **objects** – objects to be printed out.
- **minimum_debug_level** (int) – minimum debug level to print the passed objects.
- **kwargs** – all the supported kwargs of this method are documented at the print_out() function of the *core* module (including the *file* argument).

this method has an alias named *dpo()*.

dpo(**objects*, *minimum_debug_level=1*, ***kwargs*)

alias of method *debug_out()*

verbose_out(**objects*, ***kwargs*)

special verbose debug version of builtin print() function.

Parameters

- **objects** – objects to be printed out.
- **kwargs** – the *file* argument is documented at the *print_out()* method of the *AppBase* class. all other supported kwargs of this method are documented at the print_out() function of the *core* module.

this method has an alias named *vpo()*.

vpo(**objects*, ***kwargs*)

alias of method *verbose_out()*

shutdown(*exit_code=0*, *timeout=None*)

shutdown this app instance and if it is the main app instance then also any created sub-app-instances.

Parameters

- **exit_code** (Optional[int]) – set application OS exit code - ignored if this is NOT the main app instance (def=0). pass None to prevent call of sys.exit(exit_code).
- **timeout** (Optional[float]) – timeout float value in seconds used for the thread termination/joining, for the *SubApp* shutdowns and for the acquisition of the threading locks of *the ae log file* and the *app instances*.

_std_out_err_redirection(*redirect*)

enable/disable the redirection of the standard output/error TextIO streams if needed.

Parameters

redirect (bool) – pass True to enable or False to disable the redirection.

_append_eof_and_flush_file(*stream_file*, *stream_name*)

add special end-of-file marker and flush the internal buffers to the file stream.

Parameters

- **stream_file** (TextIO) – file stream.

- **stream_name** (str) – name of the file stream (only used for debugging/error messages).

_flush_and_close_log_buf()

flush and close ae log buffer and pass content to log stream if opened.

_open_log_file()

open the ae log file with path and file name specified by *_log_file_name*.

tries to create a log sub-folder - if specified in *_log_file_name* and the folder does not exist (folder creation is limited to one folder level).

Note: a already existing file with the same file name will be overwritten (file contents get lost!).

_close_log_file()

close the ae log file.

_rename_log_file()

rename rotating log file while keeping first/startup log and log file count below MAX_NUM_LOG_FILE.

class SubApp(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*)

Bases: *AppBase*

separate/additional sub-app/thread/task with own/individual logging/debug configuration.

create an instance of this class for every extra thread and task where your application needs separate logging and/or debug configuration - additional to the main app instance.

all members of this class are documented at the *AppBase* class.

startup_beg: datetime

begin of app startup datetime

app_path: str

path to folder of your main app code file

4.10 ae.lockname

4.10.1 named threading locks

named locks are used in multi-threaded applications and based on the Python threading lock classes threading.Lock and threading.RLock. the advantage of the named locks in contrary to Python threading locks is that a lock don't need to create and store a reference of a Python threading lock object - the *NamedLocks* does this automatically for your application and does keep track of all the named locks of your application in its class variables.

so a named lock get exclusively identified only by an unique string. and to create other blocking locks you only need a reference to the *NamedLocks* class.

named locks are very useful e.g. if you want to lock a certain record of database table. for this you simply create a new instance of the *NamedLocks* class and as unique string you can use the table name followed by the primary key of the record to lock:

```
named_lock = NamedLocks()
if named_lock.acquire(table_name + primary_key)

    ...     # locked database transaction code goes here

    named_lock.release(table_name + primary_key)
```

if now any other process of your application want to lock the same record (same table name and primary key) then it will be blocked until the process that acquired this named lock first is releasing the table record lock.

alternatively and especially if your application want to create multiple named locks you can use the class *NamedLocks* as a context manager, passing all the named lock strings to the constructor:

```
with NamedLocks(table_name1 + primary_key1, table_name2 + primary_key2, ...):
    ...     # locked database transaction
```

Classes

NamedLocks(*lock_names[, reentrant_locks, ...])	manage all named locks of your application.

class NamedLocks(**lock_names*, *reentrant_locks=True*, *sys_lock=False*)

Bases: object

manage all named locks of your application.

migrated from https://stackoverflow.com/users/355230/martineau answer in stackoverflow on the question https://stackoverflow.com/questions/37624289/value-based-thread-lock.

Note: currently the sys_lock feature is not implemented. use either ae.lockfile or the github extension portalocker (see https://github.com/WoLpH/portalocker) or the encapsulating extension ilock (https://github.com/symonsoft/ilock). more on system wide named locking: https://stackoverflow.com/questions/6931342/system-wide-mutex-in-python-on-linux.

locks_change_lock: ClassVar[allocate_lock] = <unlocked _thread.lock object>

threading lock class variable used to change status of all NamedLock instances

active_locks: ClassVar[Dict[str, Union[allocate_lock, RLock]]] = {}

class variable keeping a dictionary of all active RLock/Lock instances

active_lock_counters: ClassVar[Dict[str, int]] = {}

lock counters class variable for reentrant locks

__init__(**lock_names*, *reentrant_locks=True*, *sys_lock=False*)

prepare new named lock(s).

Parameters

- **lock_names** (str) – unique lock strings to be prepared to be locked by *__enter__()*.
- **reentrant_locks** (bool) – pass False to use non-reentrant locks (True=reentrant locks).
- **sys_lock** (bool) – pass True to prepare system lock (works for several independent applications). CURRENTLY NOT IMPLEMENTED.

_lock_names
tuple of lock names

_lock_class: Type[Union[allocate_lock, RLock]]
used threading lock class

_sys_lock
True if lock will be system-wide (not only application-wide)

_print_func
print function used to show debug and error messages

__enter__()
locking context enter method.

__exit__(*exc_type*, *exc_val*, *exc_tb*)
locking context exit method.

dpo(**args*, ***kwargs*)
print function which is suppressing printout if debug level is too low.

acquire(*lock_name*, **args*, ***kwargs*)
acquire the named lock specified by the *lock_name* argument.

Parameters

- **lock_name** (str) – name of the lock to acquire.
- **args** – passed to the acquire method of the underlying RLock respectively Lock class instance.
- **kwargs** – passed to the acquire method of the underlying RLock or Lock class instance.

Return type
bool

Returns
True if named lock got acquired successfully, else False.

release(*lock_name*)
release the named lock specified by the *lock_name* argument.

Parameters
lock_name (str) – name of the lock to release.

4.11 ae.dynamicod

4.11.1 dynamic execution of code blocks and expressions

this ae namespace portion provides useful helper functions to evaluate Python expressions and execute Python code dynamically at application run-time.

dynamically executed code block or expression string offers convenience for powerful system and application configuration and for data-driven architectures.

for the dynamic execution of functions and code blocks the helper functions *try_call()*, *try_exec()* and *exec_with_return()* are provided. the helper function *try_eval()* evaluates dynamic expressions.

Note: security considerations

make sure that any dynamically executed code is from a secure source to prevent code injections of malware. treat configuration files from untrusted sources with extreme caution and only execute them after a complete check and/or within a sandbox.

Hint: these functions are e.g. used by the *Literal* class to dynamically determine literal values.

Module Attributes

base_globals	default if no global variables get passed in dynamic code/expression evaluations

Functions

exec_with_return(code_block[, ...])	execute python code block and return the resulting value of its last code line.
try_call(callee, *args[, ignored_exceptions])	execute callable while ignoring specified exceptions and return callable return value.
try_eval(expr[, ignored_exceptions, ...])	evaluate expression string ignoring specified exceptions and return evaluated value.
try_exec(code_block[, ignored_exceptions, ...])	execute python code block string ignoring specified exceptions and return value of last code line in block.

exec_with_return(*code_block*, *ignored_exceptions=()*, *glo_vars=None*, *loc_vars=None*)

execute python code block and return the resulting value of its last code line.

Parameters

- **code_block** (str) – python code block to execute.
- **ignored_exceptions** (Tuple[Type[Exception], ...]) – tuple of ignored exceptions.
- **glo_vars** (Optional[Dict[str, Any]]) – optional globals() available in the code execution.
- **loc_vars** (Optional[Dict[str, Any]]) – optional locals() available in the code execution.

Return type
Optional[Any]

Returns
value of the expression at the last code line or UNSET if either code block is empty, only contains comment lines, or one of the ignorable exceptions raised or if last code line is no expression.

inspired by this SO answer https://stackoverflow.com/questions/33409207/how-to-return-value-from-exec-in-function/52361938#52361938.

try_call(*callee*, **args*, *ignored_exceptions=()*, ***kwargs*)

execute callable while ignoring specified exceptions and return callable return value.

Parameters

- **callee** (Callable) – pointer to callable (either function pointer, lambda expression, a class, ...).
- **args** – function arguments tuple.
- **ignored_exceptions** (Tuple[Type[Exception], ...]) – tuple of ignored exceptions.
- **kwargs** – function keyword arguments dict.

Return type
Optional[Any]

Returns
function return value or UNSET if an ignored exception got thrown.

try_eval(*expr*, *ignored_exceptions=()*, *glo_vars=None*, *loc_vars=None*)

evaluate expression string ignoring specified exceptions and return evaluated value.

Parameters

- **expr** (str) – expression to evaluate.
- **ignored_exceptions** (Tuple[Type[Exception], ...]) – tuple of ignored exceptions.
- **glo_vars** (Optional[Dict[str, Any]]) – optional globals() available in the expression evaluation.
- **loc_vars** (Optional[Dict[str, Any]]) – optional locals() available in the expression evaluation.

Return type
Optional[Any]

Returns
function return value or UNSET if an ignored exception got thrown.

try_exec(*code_block*, *ignored_exceptions=()*, *glo_vars=None*, *loc_vars=None*)

execute python code block string ignoring specified exceptions and return value of last code line in block.

Parameters

- **code_block** (str) – python code block to be executed.
- **ignored_exceptions** (Tuple[Type[Exception], ...]) – tuple of ignored exceptions.
- **glo_vars** (Optional[Dict[str, Any]]) – optional globals() available in the code execution.
- **loc_vars** (Optional[Dict[str, Any]]) – optional locals() available in the code execution.

Return type
Optional[Any]

Returns
function return value or UNSET if an ignored exception got thrown.

base_globals

default if no global variables get passed in dynamic code/expression evaluations

4.12 ae.i18n

4.12.1 internationalization / localization helpers

on importing this portion it will automatically determine the default locale (language) and encoding of your operating system and user configuration.

the functions *default_language()* and *default_encoding()* - provided by this portion - are determining or changing the default language and translation texts encoding.

additional languages will be automatically loaded by the function *load_language_texts()*.

translation texts locale paths

multiple paths can be provided by your app as well as any python package and namespace portion to store translation texts. by default they are situated in a sub-folder with the name *loc* underneath of your app/package root folder. to load them call the function *register_package_translations()* from the module using the locale texts.

Hint: see e.g. the ae namespace portion *ae.gui_help* loading package/module specific translation messages.

Hint: the `on_app_build()` app event is used by *gui_help* to load the app-specific translation texts on app startup.

to specify additional locale folders you can use the function *register_translations_path()*.

in a locale folder have to exist at least one sub-folder with a name of the language code for each supported language (e.g. 'loc/en' for english).

in each of these sub-folders at least one message translation file with a file name ending in the string specified by the constant *MSG_FILE_SUFFIX* has to exist.

translatable message texts and f-strings

simple message text strings can be enclosed in the code of your application with the *get_text()* function provided by this portion/module:

```
from ae.i18n import get_text

message = get_text("any translatable message displayed to the app user.")
print(message)          # prints the translated message text
```

for more complex messages with placeholders you can use the *get_f_string()* function:

```
from ae.i18n import get_f_string

my_var = 69
print(get_f_string("The value of my_var is {my_var}."))
```

translatable message can also be provided in various pluralization forms. to get a pluralized message you have to pass the *count* keyword argument of *get_text()*:

```
print(get_text("child", count=1))     # translated into "child" (in english) or e.g.
↪"Kind" in german
print(get_text("child", count=3))     # -> "children" (in english) or e.g. "Kinder" (in␣
↪german)
```

for pluralized message translated by the *get_f_string()* function, the count value have to be passed in the *count* item of the *loc_vars*:

```
print(get_f_string("you have {count] children", loc_vars=dict(count=1)))
# -> "you have 1 child" or e.g. "Sie haben 1 Kind"
print(get_f_string("you have {count] children", loc_vars={'count': 3}))
# -> "you have 3 children" or "Sie haben 3 Kinder"
```

you can load several languages into your app run-time. to get the translation for a language that is not the current default language you have to pass the *language* keyword argument with the desired language code onto the call of *get_text()* (or *get_f_string()*):

```
print(get_text("message", language='es'))   # returns the spanish translation text of
↪"message"
print(get_text("message", language='de'))   # returns the german translation text of
↪"message"
```

the helper function *translation()* can be used to determine if a translation exists for a message text.

Hint: the ae portion *ae.kivy.i18n* is implementing translation messages for kv files and provides additional helper functions and methods.

Module Attributes

MsgType	type of message literals in translation text files
LanguageMessages	type of the data structure storing the loaded messages
DEF_ENCODING	encoding of the messages in your app code
DEF_LANGUAGE	language code of the messages in your app code
LOADED_TRANSLATIONS	message text translations of all loaded languages
MSG_FILE_SUFFIX	name suffix of translation text files
TRANSLATIONS_PATHS	file paths to search for translations
default_locale	language and encoding code of the current language/locale

Functions

default_encoding([new_enc])	get and optionally set the default message text encoding.
default_language([new_lang])	get and optionally set the default language code.
get_f_string(f_str[, key_suffix, language, ...])	translate passed f-string into a message string of the passed / default language.
get_text(text[, count, key_suffix, language])	translate passed text string into the current language.
load_language_file(file_name, encoding, language)	load file content encoded with the given encoding into the specified language.
load_language_texts([language, encoding, ...])	load translation texts for the given language and optional domain.
plural_key(count)	convert number in *count* into a dict key to access the correct plural form.
register_package_translations()	call from module scope of the package to register/add translations resources path.
register_translations_path([translation_path])	add/register the passed root path as new resource of translation texts.
translation(text[, language])	determine translation for passed text string and language.

MsgType

type of message literals in translation text files

alias of `Union[str, Dict[str, str]]`

LanguageMessages

type of the data structure storing the loaded messages

alias of `Dict[str, Union[str, Dict[str, str]]]`

DEF_ENCODING = 'UTF-8'

encoding of the messages in your app code

DEF_LANGUAGE = 'en'

language code of the messages in your app code

LOADED_TRANSLATIONS: Dict[str, Dict[str, str | Dict[str, str]]] = {}

message text translations of all loaded languages

MSG_FILE_SUFFIX = 'Msg.txt'

name suffix of translation text files

TRANSLATIONS_PATHS: List[str] = ['/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_help/loc', '/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/kivy_qr_displayer/loc', '/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/kivy_sideloading/loc']

file paths to search for translations

default_locale: List[str] = ['en', 'UTF-8']

language and encoding code of the current language/locale

default_encoding(*new_enc=''*)

get and optionally set the default message text encoding.

Parameters
new_enc¶ (str) – new default encoding to be set. kept unchanged if not passed.

Return type
str

Returns
old default encoding (current if *new_enc* get not passed).

default_language(*new_lang=''*)

get and optionally set the default language code.

Parameters
new_lang¶ (str) – new default language code to be set. kept unchanged if not passed.

Return type
str

Returns
old default language (or current one if *new_lang* get not passed).

get_text(*text*, *count=None*, *key_suffix=''*, *language=''*)

translate passed text string into the current language.

Parameters

- **text**¶ (str) – text message to be translated.
- **count**¶ (Optional[int]) – pass int value if the translated text has variants for their pluralization. the count value will be converted into an amount/pluralize key by the function *plural_key()*.
- **key_suffix**¶ (str) – suffix to the key used if the translation is a dict.
- **language**¶ (str) – language code to load (def=current language code in 1st item of *default_locale*).

Return type
str

Returns
translated text message or the value passed into *text* if no translation text got found for the current language.

get_f_string(*f_str*, *key_suffix=''*, *language=''*, *glo_vars=None*, *loc_vars=None*)

translate passed f-string into a message string of the passed / default language.

Parameters

- **f_str**¶ (str) – f-string to be translated and evaluated.
- **key_suffix**¶ (str) – suffix to the key used if the translation is a dict.
- **language**¶ (str) – language code to load (def=current language code in 1st item of *default_locale*).
- **glo_vars**¶ (Optional[Dict[str, Any]]) – global variables used in the conversion of the f-string expression to a string. the globals() of the caller of the callee will be available too and get overwritten by the items of this argument.
- **loc_vars**¶ (Optional[Dict[str, Any]]) – local variables used in the conversion of the f-string expression to a string. the locals() of the caller of the callee will be available too and get overwritten by the items of this argument. pass a numeric value in the *count* item of this dict for pluralized translated texts (see also *count* parameter of the function *get_text()*).

Return type
str

Returns
translated text message including evaluated and formatted variables/expressions of the f-string passed-in *f_str*. if no translation text got found for the current language then the original text message will be returned. any syntax errors and exceptions occurring in the conversion of the f-string are silently ignored.

`load_language_file`(*file_name*, *encoding*, *language*)

load file content encoded with the given encoding into the specified language.

Parameters

- **file_name** (str) – file name, inclusive path and extension to load.
- **encoding** (str) – encoding id string.
- **language** (str) – language id string.

`load_language_texts`(*language=''*, *encoding=''*, *domain=''*, *reset=False*)

load translation texts for the given language and optional domain.

Parameters

- **language** (str) – language code of the translation texts to load. use the default language if not passed.
- **encoding** (str) – encoding to use to load message file.
- **domain** (str) – optional domain id, e.g. the id of an app, attached process or a user. if passed then it will be used as prefix for the message file name to be loaded additionally and after the default translation texts get loaded (overwriting the default translations).
- **reset** (bool) – pass True to clear all previously added language/locale messages.

Return type
str

Returns
language code of the loaded/default language.

`plural_key`(*count*)

convert number in *count* into a dict key to access the correct plural form.

Parameters
count (Optional[int]) – number of items used in the current context or None (resulting in empty string).

Return type
str

Returns
dict key (prefix) within the MsgType part of the translation data structure.

`register_package_translations`()

call from module scope of the package to register/add translations resources path.

no parameters needed because we use here *stack_var()* helper function to determine the the module file path via the *__file__* module variable of the caller module in the call stack. in this call we have to overwrite the default value (*SKIPPED_MODULES*) of the *skip_modules* parameter to not skip ae portions that are providing package resources and are listed in the *SKIPPED_MODULES*, like e.g. *ae.gui_app* and *ae.gui_help* (passing empty string '' to overwrite default skip list).

register_translations_path(*translation_path=''*)

add/register the passed root path as new resource of translation texts.

Parameters
translation_path (str) – root path of a translations folder structure to register, using cwd if not specified.

Return type
bool

Returns
True if the translation folder structure exists and got properly added/registered, else False.

translation(*text*, *language=''*)

determine translation for passed text string and language.

Parameters

- **text** (str) – text message to be translated.
- **language** (str) – language code to load (def=current language code in 1st item of *default_locale*).

Return type
Union[str, Dict[str, str], None]

Returns
None if not found else translation message or dict with plural forms.

4.13 ae.parse_date

4.13.1 parse date strings more flexible and less strict

this module is pure python and has no namespace external dependencies.

the *parse_date()* helper function is converting a wide range of date and datetime string literal formats into the built-in types datetime.datetime and datetime.date.

this function extends (and fully replaces) Pythons standard method strptime() and supports multiple date formats which are much more flexible interpreted.

Functions

parse_date(literal, *additional_formats[, ...])	parse a date literal string, returning the represented date/datetime or None if date literal is invalid.

parse_date(*literal*, **additional_formats*, *replace=None*, *ret_date=False*, *dt_seps=('T', ' ')*, *ti_sep=':'*, *ms_sep='.'*, *tz_sep='+'*)

parse a date literal string, returning the represented date/datetime or None if date literal is invalid.

this function checks/corrects the passed date/time literals to support a wider range of ISO and additional date/time formats as Pythons strptime().

Hint: Pythons `strptime()` to parse date and time strings into `datetime.date` or `datetime.datetime` objects is very strict and does not respect the formatting alternatives of ISO8601 (see https://bugs.python.org/issue15873 and https://github.com/boxed/iso8601).

additionally a `datetime.date` object can be created/returned automatically if no time info is specified in the date string/literal (see *`ret_date`* parameter).

Parameters

- **`literal`**¶ (`str`) – date literal string in the format of `DATE_ISO`, `DATE_TIME_ISO` or in one of the additional formats passed into the *`additional_formats`* arguments.
- **`additional_formats`**¶ (`str`) – additional date literal format string masks (supported mask characters are documented at the *format* argument of the python method `strptime()`).
- **`replace`**¶ (`Optional`[`Dict`[`str`, `Any`]]) – dict of replace keyword arguments for `datetime.datetime.replace()` call. pass e.g. dict(microsecond=0, tzinfo=None) to set the microseconds of the resulting date to zero and to remove the timezone info.
- **`ret_date`**¶ (`Optional`[`bool`]) – request return value type: True=datetime.date, False=datetime.datetime (the default) or None=determine type from literal (short date if dt_seps are not in literal).
- **`dt_seps`**¶ (`Tuple`[`str`, `...`]) – tuple of supported separator characters between the date and time literal parts.
- **`ti_sep`**¶ (`str`) – separator character of the time parts (hours/minutes/seconds) in literal.
- **`ms_sep`**¶ (`str`) – microseconds separator character.
- **`tz_sep`**¶ (`str`) – time-zone separator character.

Return type
`Union`[`date`, `datetime`, `None`]

Returns
represented date/datetime or None if date literal is invalid.

4.14 ae.literal

4.14.1 literal type detection and evaluation

a number, calendar date or other none-text-value gets represented by a literal string, if entered as user input or has to be stored, e.g. as *configuration variable*.

the *`Literal`* class implemented by this portion converts such a *evaluable literal string* into the corresponding value (and type).

Functions

evaluable_literal(literal)	check evaluable format of literal string, possibly return appropriate evaluation function and stripped literal.

Classes

Literal([literal_or_value, value_type, name])	convert literal string into the corresponding value (and type).

evaluable_literal(*literal*)

check evaluable format of literal string, possibly return appropriate evaluation function and stripped literal.

Parameters
literal (str) – string to be checked if it is in the *evaluable literal format* and if it has to be stripped.

Return type
Tuple[Optional[Callable], Optional[str]]

Returns
tuple of evaluation/execution function and the (optionally stripped) literal string (removed triple high-commas on expression/code-blocks) - if *literal* is in one of the supported *evaluable literal formats* - else the tuple (None, <empty string>).

class Literal(*literal_or_value=None*, *value_type=None*, *name='LiT'*)

Bases: object

convert literal string into the corresponding value (and type).

pass the literal string on instantiation as the first (the *literal_or_value*) argument:

```
>>> number = Literal("3")
>>> number
Literal('3')
>>> number.value
3
>>> type(number.value)
<class 'int'>
```

Literal will interpret the value type from the specified literal string. the corresponding *int* value provides the *value* attribute.

to make sure that a number-like literal will be interpreted as a string enclose it in high-commas. the following example will therefore result as a string type:

```
>>> number = Literal("'3'")
>>> number
Literal("'3'")
>>> number.value
'3'
>>> type(number.value)
<class 'str'>
```

another way to ensure the correct value type, is to specify it with the optional *second argument*:

```
>>> number = Literal("3", str)
>>> number
Literal('3')
>>> number.value
'3'
```

alternatively assign the *evaluable literal string* after the instantiation, directly to the *value* attribute of the *Literal* instance:

```
>>> number = Literal(value_type=str)
>>> number.value = "3"
>>> number.value
'3'
```

any type can be specified as the literal value type:

```
>>> my_list = Literal(value_type=list)
>>> my_dict = Literal(value_type=dict)
>>> my_datetime = Literal(value_type=datetime.datetime)
>>> class MyClass:
...     pass
>>> my_instance = Literal(value_type=MyClass)
```

the value type get automatically determined also for *evaluable python expression literal* . for example the following literal gets converted into a *datetime* object:

```
>>> datetime_value = Literal('(datetime.datetime.now())')
```

also if assigned directly to the *value* attribute:

```
>>> date_value = Literal()
>>> date_value.value = '(datetime.date.today())'
```

Note: the literal string of the last two examples has to start and end with round brackets, to mark it as a *evaluable literal*.

---

to convert calendar date literal strings into one of the supported ISO formats (*DATE_TIME_ISO* or *DATE_ISO*), the expected value type has to be specified:

```
>>> date_value = Literal('2033-12-31', value_type=datetime.date)
>>> assert date_value.value == datetime.date(2033, 12, 31)
```

a *ValueError* exception will be raised if the conversion fails, or if the result cannot be converted into the requested value type:

```
>>> date_literal = Literal(value_type=datetime.date)
>>> date_literal.value = "invalid-date-literal"
>>> date_value = date_literal.value
Traceback (most recent call last):
...
ValueError
```

all supported literal formats are documented at the *value* property/attribute.

__init__(*literal_or_value=None*, *value_type=None*, *name='LiT'*)

create new Literal instance.

Parameters

- **literal_or_value** (Optional[Any]) – initial literal (evaluable string expression) or value of this instance.
- **value_type** (Optional[Type]) – type of the value of this instance (def=determined latest by/in the *value* property getter).
- **name** (str) – name of the literal (only used for debugging/error-message).

__repr__()

Return repr(self).

property value: Any

property representing the value of this Literal instance.

Setter
assign literal or a new value; can be either a value literal string or directly the represented/resulting value. if the assigned value is not a string and the value type of this instance got still unspecified then this instance will be restricted to the type of the assigned value. assigning a None value will be ignored - neither the literal nor the value will change with that!

Getter
return the literal value; on the first call the literal will be evaluated (lazy/late) and the value type will be set if still unspecified. further getter calls will directly return the already converted literal value.

if the literal of this *Literal* instance coincide with one of the following evaluable formats then the value and the type of the value gets automatically recognized. an evaluable formatted literal strings has to start and end with one of the character pairs shown in the following table:

starts with	ends with	evaluation value type
(	)	tuple literal or expression
[	]	list literal
{	}	dict literal
‘	‘	string literal
“	“	string literal
‘’’	‘’’	code block with return
“””	“””	code block with return

other supported literals and values

literals with type restriction to a boolean type are evaluated as python expression. this way literal strings like ‘True’, ‘False’, ‘0’ and ‘1’ will be correctly recognized and converted into a boolean value.

literal strings that representing a date value (with type restriction to either datetime.datetime or datetime.date) will be converted with the parse_date() function and should be formatted in one of the standard date formats (defined via the *ae.base* constants *DATE_TIME_ISO* and *DATE_ISO*).

literals and values that are not in one of the above formats will finally be passed to the constructor of the restricted type class to try to convert them into their representing value.

append_value(*item_value*)

add new item to the list value of this Literal instance (lazy *self.value* getter call function pointer).

Parameters
item_value¶ (Any) – value of the item to be appended to the value of this Literal instance.

Return type
Any

Returns
the value (==list) of this Literal instance.

this method gets e.g. used by the *ConsoleApp* method *add_option()* to have a function pointer to this literal value with lazy/late execution of the value getter (value.append cannot be used in this case because the list could have be changed before it get finally read/used).

Note: this method calls the append method of the value object and will therefore only work if the value is of type `list` (or a compatible type).

---

convert_value(*lit_or_val*)

set/change the literal/value of this *Literal* instance and return the represented value.

Parameters
lit_or_val¶ (Any) – the new value to be set.

Return type
Any

Returns
the final/converted value of this Literal instance.

this method gets e.g. used by the *ConsoleApp* method *add_option()* to have a function pointer to let the ArgumentParser convert a configuration option literal into the represented value.

type_mismatching_with(*value*)

check if this literal instance would reject the passed value because of type mismatch.

Parameters
value¶ (Any) – new literal value.

Return type
bool

Returns
True if the passed value would have a type mismatch or if literal type is still not set, else False.

_determine_value(*lit_or_val*)

check passed value if it is still a literal determine the represented value.

Parameters
lit_or_val¶ (Any) – new literal value or the representing literal string.

Return type
Any

Returns
determined/converted value or self._lit_or_val if value could not be recognized/converted.

_chk_val_reset_else_set_type(*value*)

reset and return passed value if is None, else determine value type and set type (if not already set).

Parameters

value¶ (Any) – just converted new literal value to be checked and if ok used to set an unset type.

Return type

Any

Returns

passed value or the stored literal/value if passed value is None.

4.15 ae.progress

4.15.1 display progress of long running processes

this module is simplifying the display of progress messages on the command console/shell of your OS for long running processes.

basic usage of progress portion

create an instance of the *Progress* for each process/thread your application spawns:

```
from ae.core import SubApp
from ae.progress import Progress

app = SubApp(...)
...
progress = Progress(app, ...)
```

now you can call the *next()* method within your long running process on each processed item/percentage:

```
while process_is_not_finished:
    ...
    progress.next()
```

optionally you can request to print an end-message by calling the *finished()* method of your *Progress* instance as soon as the process is finished:

```
...
progress.finished()
```

the above code snippets are printing a start message to your console at the instantiation of *Progress*. then every call of the method *next()* will print the next message and finally the method *finished()* will print an end message.

to use the integrated error counter and automatic printout of an error message to the console, pass the message text of any occurring error to the argument *error_msg* of *next()* or *finished()*.

progress instantiation

the *first argument* expects the instance of the application class (either *AppBase*, *SubApp* or *ConsoleApp*) that spawns the process.

the next three arguments are configuring a run or item counter. and the other arguments may be used to adopt the format of the displayed messages to your needs.

process run counter

depending on the type of process you want to show progress differently, e.g. on each processed item or after passing a certain percentage value and either as incrementing or decrementing number. for that *Progress* provides a counter which can be flexible configured with the arguments *start_counter* and *total_count*.

specifying only *start_counter* results in a countdown. e.g. to display the number of items waiting to be processed::.

    progress = Progress(app, number_of_items_to_be_processed)

by only specifying *total_count* you get an incremental process run counter:

```
progress = Progress(app, total_count=number_of_items_to_be_processed)
```

to display a percentage on the console in 5 percent steps specify *total_count* as 100 percent and *delta* as +5:

```
progress = Progress(app, total_count=100, delta=5)
```

individual message templates

progress displays 5 types of messages on the console. to overwrite one of the generic default message templates, simply pass your template on instantiation of *Progress* into the argument that is displayed underneath directly after the message type:

- ***start*** **-** ***start_msg***
  start message printed on process start and class instantiation
- ***next*** **-** ***next_msg***
  next run message (printed on each call of *next()*)
- ***end*** **-** ***end_msg***
  finished message (printed on call of *finished()*)
- ***error*** **-** ***err_msg***
  error message on any occurring error (printed on each call of either *next()* or *finished()*)
- ***nothing-to_do*** **-** ***nothing_to_do_msg***
  nothing-to-do message printed on process start (class instantiation) if total count is zero

the following table shows which progress state placeholders you can use in which message type:

placeholder	*available in message type*
run_counter	*start, next, end, error*
total_count	*start, next, end, error*
processed_id	*next, end, error*
err_counter	*next, end, error*
err_msg	*next, end, error*

Hint: only the *nothing-to-do* message type does not provide any placeholders.

Classes

Progress(app_base[, start_counter, ...])	display progress on the console/log output for a long running tasks.

class Progress(*app_base*, *start_counter=0*, *total_count=0*, *delta=-1*, *start_msg=''*, *next_msg=''*, *end_msg='Finished processing of {total_count} having {err_counter} failures: ¡{err_msg}!'*, *err_msg='{err_counter} errors on processing {total_count} items, current={run_counter}: ¡{err_msg}!'*, *nothing_to_do_msg=''*)

Bases: object

display progress on the console/log output for a long running tasks.

__init__(*app_base*, *start_counter=0*, *total_count=0*, *delta=-1*, *start_msg=''*, *next_msg=''*, *end_msg='Finished processing of {total_count} having {err_counter} failures: ¡{err_msg}!'*, *err_msg='{err_counter} errors on processing {total_count} items, current={run_counter}: ¡{err_msg}!'*, *nothing_to_do_msg=''*)

prepare print-outs for a new progress (long running process with incrementing or decrementing item counter).

Parameters

- **app_base** (*AppBase*) – instance of the application class that is spawning the long-running process.
- **start_counter** (int) – process item counter start value. counter decrements on each call of *next()* (if *total_count* not specified).
- **total_count** (int) – number of items that will be processed with an incrementing counter. by passing a positive integer the process item counter will be incremented on each call of *next()*.
- **delta** (int) – difference to decrement/increment on each call of *next()*.
- **start_msg** (str) – start message template with placeholders.
- **next_msg** (Optional[str]) – next message - if an empty string get passed then a default message will be provided with placeholders - pass None if you want to suppress the print-out of a next message.
- **end_msg** (str) – end message template with placeholders, pass None if you want to suppress the print-out of an end message (in this case only a new line will be printed).
- **err_msg** (str) – error message template with placeholders.
- **nothing_to_do_msg** (str) – message template printed-out if the values of the two arguments *start_counter* and *total_count* are not specified or are both less or equal to zero.

_app: *AppBase*

used application class instance

_next_msg
: next message template

_end_msg
: end message template

_err_msg
: error message template

_err_counter
: error counter

_run_counter
: item, percentage or run counter

_total_count
: total count of item/percentage/run counter

_delta
: delta value to increment/decrement run counter

next(*processed_id=''*, *error_msg=''*, *next_msg=''*, *delta=0*)
: log the processing of the next item of this long-running task.

  Parameters

  - **processed_id** (str) – id(s) of the next item (to be displayed on console/logging output).
  - **error_msg** (str) – pass the error message to display if the next item produced any errors. if an error message get passed then the *_err_counter* will be incremented.
  - **next_msg** (str) – message to output (use instance message if not passed/empty).
  - **delta** (int) – delta for decrement/increment process run counter (use instance default if not passed).

finished(*processed_id=''*, *error_msg=''*)
: display end of processing for the current item.

  Parameters

  - **processed_id** (str) – id(s) of the next item (to be displayed on console/logging output).
  - **error_msg** (str) – optional error message to display if current items produced any error. if an error message get passed then the *_err_counter* will be incremented.

get_end_message(*processed_id=''*, *error_msg=''*)
: determine message text for finishing the currently processed item.

  Parameters

  - **processed_id** (str) – id(s) of the next item (to be displayed on console/logging output).
  - **error_msg** (str) – optional error message to display if current items produced any error.

  Return type
  str

  Returns
  message text to display.

4.16 ae.updater

4.16.1 application environment updater

this module is providing check functions running on app startup for easy deployment of Python application updates.

updater check functions

update any files on the destination machine with the help of the check functions *check_moves()* and *check_overwrites()*.

the check function *check_local_updates()* checks if your deployment package contains a Python update script that will be executed (and only one time after an app update) on the next startup of your application.

for a temporary work-around or bug-fix you can deploy your application update with a bootstrap Python script which will be executed on every startup of your application. the detection and execution of such bootstrap script is done by the function `check_local_bootstrap()`-

the function *check_all()* combines all the above checks.

the following skeleton of a main app module demonstrates a typical usage of *check_all()*:

```
from python_and_3rd_party_libs import ...
from ae.updater import check_all
from project_local_libs import ...

check_all()

app = WhatEverApp(app_name=...)
...
app.run_app()
```

replace the *check_all()* call with the needed check function(s) if your app only needs certain checks.

Note: make sure that the check function(s) get called before you initialize any app instances if you want to update ony *config variables*, like *application status* or user preferences.

..hint: more info you find in the doc-strings of each of the check functions.

Functions

check_all([copy_src_folder, ...])	check all outstanding scripts to be executed and files to be moved/overwritten.
check_copies([src_folder, dst_folder])	check on new or missing files to be copied from src_folder to the dst_folder.
check_local_bootstraps()	check if ae_bootstrap script exists in the current working directory to be executed on app startup.
check_local_updates()	check if ae_updater script exists in the current working directory to be executed and deleted.
check_moves([src_folder, dst_folder])	check on missing files to be moved from src_folder to the dst_folder.
check_overwrites([src_folder, dst_folder])	check on files to be moved from the source directory and overwritten within the destination directory.

check_copies(*src_folder='ae_updater_copies'*, *dst_folder=''*)

check on new or missing files to be copied from src_folder to the dst_folder.

Parameters

- **src_folder** (str) – path to source folder/directory where the files get copied from. If not specified then COPIES_SRC_FOLDER_NAME will be used.
- **dst_folder** (str) – path to destination folder/directory where the files get copied to. If not specified or if you pass an empty string then the user data/preferences path ({usr}) will be used.

Return type
List[str]

Returns
list of moved files, with their destination path.

check_moves(*src_folder='ae_updater_moves'*, *dst_folder=''*)

check on missing files to be moved from src_folder to the dst_folder.

Parameters

- **src_folder** (str) – path to source folder/directory where the files get moved from. If not specified then MOVES_SRC_FOLDER_NAME will be used. Please note that the source folder itself will neither be moved nor removed (but will be empty after the operation finished).
- **dst_folder** (str) – path to destination folder/directory where the files get moved to. If not specified or if you pass an empty string then the user data/preferences path ({usr}) will be used.

Return type
List[str]

Returns
list of moved files, with their destination path.

check_overwrites(*src_folder='ae_updater_overwrites'*, *dst_folder=''*)

check on files to be moved from the source directory and overwritten within the destination directory.

Parameters

- **src_folder** (str) – path to source folder/directory where the files get moved from. If not specified then MOVES_SRC_FOLDER_NAME will be used. Please note that the source folder itself will neither be moved nor removed (but will be empty after the operation finished).
- **dst_folder** (str) – path to destination folder/directory where the files get moved to. If not specified or if you pass an empty string then the user data/preferences path ({usr}) will be used.

Return type
List[str]

Returns
list of moved and possibly overwritten files, with their destination path.

check_local_updates()

check if ae_updater script exists in the current working directory to be executed and deleted.

Return type
bool

Returns
return value (True) of executed run_updater method (if module&function exists), else False.

check_local_bootstraps()

check if ae_bootstrap script exists in the current working directory to be executed on app startup.

Return type
bool

Returns
return value (True) of executed run_updater function (if module&function exists) else False.

check_all(*copy_src_folder=''*, *move_src_folder=''*, *over_src_folder=''*, *dst_folder=''*)

check all outstanding scripts to be executed and files to be moved/overwritten.

Parameters

- **copy_src_folder** (str) – path to source folder/directory where the files get copied from. if not specified or if you pass an empty string then COPIES_SRC_FOLDER_NAME will be used.
- **move_src_folder** (str) – path to source folder/directory where the files get moved from. if not specified or if you pass an empty string then MOVES_SRC_FOLDER_NAME will be used.
- **over_src_folder** (str) – path to source folder/directory where the files get moved from and overwritten to. if not specified then OVERWRITES_SRC_FOLDER_NAME will be used.
- **dst_folder** (str) – path to destination folder/directory where the files get moved to. if not specified or if you pass an empty string then the user data/preferences path ({usr}) will be used.

Return type
List[str]

Returns
list of processed (copied, moved or overwritten) files, with their destination path.

4.17 ae.console

4.17.1 console application environment

an instance of the *ConsoleApp* class is representing a python application with dynamically configurable logging, debugging features (inherited from *AppBase*), command line arguments and config files and options.

the helper function *sh_exec()* provided by this portion simplifies the execution of shell/console commands.

define command line arguments and options

the methods *add_argument()* and *add_option()* are defining command line arguments and *config options*, finally parsed/loaded by calling *run_app()*:

```
ca = ConsoleApp(app_title="command line arguments demo", app_version="3.6.9")
ca.add_argument('argument_name_or_id', help="Help text for this command line argument")
ca.add_option('option_name_or_id', "help text for this command line option", "default_
↪value")
...
ca.run_app()
```

the values of the commend line arguments and options are determined via the methods *get_argument()* and *get_option()*. additional configuration values, stored in *INI/CFG files*, are accessible via the *get_variable()* method.

auto-collected app name, title and version

if one of the kwargs *app_title* or *app_version* is not specified in the init call of the *ConsoleApp* class instance, then they will automatically get determined from your app main module: the app title from the docstring title, and the application version string from the *__version__* variable:

```
""" module docstring title """
from ae.console import ConsoleApp

__version__ = '1.2.3'

ca = ConsoleApp()

assert ca.app_title == "module docstring title"
assert ca.app_version == '1.2.3'
```

ConsoleApp also determines on instantiation the name/id of your application, if not explicitly specified in *app_name*. other application environment vars/options (like e.g. the application startup folder path and the current working directory path) will be automatically initialized and provided via the app instance.

configuration files, sections, variables and options

a config file consists of config sections, each section provides config variables and config options to parametrize your application at run-time.

config files

configuration files can be shared between apps or used exclusively by one app. the following file names are recognized and loaded automatically on app initialization:

config file	used for .... config variables and options
<any_path_name_and_ext>	application/domain specific
<app_name>.ini	application specific (read-/write-able)
<app_name>.cfg	application specific (read-only)
.app_env.cfg	application/suite specific (read-only)
.sys_env.cfg	general system (read-only)
.sys_env<SYS_ENV_ID>.cfg	the system with SYS_ID (read-only)

the above table is ordered by the preference to search/get the value of a config variable/option. so the values stored in the domain/app specific config file will always precede/overwrite any application and system specific values.

app/domain-specific config files have to be specified explicitly, either on initialization of the *ConsoleApp* instance via the kwarg *additional_cfg_file*, or by calling the method *add_cfg_files()*. they can have any file extension and can be placed into any accessible folder.

all the other config files have to have the specified name with a *.ini* or *.cfg* file extension, and get recognized in the current working directory, in the user data directory (see *ae.paths.user_data_path()*) and in the application installation directory.

config sections

this module is supporting the config file format of Pythons built-in `ConfigParser` class, extended by more complex config value types. the following examples shows a config file with two config sections containing one config option (named *log_file*) and two config variables (*configVar1* and *configVar2*):

```
[aeOptions]
log_file = './logs/your_log_file.log'
configVar1 = ['list-element1', ('list-element2-1', 'list-element2-2', ), {}]

[YourSectionName]
configVar2 = {'key1': 'value 1', 'key2': 2222, 'key3': datetime.datetime.now()}
```

the config section *aeOptions* (defined by *MAIN_SECTION_NAME*) is the default or main section, storing the values of any pre-defined *config option* and of some *config variables*.

config variables

config variables can store complex data types. in the example config file above the config variable *configVar1* holds a list with 3 elements: the first element is a string, the second element a tuple, and the third element an empty dict.

all the values, of which its *repr* string can be evaluated with the built-in `eval()` function, can be stored in a config variable, by calling the `set_variable()` method. to read/fetch their value, call the method `get_variable()` with the name and section names of the config variable. you can specify the type of an config variable via the value passed into `value` argument or by the see `special encapsulated strings`, respectively the config value literal.

the following config variables are pre-defined in the *main config section* and recognized by `this module`, some of them also by the module/portion `ae.core`:

- *debug_level*: debug logging verbosity level *config option*
- *log_file*: ae logging file name (this is also a *config option* - set-able as command line arg)
- *logging_params*: `general ae logging configuration parameters (py and ae logging)`
- *py_logging_params*: python logging configuration
- *onboarding_tour_started*: count the onboarding tour starts since the installation of the app. will be reset to zero after a user registration (by calling `register_user()`)
- *registered_users*: users registered with their OS user name as user id (see `register_user()`)
- *user_id*: id of the app user (default is determined from the *system user name* <*ae.base.os_user_name*>)
- *user_specific_cfg_vars*: list of config variables storing an individual value for each registered user (see section *user specific config variables*)

Note: the value of a config variable can be overwritten by defining an OS environment variable with a name that is equal to the `snake+upper-case converted names` of the config-section and -variable. e.g. declare an OS environment variable with the name *AE_OPTIONS_LOG_FILE* to overwrite the value of the *pre-defined config option/variable log_file*.

---

config options

config options are config variables, defined persistently in the config section `aeOptions`. specifying them on the command line, preceding the option name with two leading hyphen characters, and using an equal character between the name and the option value, overwrites the value stored in the config file:

```
$ your_application --log_file='your_new_log_file.log'
```

the default value of a not specified config option gets searched first in the config files (the exact search order is documented in the doc-string of the method `add_cfg_files()`), or if not found then the default value will be used, that is specified in the definition of the config option (the call of `add_option()`).

the method `get_option()` determines the value of a config option:

```
my_log_file_name = ca.get_option('log_file')
```

use the `set_option()` if you want to change the value of a configuration option at run-time. to read the default value of a config option or variable directly from the available configuration files use the method `get_variable()`. the default value of a config option or variable can also be set or changed directly from within your application by calling the `set_variable()` method.

pre-defined configuration options

for a more verbose logging to the console output specify, either on the command line or in a config files, the config option *debug_level* (or as short option *-D*) with a value of 2 (for verbose). the supported config option values are documented *here*.

the value of the second pre-defined config option *log_file* specifies the log file path/file_name. also this option can be abbreviated on the command line with the short *-L* option id.

Note: after an explicit definition of the optional config option *user_id* via *add_option()* it will be automatically used to initialize the *user_id* attribute.

---

user specific config variables

config variables specified in the set `user_specific_cfg_vars` get automatically recognized as user-specific. override the method :meth`~ConsoleApp._init_default_user_cfg_vars` in your main app instance to define or revoke which config variables the app is storing individually for each user.

Hint: to permit individual sets of user-specific config variables for a user (or group) add the config variable *user_specific_cfg_vars* in the user-specific config file section(s). don't forget in this special case to also add there also this config variable, e.g. as *('aeOptions', 'user_specific_cfg_vars')*.

Module Attributes

MAIN_SECTION_NAME	default name of main config section

Functions

config_value_string(value)	convert passed value to a string to store them in a config/ini file.
sh_exec(command_line[, extra_args, ...])	execute command in the current working directory of the OS console/shell.

Classes

ConsoleApp([app_title, app_name, ...])	provides command line arguments and options, config options, logging and debugging for your application.

MAIN_SECTION_NAME: str = 'aeOptions'

default name of main config section

config_value_string(*value*)

convert passed value to a string to store them in a config/ini file.

Parameters
value¶ (Any) – value to convert to ini variable string/literal.

Return type
str

Returns
ini variable literal string.

Note: *Literal* converts the returned string format back into the representing value.

sh_exec(*command_line*, *extra_args=()*, *console_input=''*, *lines_output=None*, *cae=None*, *shell=False*)

execute command in the current working directory of the OS console/shell.

Parameters

- **command_line**¶ (str) – command line string to execute on the console/shell. could contain command line args separated by whitespace characters (alternatively use *extra_args*).
- **extra_args**¶ (Sequence) – optional sequence of extra command line arguments.
- **console_input**¶ (str) – optional string to be sent to the stdin stream of the console/shell.
- **lines_output**¶ (Optional[List[str]]) – optional list to return the lines printed to stdout/stderr on execution.
- **cae**¶ (Optional[Any]) – optional *ConsoleApp* instance, only used for logging. to suppress any logging output pass *UNSET*.
- **shell**¶ (bool) – pass True to execute command in the default OS shell (see `subprocess.run()`).

Return type
int

Returns
return code of the executed command or 126 if execution raised any other exception.

class ConsoleApp(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*, *cfg_opt_eval_vars=None*, *additional_cfg_files=()*, *cfg_opt_val_stripper=None*, *formatter_class=None*, *epilog=''*, ***logging_params*)

Bases: *AppBase*

provides command line arguments and options, config options, logging and debugging for your application.

most applications only need a single instance of this class. each instance is encapsulating a ConfigParser and a ArgumentParser instance. so only apps with threads and different sets of config options for each thread could create a separate instance of this class.

instance attributes (ordered alphabetically - ignoring underscore characters):

- *_arg_parser* ArgumentParser instance.
- *cfg_opt_choices* valid choices for pre-/user-defined options.
- *cfg_opt_eval_vars* additional dynamic variable values that are getting set via the *cfg_opt_eval_vars* argument of the method *ConsoleApp.__init__()* and get then used in the evaluation of *evaluable config option values*.

- *_cfg_files* iterable of config file names that are getting loaded and parsed (specify additional configuration/INI files via the *additional_cfg_files* argument).
- *cfg_options* pre-/user-defined options (dict of *Literal* instances defined via *add_option()*).
- *_cfg_parser* ConfigParser instance.
- *_main_cfg_fnam* main config file name.
- *_main_cfg_mod_time* last modification datetime of main config file.
- *_cfg_opt_val_stripper* callable to strip option values.
- *_parsed_arguments* ArgumentParser.parse_args() return.

__init__(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*, *cfg_opt_eval_vars=None*, *additional_cfg_files=()*, *cfg_opt_val_stripper=None*, *formatter_class=None*, *epilog=''*, ***logging_params*)

initialize a new *ConsoleApp* instance.

Parameters

- **app_title** (str) – application title/description to set the instance attribute *app_title*.

  if not specified then the docstring of your app's main module will be used (see *example*).

- **app_name** (str) – application instance name to set the instance attribute *app_name*.

  if not specified then base name of the main module file name will be used.

- **app_version** (str) – application version string to set the instance attribute *app_version*.

  if not specified then value of a global variable with the name __version__` will be used (*if declared in the actual call stack*).

- **sys_env_id** (str) – system environment id to set the instance attribute *sys_env_id*.

  this value is also used as file name suffix to load all the system config variables in sys_env<suffix>.cfg. pass e.g. 'LIVE' to init this ConsoleApp instance with config values from sys_envLIVE.cfg.

  the default value of this argument is an empty string.

  Note: if the argument value results as empty string then the value of the optionally defined OS environment variable *AE_OPTIONS_SYS_ENV_ID* will be used as default.

- **debug_level** (int) – default debug level to set the instance attribute *debug_level*.

  the default value of this argument is *DEBUG_LEVEL_DISABLED*.

- **multi_threading** (bool) – pass True if instance is used in multi-threading app.
- **suppress_stdout** (bool) – pass True (for wsgi apps) to prevent any python print outputs to stdout.
- **cfg_opt_eval_vars** (Optional[dict]) – dict of additional application specific data values that are used in eval expressions (e.g. AcuSihotMonitor.ini).
- **additional_cfg_files** (Iterable) – iterable of additional CFG/INI file names (opt. incl. abs/rel. path).
- **cfg_opt_val_stripper** (Optional[Callable]) – callable to strip/reformat/normalize the option choices values.

- **formatter_class**¶ (Optional[Any]) – alternative formatter class passed onto ArgumentParser instantiation.
- **epilog**¶ (str) – optional epilog text for command line arguments/options help text (passed onto ArgumentParser instantiation).
- **logging_params**¶ – all other kwargs are interpreted as logging configuration values - the supported kwargs are all the method kwargs of *init_logging()*.

_cfg_parser

ConfigParser instance

cfg_options: Dict[str, *Literal*]

all config options

cfg_opt_choices: Dict[str, Iterable]

all valid config option choices

cfg_opt_eval_vars: dict

app-specific vars for init of cfg options

_cfg_files: list

list of all found INI/CFG files

_main_cfg_fnam: str

def main config file name

_main_cfg_mod_time: float

main config file modification datetime

_cfg_opt_val_stripper: Optional[Callable]

callable to strip or normalize config option choice values

_parsed_arguments: Optional[Namespace]

storing returned namespace of ArgumentParser.parse_args() call, used to retrieve command line args

_arg_parser: ArgumentParser

ArgumentParser instance

_init_default_user_cfg_vars()

init user default config variables.

override this method to add module-/app-specific config vars that can be set individually per user.

_init_logging(*logging_params*)

determine and init logging config.

Parameters

logging_params¶ (Dict[str, Any]) – logging config dict passed as args by user that will be amended with cfg values.

Return type

Optional[str]

Returns

None if py logging is active, log file name if ae logging is set in cfg or args or empty string if no logging got configured in cfg/args.

the logging configuration can be specified in several alternative places. the precedence on various existing configurations is (the highest precedence first):

- *log_file configuration option* specifies the name of the used ae log file (will be read after initialisation of this app instance)
- *logging_params configuration variable* dict with a *py_logging_params* key to activate python logging
- *logging_params configuration variable* dict with the ae log file name in the key *log_file_name*
- *py_logging_params configuration variable* to use the python logging module
- *log_file configuration variable* specifying ae log file
- `logging_params` dict passing the python logging configuration in the key *py_logging_params* to this method
- `logging_params` dict passing the ae log file in the logging key *log_file_name* to this method

__del__()

deallocate this app instance by calling `ae.core.AppBase.shutdown()`.

property debug_level: int

debug level property:

Getter
return the current debug level of this app instance.

Setter
change the debug level of this app instance.

add_cfg_files(**additional_cfg_files*)

extend list of available and additional config files (in `_cfg_files`).

Parameters
additional_cfg_files¶ (str) – domain/app-specific config file names to be defined/registered additionally.

Return type
str

Returns
empty string on success else line-separated list of error message text.

underneath the search order of the config files variable value - the first found one will be returned:

1. the domain/app-specific *config files* added in your app code by this method. these files will be searched for the config option value in reversed order - so the last added *config file* will be the first one where the config value will be searched.
2. *config files* added via `additional_cfg_files` argument of `ConsoleApp.__init__()` (searched in the reversed order).
3. <app_name>.INI file in the <app_dir>
4. <app_name>.CFG file in the <app_dir>
5. <app_name>.INI file in the <usr_dir>
6. <app_name>.CFG file in the <usr_dir>
7. <app_name>.INI file in the <cwd>
8. <app_name>.CFG file in the <cwd>
9. .sys_env.cfg in the <app_dir>
10. .sys_env<sys_env_id>.cfg in the <app_dir>

11. .app_env.cfg in the <app_dir>
12. .sys_env.cfg in the <usr_dir>
13. .sys_env<sys_env_id>.cfg in the <usr_dir>
14. .app_env.cfg in the <usr_dir>
15. .sys_env.cfg in the <cwd>
16. .sys_env<sys_env_id>.cfg in the <cwd>
17. .app_env.cfg in the <cwd>
18. .sys_env.cfg in the parent folder of the <cwd>
19. .sys_env<sys_env_id>.cfg in the parent folder of the <cwd>
20. .app_env.cfg in the parent folder of the <cwd>
21. .sys_env.cfg in the parent folder of the parent folder of the <cwd>
22. .sys_env<sys_env_id>.cfg in the parent folder of the parent folder of the <cwd>
23. .app_env.cfg in the parent folder of the parent folder of the <cwd>
24. value argument passed into the add_opt() method call (defining the option)
25. default_value argument passed into this method (only if *add_option* didn't get called)

legend of the placeholders in the above search order lists (see also *ae.paths.PATH_PLACEHOLDERS*):

- <*cwd*> is the current working directory of your application (determined with `os.getcwd()`)
- <*app_name*> is the base app name without extension of your main python code file.
- <*app_dir*> is the application data directory (APPDATA/<app_name> in Windows, ~/.config/<app_name> in Linux).
- <*usr_dir*> is the user data directory (APPDATA in Windows, ~/.config in Linux).
- <*sys_env_id*> is the specified argument of *ConsoleApp.__init__()*.

cfg_section_variable_names(*section*, *cfg_parser=None*)

determine current config variable names/keys of the passed config file section.

Parameters

- **section** (`str`) – config file section name.
- **cfg_parser** (`Optional[ConfigParser]`) – ConfigParser instance to use (def=self._cfg_parser).

Return type
`Tuple[str, ...]`

Returns
tuple of all config variable names.

_get_cfg_parser_val(*name*, *section*, *default_value=None*, *cfg_parser=None*)

determine thread-safe the value of a config variable from the config file.

Parameters

- **name** (`str`) – name/option_id of the config variable.
- **section** (`str`) – name of the config section.

- **default_value**¶ (Optional[Any]) – default value to return if config value is not specified in any config file.
- **cfg_parser**¶ (Optional[ConfigParser]) – ConfigParser instance to use (def=self._cfg_parser).

Return type
Any

load_cfg_files(*config_modified=True*)

(re)load and parse all config files.

Parameters
config_modified¶ (bool) – pass False to prevent the refresh/overwrite the initial config file modified date.

is_main_cfg_file_modified()

determine if main config file got modified.

Return type
bool

Returns
True if the content of the main config file got modified/changed.

get_variable(*name*, *section=None*, *default_value=None*, *cfg_parser=None*, *value_type=None*)

get value of *config option*, OS environ or *config variable*.

Parameters

- **name**¶ (str) – name of a *config option* or of an existing/declared *config variable*.
- **section**¶ (Optional[str]) – name of the *config section*. defaulting to the app options section (*MAIN_SECTION_NAME*) if not specified or if None or empty string passed.
- **default_value**¶ (Optional[Any]) – default value to return if config value is not specified in any config file.
- **cfg_parser**¶ (Optional[ConfigParser]) – optional ConfigParser instance to use (def=*_cfg_parser*).
- **value_type**¶ (Optional[Type]) – optional type of the config value. only used for *config variables* and ignored for *config options*.

Return type
Any

Returns

variable value which will be searched in the *config options*, the OS environment and in the *config variables* in the following order and manner:

- **config option** with a name equal to the *name* argument (only if the passed *section* value is either empty, None or equal to *MAIN_SECTION_NAME*).
- **OS environment variable** with a matching snake+upper-cased name, compiled from the *section* and *name* arguments, separated by an underscore character.
- **config variable** with a name and section equal to the values passed into the *name* and *section* arguments.

if no variable could be found then a None value will be returned.

this method has an alias named *get_var()*.

get_var(*name*, *section=None*, *default_value=None*, *cfg_parser=None*, *value_type=None*)

alias of method *get_variable()*

Return type
Any

set_variable(*name*, *value*, *cfg_fnam=None*, *section=None*, *old_name=''*)

set/change the value of a *config variable* and if exists the related config option.

if the passed string in *name* is the id of a defined *config option* and *section* is either empty or equal to the value of *MAIN_SECTION_NAME* then the value of this config option will be changed too.

if the section does not exist it will be created (in contrary to Pythons ConfigParser).

Parameters

- **name** (str) – name/option_id of the config value to set.
- **value** (Any) – value to assign to the config value, specified by the *name* argument.
- **cfg_fnam** (Optional[str]) – file name (def= *_main_cfg_fnam*) to save the new option value to.
- **section** (Optional[str]) – name of the *config section*. defaulting to the app options section (*MAIN_SECTION_NAME*) if not specified or if None or empty string passed.
- **old_name** (str) – old name/option_id that has to be removed (used to rename config option name/key).

Return type
str

Returns
empty string on success else error message text.

this method has an alias named *set_var()*.

set_var(*name*, *value*, *cfg_fnam=None*, *section=None*, *old_name=''*)

alias of method *set_variable()*

Return type
str

add_argument(**args*, ***kwargs*)

define new command line argument.

original/underlying args/kwargs of argparse.ArgumentParser are used - please see the description/definition of add_argument().

this method has an alias named *add_arg()*.

add_arg(**args*, ***kwargs*)

alias of method *add_argument()*

get_argument(*name*)

determine the command line parameter value.

Parameters
name (str) – argument id of the parameter.

Return type
Any

Returns
value of the parameter.

this method has an alias named *get_arg()*.

get_arg(*name*)

alias of method *get_argument()*

Return type
Any

add_option(*name*, *desc*, *value*, *short_opt=''*, *choices=None*, *multiple=False*)

defining and adding a new config option for this app.

Parameters

- **name** (str) – string specifying the option id and short description of this new option. the name value will also be available as long command line argument option (case-sens.).
- **desc** (str) – description and command line help string of this new option.
- **value** (Any) – default value and type of the option. returned by *get_option()* if this option is not specified as command line argument nor exists as config variable in any config file. pass *UNSET* to define a boolean flag option, specified without a value on the command line. the resulting value will be *True* if the option will be specified on the command line, else *False*. specifying a value on the command line results in a *SystemExit* on parsing.
- **short_opt** (Union[str, *UnsetType*]) – short option character. if not passed or passed as '' then the first character of the name will be used. passing *UNSET* or *None* prevents the declaration of a short option. please note that the short options 'h', 'D' and 'L' are already used internally by the classes ArgumentParser and *ConsoleApp*.
- **choices** (Optional[Iterable]) – list of valid option values (optional, default=allow all values).
- **multiple** (bool) – True if option can be added multiple times to command line (optional, default=False).

the value of a config option can be of any type and gets represented by an instance of the *Literal* class. supported value types and literals are documented *here*.

this method has an alias named *add_opt()*.

add_opt(*name*, *desc*, *value*, *short_opt=''*, *choices=None*, *multiple=False*)

alias of method *add_option()*

_change_option(*name*, *value*)

change config option and any references to it.

get_option(*name*, *default_value=None*)

determine the value of a config option specified by its name (option id).

Parameters

- **name** (str) – name/id of the config option.
- **default_value** (Optional[Any]) – default value of the option (if not defined with *add_option*).

Return type
Any

Returns
first found value of the option identified by *name*. the returned value has the same type as the value specified in the *add_option()* call. if not given on the command line, then it gets search next in default config section (*MAIN_SECTION_NAME*) of the collected config files (the exact search order is documented in the doc-string of the method *add_cfg_files()*). if not found in the config file then the default value specified of the option definition (the *add_option()* call) will be used. the other default value, specified in the *default_value* kwarg of this method, will be returned only if the option name/id never got defined.

this method has an alias named *get_opt()*.

get_opt(*name*, *default_value=None*)

alias of method *get_option()*

Return type
Any

set_option(*name*, *value*, *cfg_fnam=None*, *save_to_config=True*)

set or change the value of a config option.

Parameters

- **name** (str) – id of the config option to set.
- **value** (Any) – value to assign to the option, identified by *name*.
- **cfg_fnam** (Optional[str]) – config file name to save new option value. if not specified then the default file name of *set_variable()* will be used.
- **save_to_config** (bool) – pass False to prevent to save the new option value also to a config file. the value of the config option will be changed in any case.

Return type
str

Returns
''/empty string on success else error message text.

this method has an alias named *set_opt()*.

set_opt(*name*, *value*, *cfg_fnam=None*, *save_to_config=True*)

alias of method *set_option()*

Return type
str

parse_arguments()

parse all command line args.

this method get normally only called once and after all the options have been added with *add_option()*. *add_option()* will then set the determined config file value as the default value and then the following call of this method will overwrite it with command line argument value, if given.

property user_id

id of the user of this app.

load_user_cfg()

load users configuration.

register_user(***user_data*)

register the current user and create/copy a new set of user specific config vars.

Parameters
user_data¶ – user data dict.

Note: this method will overwrite an existing user with the same user id, with the passed user data and the config variable values of the current/default user.

---

user_section(*section*, *name*)

return the user section name if the passed (section, name) setting id is user-specific.

Parameters

- **section**¶ (str) – section name.
- **name**¶ (str) – variable name.

Return type
str

Returns
passed section name or user-specific section name.

app_env_dict()

collect run-time app environment data and settings - for app logging and debugging.

Return type
Dict[str, Any]

Returns
dict with app environment data/settings.

run_app()

prepare app run. call after definition of command line arguments/options and before run of app code.

show_help()

print help message, listing defined command line args and options, to console output/stream.

includes command line args defined with *add_argument()*, options defined with *add_option()* and the args/kwargs defined with the respective ArgumentParser methods (see description/definition of print_help() of ArgumentParser).

4.18 ae.sys_core

4.18.1 dynamic system configuration, initialization and connection

this module allows the dynamic (data-driven) configuration and connection of your application to all their needed external systems (like servers, databases, cloud stores, ...).

system classes

the *SystemBase* class represents a single system. the creation of instances of this class is automatically done *dynamically* by the (mostly unique) instance of the class *UsedSystems*.

each instance of *SystemBase* is using a system-specific connector class to establish an internal connection to the represented system. for the implementation of a connector class, simply inherit from the abstract base class `SystemConnectionBase` - also provided by this module.

by creating an instance of the class *UsedSystems* you get a dictionary-like object with all the properties for each configured system. the items of this dict are instances of the class *SystemBase*.

dynamic system properties

the properties of your systems are specified within config *files* as *variables*.

the only mandatory config variable is a dict with the name *availableSystems*, with the system-ids as dict keys. the items of this dict are also dicts with string keys and called *system configs*. only the key *name* is mandatory in these system configs. the following system config dict keys are supported:

- **name** : system name (long name of system-id, only used for repr() and debugging).
- **credential_keys** : List[str] of credential keys needed to connect to this system.
- **feature_keys** : List[str] of feature keys supported by this system.
- **available_rec_types** : Dict[str, str] of supported rec type ids and names.
- **connector_module** : name of the connector package.module (def=ae.sys_data_<system-id-in-lower-case>).
- **connector_class** : name of the connector class (def=<system-id>SysConnector),

the value of each credential and feature key gets automatically dynamically read from a config option or variable. the name of these config options/variables consists of the key string, prefixed with the system id in lower case. so if e.g. the value of a credential key 'User' for the system id 'Abc' will be loaded from a config variable with the name *abcUser*. all the credential and feature config variables can be either stored in the main or the systems section of the config files - see also *config_value()*.

an example of a system configuration you find in the config file *test.cfg*, situated in the tests folder of this package.

multiple system configurations

to provide multiple system configurations for the same application (suite) you can use individual config files for each application instance with a separate config file (see .sys_env<SYS_ENV_ID>.cfg *here*).

Module Attributes

SYS_SECTION_NAME	section name for system config variables

Functions

config_value(var_name, console_app)	determine system configuration setting value from application config options/variables.

Classes

SystemBase(sys_id, console_app, credentials)	instance represents an external system, including their system properties and a connection object.
SystemConnectorBase(system)	abstract system connector base class - sub-class establishes the connection to an external system.
UsedSystems(console_app, *selected_systems, ...)	an instance of this class is keeping a dictionary of all the found and used systems of this application.

SYS_SECTION_NAME = 'aeSystems'

section name for system config variables

config_value(*var_name*, *console_app*)

determine system configuration setting value from application config options/variables.

Parameters

- **var_name** (str) – config variable name. variable value gets searched in the application instance environment, first in the *application config options*, then in the *application config variables* (first within the section MAIN_SECTION_NAME/aeOptions and then within SYS_SECTION_NAME/aeSystems.
- **console_app** (ConsoleApp) – ConsoleApp instance of the application providing these config options/variables.

Return type
Any

Returns
config variable value if found else None.

class SystemBase(*sys_id*, *console_app*, *credentials*, *features=()*, *rec_types=None*)

Bases: object

instance represents an external system, including their system properties and a connection object.

instances: int = 0

__init__(*sys_id*, *console_app*, *credentials*, *features=()*, *rec_types=None*)

create new SystemBase instance.

Parameters

- **sys_id** (str) – unique str to identify a system (also used as prefix/suffix).
- **console_app** (ConsoleApp) – ConsoleApp instance of the application using these systems.
- **credentials** (Dict[str, str]) – dict to access system, containing e.g. user name, password, token, dsn...

- **features** (Sequence[str]) – optional list with special features for this system (see SDF_* constants).
- **rec_types** (Optional[Dict[str, str]]) – optional dict of available record types for this system.

sys_id

system id

console_app

application instance and config environment

credentials

credentials to connect to this system

features

system specific features and configs

available_rec_types: Dict[str, str]

record types available in this system

connection: Any

object with opt. connect() and disconnect()/close() methods

conn_error: str

error message of last system access or dis-/connect

__repr__()

representation string of this *SystemBase* instance.

Return type
str

Returns
representation string of this instance.

connect(*sys_config*, *force_reconnect=False*)

connect this instance to his system using a system connector class.

Parameters

- **sys_config** (Dict[str, str]) – dict with the *dynamic config properties of this system*.
- **force_reconnect** (bool) – optional; pass True to force re-connection (even if self.connection is already initialized).

Return type
str

Returns
error message or empty string in case of no errors while re-connecting.

disconnect()

disconnect this external system.

Return type
str

Returns
error message string or empty string if no errors occurred on disconnection.

class SystemConnectorBase(*system*)

Bases: ABC

abstract system connector base class - sub-class establishes the connection to an external system.

instances: int = 0

__init__(*system*)

create new *SystemBase* instance.

Parameters
system (*SystemBase*) – instance of *SystemBase* class that is representing the system using this connection.

system

SystemBase instance

console_app

application instance and config environment

last_err_msg: str

last system connection error message(s)

__repr__()

representation string of this *SystemBase* instance.

Return type
str

Returns
representation string of this instance.

abstract connect()

abstract method - raising NotImplementedError, so has to be implemented by the sub-class.

Return type
str

class UsedSystems(*console_app*, **selected_systems*, ***entered_credentials*)

Bases: OrderedDict

an instance of this class is keeping a dictionary of all the found and used systems of this application.

the keys of this dictionary-like class are the system ids of your application. the items are instances of the *SystemBase* class.

each instance is providing additionally the following instance attributes:

- *console_app* - instance of the *ConsoleApp* class that is using these systems.
- *available_systems* - system properties loaded from config files.

all the system-specific properties are stored within the *SystemBase* instances.

__init__(*console_app*, **selected_systems*, ***entered_credentials*)

create new instance of *UsedSystems*.

Parameters

- **console_app** (*ConsoleApp*) – ConsoleApp instance of the application using these systems.

- **selected_systems** (str) – optional iterable of system id strings of the available systems that have to be initialized for the app specified by the *console_app* argument. if no system id get passed then all available systems will be initialized from the config variable *availableSystems*.
- **entered_credentials** (str) – optional dict of credentials entered by a user at run-time (overwriting the values set in the related config variable *availableSystems*). the keys of this dictionary starting with the system id in lower case, followed by one of the key strings specified in the **credential_keys** config list (see *dynamic system properties*).

_load_merge_cred_feat(*entered_credentials*)

load and initialize all used system configs and merge entered credentials into it.

Parameters

entered_credentials (Dict[str, str]) – passed over unchanged from *UsedSystems* - see *entered_credentials*.

_add_system(*sys_id*, *credentials*, *features=()*)

add new *SystemBase* instance to this *UsedSystems* instance.

Parameters

- **sys_id** (str) – system id.
- **credentials** (Dict[str, str]) – credentials of the new system to add.
- **features** (Sequence[str]) – optional list with special features for this system (see SDF_* constants).

connect(*force_reconnect=False*)

connect all the selected systems of this instance.

Parameters

force_reconnect (bool) – optional; pass True to force re-connection (even if self.connection is initialized).

Return type

str

Returns

error message or empty string in case of no errors while re-connecting.

disconnect()

disconnect all available and already connected systems.

Return type

str

Returns

error message string or empty string if no errors occurred on disconnection.

4.19 ae.sys_data

4.19.1 external system data structures

this module allows your application to easily manage data structures to interface data flows from and onto external systems.

an external system can be nearly anything: like e.g.: a database or web server or an application or software suite with an interface for data exchanges; even data files can be used as an external system.

the same data can be represented differently on different systems. this module allows you to specify individual data structures for each system and provides hooks to easily integrate system-specific data conversion functions.

data structures

use the classes of this module to define any kind of simple data structures - like lists, mappings and trees:

- a list can get represented by an instance of one of the classes *Records* or *Values*. each *Records* instance is a sequence of 0..n *Record* instances. a *Values* instance is a sequence of 1..n *Value* instances.
- a mapping get represented by an instance of the class *Record*, whereas each mapping item gets represented by an instance of the private class *_Field*.
- a node of a tree structure can be represented by an instance of the classes *Values*, *Records* or *Record*.

by combining these simple data structures you can build any complex data structures. the root of such a complex data structure can be defined by an instance of either *Records* or *Record*.

the leaves of any simple or complex data structure are represented by instances of the *Value* class.

the following diagram is showing a complex data structure with all the possible combinations (of a single system):

data reference index paths

an *index path* is a tuple of indexes/keys that is referencing one part or a value of a data structure.

the items of an index path are either of type int (specifying a list index in a *Values* or *Records* instance) or of type str (specifying a field name of a *Record* instance).

e.g. the index path to reference in the above graph example the field **DA** from the record **Record (root)** would be `('D', 0, 'DA')`. the first item is referencing the field **D** in the root record. the second item **0** is referencing the first item of the *Records* instance/value of field **D**, which is a *Record* instance. and finally the last item **DA** specifies the field name in this *Record* instance.

the hierarchically highest placed *Record* instance in a data structure is called the *root record*, and the *Value* instances at the lowest level in a data structure are the leaves of a data structure.

each *_Field* is providing apart from its field value also a reference to its root record as well as a *root index*. the root index is an index path to reference the *_Field* instance from the root record.

if the format/representation of the data value or of any references differs at one of the used systems then a *_Field* instance allows you to store a separate system-specific value. to access a system-specific value or reference in your data structure you have to specify additionally to the index path also a *system identifier* and optionally a *direction identifier*.

system and direction identifiers

a *system id* is a user-defined string that is uniquely identifying a system and should consist of at least two characters (see *_ASP_SYS_MIN_LEN*).

a *direction id* is specifying the data flow direction. the two pre-defined direction ids are:

- *FAD_FROM* for pulling data from a system or
- *FAD_ONTO* for pushing data onto a system.

the methods *pull()* and *push()* of the *Record* class are used to pull/push and convert system-specific data from/onto a system.

Module Attributes

ACTION_INSERT	insert action
ACTION_UPDATE	update action
ACTION_UPSERT	insert or update (if already exists) action
ACTION_DELETE	delete action
ACTION_SEARCH	search action
ACTION_PARSE	parse action
ACTION_BUILD	build action
ACTION_PULL	pull-from-system action
ACTION_PUSH	push-to-system action
ACTION_COMPARE	compare action
FAT_IDX	main/system field name within parent Record or list index within Records/Values
FAT_VAL	main/system field value - storing one of the VALUE_TYPES instance
FAT_CLEAR_VAL	field default/clear value (init by _Field.set_clear_val(), used by clear_leaves())
FAT_REC	root Record instance
FAT_RCX	field index path (idx_path) from the root Record instance
FAT_CAL	calculator callable
FAT_CHK	validator callable
FAT_CNV	system value converter callable
FAT_FLT	field filter callable
FAT_SQE	SQL expression to fetch field value from db
ALL_FATS	tuple of all pre-defined field aspect types/prefixes
FAD_FROM	FROM field aspect direction
FAD_ONTO	ONTO field aspect direction
IDX_PATH_SEP	separator character used for idx_path values (especially if field has a Record value).
ALL_FIELDS	special key of fields_patches argument of Record.copy() to allow aspect value patching for all fields.
CALLABLE_SUFFIX	suffix for aspect keys - used by _Field.set_aspects()
IDX_TYPES	tuple of classes/types used for system data index path items
PATH_TYPES	path items, plus field name path, plus index path
AspectKeyType	type of _Field aspect key
IdxItemType	types of idx_path items
IdxPathType	idx_path type

continues on next page

Table 4.3 – continued from previous page

IdxTypes	types of either idx_path or idx_path items
NodeType	node types (*NODE_TYPES*)
ListType	list/sequence types (*LIST_TYPES*)
ValueType	value types (*VALUE_TYPES*)
NodeChildType	node child types
FieldCallable	callable with _Field argument
FieldValCallable	callable with _Field and additional argument
VALUE_TYPES	tuple of classes/types used for system data values
LIST_TYPES	tuple of classes/types used for system data lists
NODE_TYPES	tuple of classes/types used for system data nodes
NODE_CHILD_TYPES	tuple of classes/types used for system data node children
PARENT_TYPES	value types that can have children (Value excluded)

Functions

aspect_key(type_or_key[, system, direction])	compiles an aspect key from the given args
aspect_key_direction(key)	determines the direction id string from an aspect key.
aspect_key_system(key)	determines the system id string from an aspect key.
compose_current_index(node, idx_path, ...)	determine tuple with the current indexes.
deeper(deepness, instance)	check and calculate resulting/remaining deepness for Record/_Field/Records.copy() when going one level deeper.
field_name_idx_path(field_name[, ...])	converts a field name path string into an index path tuple.
field_names_idx_paths(field_names)	return list of the full idx paths names for all the fields specified in the field_names argument.
get_current_index(node)	get current index of passed *node*.
idx_path_field_name(idx_path[, add_sep])	convert index path tuple/list into field name string.
init_current_index(node, idx_path, use_curr_idx)	determine current index of *node* and if not set then initialize to the first index path item.
set_current_index(node[, idx, add])	set current index of *node*.
string_to_records(str_val, field_names[, ...])	convert formatted string into a *Records* instance containing several *Record* instances.
template_idx_path(idx_path[, is_sub_rec])	check/determine if *idx_path* is referring to template item.
use_current_index(node, idx_path, use_curr_idx)	determine index path of *node* by using current index of *node* if exists and is enabled by *use_curr_idx* arg.
use_rec_default_root_rec_idx(rec, root_rec)	helper function to determine resulting root record and root index.
use_rec_default_sys_dir(rec, system, direction)	helper function to determine resulting system/direction.

Classes

Record([template, fields, system, ...])	instances of this mapping class are used to represent record-like data structures.
Records([seq])	Records class.
Value([iterable])	represents a value.
Values([seq])	ordered/mutable sequence/list, which contains 0..n instances of the class *Value*.

ACTION_INSERT = 'INSERT'
insert action

ACTION_UPDATE = 'UPDATE'
update action

ACTION_UPSERT = 'UPSERT'
insert or update (if already exists) action

ACTION_DELETE = 'DELETE'
delete action

ACTION_SEARCH = 'SEARCH'
search action

ACTION_PARSE = 'PARSE'
parse action

ACTION_BUILD = 'BUILD'
build action

ACTION_PULL = 'PULL'
pull-from-system action

ACTION_PUSH = 'PUSH'
push-to-system action

ACTION_COMPARE = 'COMPARE'
compare action

FAT_IDX = 'idx'
main/system field name within parent Record or list index within Records/Values

FAT_VAL = 'vle'
main/system field value - storing one of the VALUE_TYPES instance

FAT_CLEAR_VAL = 'vwc'
field default/clear value (init by _Field.set_clear_val(), used by clear_leaves())

FAT_REC = 'rrd'
root Record instance

FAT_RCX = 'rrx'
field index path (idx_path) from the root Record instance

FAT_CAL = 'clc'
calculator callable

FAT_CHK = 'chk'

validator callable

FAT_CNV = 'cnv'

system value converter callable

FAT_FLT = 'flt'

field filter callable

FAT_SQE = 'sqc'

SQL expression to fetch field value from db

ALL_FATS = ('idx', 'vle', 'vwc', 'rrd', 'rrx', 'clc', 'chk', 'cnv', 'flt', 'sqc')

tuple of all pre-defined field aspect types/prefixes

FAD_FROM = 'From'

FROM field aspect direction

FAD_ONTO = 'Onto'

ONTO field aspect direction

IDX_PATH_SEP = '/'

separator character used for idx_path values (especially if field has a Record value).

don't use dot char because this is used e.g. to separate system field names in xml element name paths.

ALL_FIELDS = ''**

special key of fields_patches argument of Record.copy() to allow aspect value patching for all fields.

CALLABLE_SUFFIX = '()'

suffix for aspect keys - used by _Field.set_aspects()

_ASP_TYPE_LEN = 3

aspect key type string length

_ASP_DIR_LEN = 4

aspect key direction string length

_ASP_SYS_MIN_LEN = 2

aspect key system id string length

IDX_TYPES: Tuple[Type, ...] = (<class 'int'>, <class 'str'>)

tuple of classes/types used for system data index path items

PATH_TYPES = (<class 'int'>, <class 'str'>, <class 'tuple'>)

path items, plus field name path, plus index path

AspectKeyType

type of _Field aspect key

IdxItemType

types of idx_path items

alias of Union[int, str]

IdxPathType

idx_path type

alias of Tuple[Union[int, str], ...]

IdxTypes

types of either idx_path or idx_path items

alias of Union[int, str, Tuple[Union[int, str], ...]]

NodeType

node types (*NODE_TYPES*)

alias of Union[*Record*, *Records*]

ListType

list/sequence types (*LIST_TYPES*)

alias of Union[*Values*, *Records*]

ValueType

value types (*VALUE_TYPES*)

alias of Union[*Value*, *Values*, *Record*, *Records*]

NodeChildType

node child types

alias of Union[*_Field*, *Record*]

FieldCallable

callable with _Field argument

alias of Callable[[*_Field*], Any]

FieldValCallable

callable with _Field and additional argument

alias of Callable[[*_Field*, Any], Any]

aspect_key(*type_or_key*, *system=''*, *direction=''*)

compiles an aspect key from the given args

Parameters

- **type_or_key** (str) – either FAT_* type or full key (including already the system and direction)-
- **system** (str) – system id string (if type_or_key is a pure FAT_* constant).
- **direction** (str) – direction string FAD_* constant (if type_or_key is a pure FAT_* constant).

Return type
str

Returns
compiled aspect key as string.

aspect_key_system(*key*)

determines the system id string from an aspect key.

Parameters
key (str) – aspect key string.

Return type
str

Returns
system id (SDI_* constant).

`aspect_key_direction`(*key*)

determines the direction id string from an aspect key.

Parameters
key¶ (`str`) – aspect key string.

Return type
`str`

Returns
direction id (FAD_* constant).

`deeper`(*deepness*, *instance*)

check and calculate resulting/remaining deepness for Record/_Field/Records.copy() when going one level deeper.

Parameters

- **deepness**¶ (`int`) – <0 will be returned unchanged until last level is reached: -1==full deep copy, -2==deep copy until deepest Value, -3==deep copy until deepest _Field.
- **instance**¶ (`Any`) – instance to be processed/copied (if this method is returning != 0/zero).

Return type
`int`

Returns
if deepness == 0 then return 0, if deepness < 0 then return 0 if the deepest level is reached, else (deepness > 0) return deepness - 1.

`field_name_idx_path`(*field_name*, *return_root_fields=False*)

converts a field name path string into an index path tuple.

Parameters

- **field_name**¶ (`Union`[`int`, `str`, `Tuple`[`Union`[`int`, `str`], `...`]]) – field name str or field name index/path string or field index tuple or int (for Records index).
- **return_root_fields**¶ (`bool`) – pass True to also return len()==1-tuple for fields with no deeper path (def=False).

Return type
`Tuple`[`Union`[`int`, `str`], `...`]

Returns
index path tuple (idx_path) or empty tuple if the field has no deeper path and return_root_fields==False.

`field_names_idx_paths`(*field_names*)

return list of the full idx paths names for all the fields specified in the field_names argument.

Parameters
field_names¶ (`Sequence`[`Tuple`[`Union`[`int`, `str`], `...`]]) – sequence/list/tuple of field (main or system) names.

Return type
`List`[`Tuple`[`Union`[`int`, `str`], `...`]]

Returns
list of their idx paths names.

idx_path_field_name(*idx_path*, *add_sep=False*)

convert index path tuple/list into field name string.

Parameters

- **idx_path** (Tuple[Union[int, str], ...]) – index path to convert.
- **add_sep** (bool) – pass True to always separate index with IDX_PATH_SEP. the default value False will only put a separator char if the field value is a Record (to separate the root field name from the subfield name).

Return type
str

Returns
field name string.

compose_current_index(*node*, *idx_path*, *use_curr_idx*)

determine tuple with the current indexes.

Parameters

- **node** (Union[*Values*, *Records*, *Record*]) – root node/list (Record or Records/Values instance) to process.
- **idx_path** (Tuple[Union[int, str], ...]) – index path relative to root node passed in *node* arg.
- **use_curr_idx** (List) – list of index counters within *idx_path* where the current index has to be used.

Return type
Tuple[Union[int, str], ...]

Returns
tuple of current indexes.

get_current_index(*node*)

get current index of passed *node*.

Parameters
node (Union[*Values*, *Records*, *Record*]) – instance of Record or Records (real node) or Values (simple list).

Return type
Union[int, str, None]

Returns
current index value.

init_current_index(*node*, *idx_path*, *use_curr_idx*)

determine current index of *node* and if not set then initialize to the first index path item.

Parameters

- **node** (Union[*Values*, *Records*, *Record*]) – root node/list (Record or Records/Values instance) to process.
- **idx_path** (Tuple[Union[int, str], ...]) – index path relative to root node passed in *node* arg.
- **use_curr_idx** (Optional[List]) – list of index counters within *idx_path* where the current index has to be used.

Return type
Tuple[Union[int, str], ...]

Returns
tuple of current indexes.

set_current_index(*node*, *idx=None*, *add=None*)

set current index of *node*.

Parameters

- **node**¶ (Union[*Values*, *Records*, *Record*]) – root node/list (Record or Records/Values instance) to process.
- **idx**¶ (Union[int, str, None]) – index value to set (str for field name; int for list index); if given *add* will be ignored.
- **add**¶ (Optional[int]) – value to add to list index; will be ignored if the `idx` argument is specified.

Return type
Union[int, str]

Returns
the finally set/new index value.

use_current_index(*node*, *idx_path*, *use_curr_idx*, *check_idx_type=False*, *delta=1*)

determine index path of *node* by using current index of *node* if exists and is enabled by *use_curr_idx* arg.

Parameters

- **node**¶ (Union[*Values*, *Records*, *Record*]) – root node/list (Record or Records/Values instance) to process.
- **idx_path**¶ (Tuple[Union[int, str], ...]) – index path relative to root node passed in *node* arg.
- **use_curr_idx**¶ (Optional[List]) – list of index counters within *idx_path* where the current index has to be used.
- **check_idx_type**¶ (bool) – pass True to additionally check if the index type is correct (def=False).
- **delta**¶ (int) – value to decrease the list index counters within *use_curr_idx* (def=1).

Return type
Tuple[Union[int, str], ...]

Returns
tuple of current indexes.

string_to_records(*str_val*, *field_names*, *rec_sep=','*, *fld_sep='='*, *root_rec=None*, *root_idx=()*)

convert formatted string into a *Records* instance containing several *Record* instances.

Parameters

- **str_val**¶ (str) – formatted string to convert.
- **field_names**¶ (Sequence) – list/tuple of field names of each record
- **rec_sep**¶ (str) – character(s) used in *str_val* to separate the records.
- **fld_sep**¶ (str) – character(s) used in *str_val* to separate the field values of each record.
- **root_rec**¶ (Optional[*Record*]) – root to which the returned records will be added.

- **root_idx** (Tuple[Union[int, str], ...]) – index path from root where the returned records will be added.

Return type
Records

Returns
converted *Records* instance.

template_idx_path(*idx_path*, *is_sub_rec=False*)

check/determine if *idx_path* is referring to template item.

Parameters

- **idx_path** (Tuple[Union[int, str], ...]) – index path to check.
- **is_sub_rec** (bool) – pass True to only check sub-record-fields (will then return always False for root-fields).

Return type
bool

Returns
True if *idx_path* is referencing a template item (with index zero/0), else False.

use_rec_default_root_rec_idx(*rec*, *root_rec*, *idx=()*, *root_idx=()*, *met=''*)

helper function to determine resulting root record and root index.

Parameters

- **rec** (*Record*) – current *Record* instance.
- **root_rec** (Optional[*Record*]) – default root record (def=`rec`).
- **idx** (Optional[Tuple[Union[int, str], ...]]) – current index of *rec*.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – default root index.
- **met** (str) – calling method/function name (used only for assert error message, def='').

Return type
Tuple[*Record*, Tuple[Union[int, str], ...]]

Returns
resulting root record and root index (as tuple).

use_rec_default_sys_dir(*rec*, *system*, *direction*)

helper function to determine resulting system/direction.

Parameters

- **rec** (Optional[*Record*]) – current *Record* instance.
- **system** (Optional[str]) – default system id.
- **direction** (Optional[str]) – default direction (see FAD_* constants).

Return type
Tuple[str, str]

Returns
resulting system and direction (as tuple).

class Value(*iterable=()*, */*)

Bases: list

represents a value.

this class inherits directly from the Python list class. each instance can hold either a (single/atomic) value (which can be anything: numeric, char/string or any object) or a list of these single/atomic values.

__getitem__(*key*)

determine atomic value.

Parameters
key¶ (Union[slice, int]) – list index if value is a list.

Return type
Union[Any, MutableSequence[Any]]

Returns
list item value.

__setitem__(*key*, *value*)

set/initialize list item identified by *key* to the value passed in *value*.

Parameters

- **key**¶ (Union[slice, int]) – list index if value is a list.
- **value**¶ (Any) – the new value of the list item.

Return type
None

__repr__()

representation which can be used to serialize and re-create *Value* instance.

Return type
str

Returns
Value representation string.

__str__()

string representation of this *Value* instance.

Return type
str

Returns
Value string.

property initialized

flag if this *Value* instance got already initialized.

Returns
True if already set to a value, else False.

node_child(*idx_path*, *moan=False*, ***__*)

check if *idx_path* is correct (has to be empty) and if yes then return self.

this method is to simplify the data structure hierarchy implementation.

Parameters

- **idx_path**¶ (Tuple[Union[int, str], ...]) – this argument has to be an empty tuple/list.

- **moan** (bool) – pass True to raise AssertionError if *idx_path* is not empty.

Return type
Optional[*Value*]

Returns
self or None (if *idx_path* is not empty and *moan* == False).

value(**idx_path*, **__)

check if *idx_path* is correct (has to be empty) and if yes then return self.

this method is to simplify the data structure hierarchy implementation.

Parameters
idx_path (Union[int, str]) – this argument has to be an empty tuple.

Return type
Optional[*Value*]

Returns
self or None (if *idx_path* is not empty and *moan* == False).

val(**idx_path*, **__)

check if *idx_path* is correct (either empty or contain one int) and if yes then return list item.

this method is to simplify the data structure hierarchy implementation.

Parameters
idx_path – this argument is either empty or contains a list index.

Returns
atomic/single value or list item value or empty string.

set_val(*val*, **idx_path*, **__)

set a/the value of this instance.

Parameters

- **val** (Any) – simple/atomic value to be set.
- **idx_path** (Union[int, str]) – this argument is either empty or contains a list index.

Returns
self.

copy(*_, **__)

copy the value of this Value instance into a new one.

Returns
new Value instance containing the same immutable value.

clear_leaves(**__)

clear/reset the value of this instance.

use self[-1] = '' to clear only the newest/top val.

Returns
self.

class Values(*seq=()*)

Bases: list

ordered/mutable sequence/list, which contains 0..n instances of the class *Value*.

__init__(*seq=()*)

create new *Values* instance.

Parameters

seq¶ (Iterable) – Iterable used to initialize the new instance (pass list, tuple or other iterable).

__repr__()

Return repr(self).

Return type

str

__str__()

Return str(self).

Return type

str

node_child(*idx_path*, *use_curr_idx=None*, *moan=False*, *selected_sys_dir=None*)

determine and return node instance specified by *idx_path* if exists in this instance or underneath.

Parameters

- **idx_path**¶ (Union[int, str, Tuple[Union[int, str], ...]]) – index path to the node, relative to this instance.
- **use_curr_idx**¶ (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **moan**¶ (bool) – flag to check data integrity; pass True to raise AssertionError if not.
- **selected_sys_dir**¶ (Optional[dict]) – optional dict to return the currently selected system/direction.

Return type

Union[*Value*, *Values*, *Record*, *Records*, None]

Returns

found node instance or None if not found.

value(**idx_path*, *system=''*, *direction=''*, ***kwargs*)

determine the ValueType instance referenced by *idx_path* of this *Values*/*Records* instance.

Parameters

- **idx_path**¶ (Union[int, str]) – index path items.
- **system**¶ (str) – system id.
- **direction**¶ (str) – direction id.
- **kwargs**¶ – extra args (will be passed to underlying data structure).

Return type

Union[*Value*, *Values*, *Record*, *Records*, None]

Returns

found Value or Values instance, or None if not found.

set_value(*value*, **idx_path*, *system=''*, *direction=''*, *protect=False*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*)

set the ValueType instance referenced by *idx_path* of this *Values*/*Records* instance.

Parameters

- **value** (Union[*Value*, *Values*, *Record*, *Records*]) – ValueType instance to set/change.
- **idx_path** (Union[int, str]) – index path items.
- **system** (str) – system id.
- **direction** (str) – direction id.
- **protect** (bool) – pass True to prevent replacement of already existing *ValueType*.
- **root_rec** (Optional[*Record*]) – root record.
- **root_idx** (Tuple[Union[int, str], ...]) – root index.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
Union[*Values*, *Records*]

Returns
self (this instance of *Values* or *Records*).

val(**idx_path*, *system=''*, *direction=''*, *flex_sys_dir=True*, *use_curr_idx=None*, ***kwargs*)

determine the user/system value referenced by *idx_path* of this *Values*/*Records* instance.

Parameters

- **idx_path** (Union[int, str]) – index path items.
- **system** (str) – system id.
- **direction** (str) – direction id.
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **kwargs** – extra args (will be passed to underlying data structure).

Return type
Any

Returns
found user/system value, or None if not found or empty string if value was not set yet.

set_val(*val*, **idx_path*, *protect=False*, *extend=True*, *use_curr_idx=None*)

set the user/system value referenced by *idx_path* of this *Values* instance.

Parameters

- **val** (Any) – user/system value to set/change.
- **idx_path** (Union[int, str]) – index path items.
- **protect** (bool) – pass True to prevent replacement of already existing *ValueType*.
- **extend** (bool) – pass True to allow extension of data structure.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
Values

Returns
self (this instance of *Values*).

copy(*deepness=0*, *root_rec=None*, *root_idx=()*, ***kwargs*)

copy the values/records of this *Values*/*Records* instance.

Parameters

- **deepness** (int) – deep copy levels: <0==see deeper(), 0==only copy current instance, >0==deep copy to deepness value - _Field occupies two deepness: 1st=_Field, 2nd=Value.
- **root_rec** (Optional[*Record*]) – destination root record.
- **root_idx** (Tuple[Union[int, str], ...]) – destination index path (tuple of field names and/or list/Records indexes).
- **kwargs** – additional arguments (will be passed on - most of them used by Record.copy()).

Return type
Union[*Values*, *Records*]

Returns
new instance of self (which is an instance of *Values* or *Records*).

clear_leaves(*system=''*, *direction=''*, *flex_sys_dir=True*, *reset_lists=True*)

clear/reset the user/system values of all the leaves of this *Values*/*Records* instance.

Parameters

- **system** (str) – system id.
- **direction** (str) – direction id (FAD_* constants).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **reset_lists** (bool) – pass False to prevent the reset of all the underlying lists.

Return type
Union[*Values*, *Records*]

Returns

class Record(*template=None*, *fields=None*, *system=''*, *direction=''*, *action=''*, *root_rec=None*, *root_idx=()*, *field_items=False*)

Bases: OrderedDict

instances of this mapping class are used to represent record-like data structures.

isinstance(..., dict) not working if using MutableMapping instead of OrderedDict as super class. and dict cannot be used as super class because instance as kwarg will then not work: see the Groxx's answer in stackoverflow question at https://stackoverflow.com/questions/3387691/how-to-perfectly-override-a-dict/47361653#47361653.

__init__(*template=None*, *fields=None*, *system=''*, *direction=''*, *action=''*, *root_rec=None*, *root_idx=()*, *field_items=False*)

create new Record instance, which is an ordered collection of _Field items.

Parameters

- **template** (Optional[*Records*]) – pass Records instance to use first item/[0] as template (after deep copy / values cleared).

- **fields** (Optional[Iterable]) – OrderedDict/dict of _Field instances (field order is not preserved when using dict) or Record instance (fields will be referenced, not copied!) or list of (field_name, fld_or_val) tuples.
- **system** (str) – main/current system of this record,
- **direction** (str) – interface direction of this record.
- **action** (str) – current action (see ACTION_INSERT, ACTION_SEARCH, ACTION_DELETE, ...)
- **root_rec** (Optional[*Record*]) – root record of this record (def=self will be a root record).
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index of this record (def=()).
- **field_items** (bool) – pass True to get Record items - using __getitem__() - as of type _Field (not as val()).

__repr__()

Return repr(self).

Return type
str

__str__()

Return str(self).

Return type
str

__contains__(*idx_path*)

True if the dictionary has the specified key, else False.

Return type
bool

__getitem__(*key*)

x.__getitem__(y) <==> x[y]

Return type
Any

__setitem__(*key*, *value*)

Set self[key] to value.

node_child(*idx_path*, *use_curr_idx=None*, *moan=False*, *selected_sys_dir=None*)

get the node child specified by *idx_path* relative to this *Record* instance.

Parameters

- **idx_path** (Union[int, str, Tuple[Union[int, str], ...]]) – index path or field name index string.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **moan** (bool) – flag to check data integrity; pass True to raise AssertionError if not.
- **selected_sys_dir** (Optional[dict]) – optional dict to return the currently selected system/direction.

Return type
Union[*Value*, *Values*, *Record*, *Records*, *_Field*, None]

Returns
found node instance or None if not found.

set_node_child(*fld_or_val*, **idx_path*, *system=None*, *direction=None*, *protect=False*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*, *to_value_type=False*)

set/replace the child of the node specified by *idx_path* with the value of *fld_or_val*.

Parameters

- **fld_or_val** (Any) – field or value - to be set.
- **idx_path** (Union[int, str]) – index path of the node child to set/replace.
- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).
- **protect** (bool) – pass True to prevent the replacement of a node child.
- **root_rec** (Optional[*Record*]) – root record of this data structure.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index path of this node.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **to_value_type** (bool) – pass True ensure conversion of *fld_or_val* to a Value instance.

Return type
Record

Returns
self (this Record instance).

value(**idx_path*, *system=None*, *direction=None*, ***kwargs*)

search the Value specified by *idx_path* and return it if found.

Parameters

- **idx_path** (Union[int, str]) – index path of Value.
- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).
- **kwargs** – additional keyword args (to be passed onto underlying data structure).

Return type
Union[*Value*, *Values*, *Record*, *Records*, None]

Returns
found Value instance of None if not found.

set_value(*value*, **idx_path*, *system=None*, *direction=None*, *protect=False*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*)

set/replace the Value instance of the node specified by *idx_path*.

Parameters

- **value** (Union[*Value*, *Values*, *Record*, *Records*]) – Value instance to be set/replaced.
- **idx_path** (Union[int, str]) – index path of the Value to be set.
- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).
- **protect** (bool) – pass True to protect existing node value from to be changed/replaced.
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/Record instance.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
Record

Returns
self (this Record instance).

val(**idx_path*, *system=None*, *direction=None*, *flex_sys_dir=True*, *use_curr_idx=None*, ***kwargs*)

determine the user/system value referenced by *idx_path* of this *Record* instance.

Parameters

- **idx_path** (Union[int, str]) – index path items.
- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **kwargs** – extra args (will be passed to underlying data structure).

Return type
Any

Returns
found user/system value, or None if not found or empty string if value was not set yet.

set_val(*val*, **idx_path*, *system=None*, *direction=None*, *flex_sys_dir=True*, *protect=False*, *extend=True*, *converter=None*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*, *to_value_type=False*)

set the user/system value referenced by *idx_path* of this *Record* instance.

Parameters

- **val** (Any) – user/system value to be set/replaced.
- **idx_path** (Union[int, str]) – index path of the Value to be set.
- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).

- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **protect** (bool) – pass True to protect existing node value from to be changed/replaced.
- **extend** (bool) – pass False to prevent extension of data structure.
- **converter** (Optional[Callable[[*_Field*, Any], Any]]) – converter callable to convert user values between systems.
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index to this node/Record instance.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **to_value_type** (bool) – pass True ensure conversion of *val* to a Value instance.

Return type
Record

Returns
self (this Record instance).

_add_field(*field*, *idx=''*)

add _Field instance to this Record instance.

Parameters

- **field** (*_Field*) – _Field instance to add.
- **idx** (str) – name/key/idx string to map and identify this field (mostly identical to field.name()).

Return type
Record

Returns
self.

add_fields(*fields*, *root_rec=None*, *root_idx=()*)

adding fields to this Record instance.

Parameters

- **fields** (Iterable) – either a dict, a Record or a list with (key/field_name, val/_Field) tuples. key strings containing digits/numbers are interpreted as name/idx paths (then also the specified sub-Records/-Fields will be created). values can be either _Field instances or field values.
- **root_rec** (Optional[*Record*]) – root record of this record (def=self will be a root record).
- **root_idx** (Tuple[Union[int, str], ...]) – root index of this record (def=()).

Return type
Record

Returns
self

add_system_fields(*system_fields*, *sys_fld_indexes=None*, *system=None*, *direction=None*, *extend=True*)

add/set fields to this *Record* instance from the field definition passed into *system_fields*.

make sure before you call this method that this *Record* instance has the system and direction attributes specified/set.

Parameters

- **system_fields** (Iterable[Iterable[Any]]) – tuple/list of tuples/lists with system and main field names and optional field aspects. the index of the field names and aspects within the inner tuples/lists get specified by sys_fld_indexes.
- **sys_fld_indexes** (Optional[Dict]) – mapping/map-item-indexes of the inner tuples of system_fields. keys are field aspect types (FAT_* constants), optionally extended with a direction (FAD_* constant) and a system (SDI_* constant). if the value aspect key (FAT_VAL) contains a callable then it will be set as the calculator (FAT_CAL) aspect; if contains a field value then also the clear_val of this field will be set to the specified value.
- **system** (Optional[str]) – system of the fields to be added - if not specified `system` will be used.
- **direction** (Optional[str]) – direction (FAD constants) of the fields to be added - if not passed used self.direction.
- **extend** (bool) – True=add not existing fields, False=apply new system aspects only on existing fields.

Return type
Record

Returns
self

collect_system_fields(*sys_fld_name_path*, *path_sep*)

compile list of system *_Field* instances of this *Record* instance.

Parameters

- **sys_fld_name_path** (Sequence) – sequence/tuple/list of system field names/keys.
- **path_sep** (str) – system field name/key separator character(s).

Return type
List[*_Field*]

Returns
list of :class:'_Field` instances which are having system field name/keys set.

compare_leaves(*rec*, *field_names=()*, *exclude_fields=()*)

compare the leaf 'compare' values of this *Record* instance with the one passed into *rec*.

'Compare values' are simplified user/system values generated by *Record.compare_val()*.

Parameters

- **rec** (*Record*) – other *Record* instance to compare to.
- **field_names** (Iterable) – field names to include in compare; pass empty tuple (==default) to include all fields.
- **exclude_fields** (Iterable) – field names to exclude from compare.

Return type
List[str]

Returns
list of str with the differences between self and *rec*.

compare_val(**idx_path*)

determine normalized/simplified user/system value of the node specified by *idx_path*.

Parameters
idx_path¶ (Union[int, str]) – index path to the node.

Return type
Any

Returns
normalized/simplified compare value.

copy(*deepness=0*, *root_rec=None*, *root_idx=()*, *onto_rec=None*, *filter_fields=None*, *fields_patches=None*)

copy the fields of this record.

Parameters

- **deepness**¶ (int) – deep copy level: <0==see deeper(), 0==only copy this record instance, >0==deep copy to deepness value - _Field occupies two deepness: 1st=_Field, 2nd=Value.
- **root_rec**¶ (Optional[*Record*]) – destination root record - using onto_rec/new record if not specified.
- **root_idx**¶ (Tuple[Union[int, str], ...]) – destination root index (tuple/list with index path items: field names, list indexes).
- **onto_rec**¶ (Optional[*Record*]) – destination record; pass None to create new Record instance.
- **filter_fields**¶ (Optional[Callable[[*_Field*], Any]]) – method called for each copied field (return True to filter/hide/not-include into copy).
- **fields_patches**¶ (Optional[Dict[str, Dict[str, Union[str, *Value*, *Values*, *Record*, *Records*, Callable[[*_Field*], Any]]]]]) – dict[field_name_or_ALL_FIELDS:dict[aspect_key:val_or_callable]] to set/overwrite aspect values in each copied _Field instance. the keys of the outer dict are either field names or the ALL_FIELDS value; aspect keys ending with the CALLABLE_SUFFIX have a callable in the dict item that will be called for each field with the field instance as argument; the return value of the callable will then be used as the (new) aspect value. set the aspect value that stores the field value (aspect key == *FAT_VAL*) by passing a data structure instance (of type *ValueType*).

Return type
Record

Returns
new/extended record instance.

clear_leaves(*system=''*, *direction=''*, *flex_sys_dir=True*, *reset_lists=True*)

clear the leaf values including this *Record* instance and all deeper data structures.

Parameters

- **system**¶ (str) – system id (pass None to use default system id of this *Record* instance).
- **direction**¶ (str) – direction id (pass None to use default direction id of this *Record* instance).

- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **reset_lists** (bool) – pass False to prevent reset of sub-lists.

Return type
Record

Returns

leaves(*system=None*, *direction=None*, *flex_sys_dir=True*)

generate leaves/_Field-instances of this *Record* instance.

Parameters

- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Generator[*_Field*, None, None]

Returns
leaf/_Field-instance generator.

leaf_indexes(**idx_path*, *system=None*, *direction=None*, *flex_sys_dir=True*)

generate leaf-/_Field-index paths for all fields of this *Record* instance.

Parameters

- **idx_path** (Union[int, str]) – index path to be added as base index path (index path to this *Record* instance).
- **system** (Optional[str]) – system id (pass None to use default system id of this *Record* instance).
- **direction** (Optional[str]) – direction id (pass None to use default direction id of this *Record* instance).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Generator[Tuple[Union[int, str], ...], None, None]

Returns
leaf/_Field-instance generator.

leaf_names(*system=''*, *direction=''*, *col_names=()*, *field_names=()*, *exclude_fields=()*, *name_type=None*)

compile a tuple of name types (specified by *name_type*) for this *Record* instance.

Parameters

- **system** (str) – system id (pass None to use default system id of this *Record* instance).
- **direction** (str) – direction id (pass None to use default direction id of this *Record* instance).
- **col_names** (Iterable) – system field/column names to include; pass empty tuple (==default) to include all.
- **field_names** (Iterable) – field names to include; pass empty tuple (==default) to include all fields.

- **exclude_fields** (Iterable) – field names to exclude.
- **name_type** (Optional[str]) – type of name to be included/returned - see available name types underneath.

Return type
Tuple[Union[int, str, Tuple[Union[int, str], ...]], ...]

Returns
tuple of field names/indexes of the included/found leaves.

possible values for the *name_type* argument are:

- 's': user/system field/column name.
- 'f': field name.
- 'r': root name (index path converted into string by *idx_path_field_name()*).
- 'S': index path with user/system field/column name of leaf.
- 'F': index path tuple.
- **None: names depends on each leaf:**
  - root name if leaf is not a root field.
  - user/system name if *system* is not empty/None.
  - field name.

merge_leaves(*rec*, *system=''*, *direction=''*, *flex_sys_dir=True*, *extend=True*)

merge the leaves of the other record in *rec* into this *Record* instance.

Parameters

- **rec** (*Record*) – other *Record* to merge into this one.
- **system** (str) – system id (pass None to use default system id of this *Record* instance).
- **direction** (str) – direction id (pass None to use default direction id of this *Record* instance).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **extend** (bool) – pass False to prevent extension of this data structure.

Return type
Record

Returns
self (this *Record* instance).

match_key(*match_fields*)

make tuple of user/system values for all the fields in *match_fields*.

Parameters
match_fields (Iterable) – Iterable with field names/index paths.

Return type
tuple

Returns
tuple of user/system values.

merge_values(*rec*, *system=''*, *direction=''*, *flex_sys_dir=True*, *extend=True*)

merge user/system values of the other record in *rec* into this *Record* instance.

Parameters

- **rec** (*Record*) – other *Record* to merge into this one.
- **system** (str) – system id (pass None to use default system id of this *Record* instance).
- **direction** (str) – direction id (pass None to use default direction id of this *Record* instance).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **extend** (bool) – pass False to prevent extension of this data structure.

Return type
Record

Returns
self (this *Record* instance).

missing_fields(*required_fields=()*)

check field-names/index-paths specified in *required_fields*.

Parameters
required_fields (Iterable) – list/tuple/iterable of field names or index paths of required fields.

Return type
List[Tuple[Union[int, str], ...]]

Returns
list of index paths for the ones that are missing or having an empty/None value.

pop(*idx*, *default=None*)

check if field name exists and if yes then remove the *_Field* instance from this *Record* instance.

Parameters

- **idx** (str) – field name.
- **default** (Optional[*_Field*]) – return value if a field with the name specified by idx does not exist in this Record.

Return type
Optional[*_Field*]

Returns
removed *_Field* instance if found, else None.

this method got added because the OrderedDict.pop() method does not call __delitem__() (see also __contains__())

pull(*from_system*)

pull all user/system values and convert them into field values.

Parameters
from_system (str) – system id of the system to pull from.

Return type
Record

Returns
self.

push(*onto_system*)

push field values of this *Record* instance to the related user/system values (converted).

Parameters
onto_system (str) – system id of the system to push to.

Return type
Record

Returns
self.

set_current_system_index(*sys_fld_name_prefix*, *path_sep*, *idx_val=None*, *idx_add=1*)

check and if possible set the current system index of this *Record* instance.

Parameters

- **sys_fld_name_prefix** (str) – user/system field name prefix.
- **path_sep** (str) – user/system name path separator.
- **idx_val** (Optional[int]) – new current system index value to set (if passed then *idx_add* will be ignored).
- **idx_add** (Optional[int]) – increment current system index value with the passed int value.

Return type
Optional[*Record*]

Returns
self (this *Record* instance) if current index got changed, else None.

set_env(*system=None*, *direction=None*, *action=None*, *root_rec=None*, *root_idx=()*)

set the environment (system/direction/action) of this *Record* instance.

Parameters

- **system** (Optional[str]) – system id (pass None to leave unchanged).
- **direction** (Optional[str]) – direction id (pass None to leave unchanged).
- **action** (Optional[str]) – action id (pass None to leave unchanged).
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index to this node/Record instance.

Return type
Record

Returns
self.

sql_columns(*from_system*, *col_names=()*)

return list of sql column names for given system.

Parameters

- **from_system** (str) – system from which the data will be selected/fetched.
- **col_names** (Iterable) – optionally restrict to select columns to names given in this list.

Return type
List[str]

Returns
list of sql column names.

sql_select(*from_system*, *col_names=()*)

return list of sql column names/expressions for given system.

Parameters

- **from_system** (str) – system from which the data will be selected/fetched.
- **col_names** (Iterable) – optionally restrict to select columns to names given in this list.

Return type
List[str]

Returns
list of sql column names/expressions.

to_dict(*filter_fields=None*, *key_type=<class 'str'>*, *push_onto=True*, *use_system_key=True*, *put_system_val=True*, *put_empty_val=False*, *system=None*, *direction=None*)

copy Record leaf values into a dict.

Parameters

- **filter_fields** (Optional[Callable[[*_Field*], Any]]) – callable returning True for each field that need to be excluded in returned dict, pass None to include all fields (if put_empty_val == True).
- **key_type** (Union[Type[str], Type[tuple], None]) – type of dict keys: None=field name, tuple=index path tuple, str=index path string (def).
- **push_onto** (bool) – pass False to prevent self.push(system).
- **use_system_key** (bool) – pass False to put leaf field name/index; def=True to use system field name/keys, specified by the system/direction args.
- **put_system_val** (bool) – pass False to include/use main field val; def=True to include system val specified by the system/direction args.
- **put_empty_val** (bool) – pass True to also include fields with an empty value (None/'').
- **system** (Optional[str]) – system id to determine included leaf and field val (if put_system_val == True).
- **direction** (Optional[str]) – direction id to determine included leaf and field val (if put_system_val == True).

Return type
Dict[Union[int, str, Tuple[Union[int, str], ...]], Any]

Returns
dict of filtered leaf user/system values with field names/idx_path-tuples as their key.

update(*mapping=()*, ***kwargs*)

update this *Record* instance - overwriting/extending OrderedDict/super().update() to return self.

Parameters

- **mapping** – mapping to use to update this *Record* instance.
- **kwargs** – optional keyword arguments.

Returns
self (this *Record* instance).

class Records(*seq=()*)

Bases: *Values*

Records class.

each instance of this *Records* class is a list of 0..n *Record* instances.

the not overwritten methods of the inherited *Values* class are also available - like e.g. *Values.node_child()* or *Values.val()*.

__init__(*seq=()*)

create new *Records* instance.

Parameters
seq (Iterable) – Iterable used to initialize the new instance (pass list, tuple or other iterable).

__getitem__(*key*)

x.__getitem__(y) <==> x[y]

Return type
Union[*Value*, *Values*, *Record*, *Records*, List[Union[*Value*, *Values*, *Record*, *Records*]]]

__setitem__(*key*, *value*)

Set self[key] to value.

set_node_child(*rec_or_fld_or_val*, **idx_path*, *system=''*, *direction=''*, *protect=False*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*)

set/replace the child of the node specified by *idx_path* with the value of *rec_or_fld_or_val*.

Parameters

- **rec_or_fld_or_val** – record, field or value - to be set.
- **idx_path** (Union[int, str]) – index path of the node child to set/replace.
- **system** (str) – system id (pass None to use default system id of this *Records* instance).
- **direction** (str) – direction id (pass None to use default direction id of this *Records* instance).
- **protect** (bool) – pass True to prevent the replacement of a node child.
- **root_rec** (Optional[*Record*]) – root record of this data structure.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index path of this node.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
Records

Returns
self (this *Records* instance).

set_val(*val*, **idx_path*, *system=''*, *direction=''*, *flex_sys_dir=True*, *protect=False*, *extend=True*, *converter=None*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*)

set the user/system value referenced by *idx_path* of this *Records* instance.

Parameters

- **val** (Any) – user/system value to be set/replaced.

- **idx_path** (Union[int, str]) – index path of the Value to be set.
- **system** (str) – system id (pass None to use default system id of this *Records* instance).
- **direction** (str) – direction id (pass None to use default direction id of this *Records* instance).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **protect** (bool) – pass True to protect existing node value from to be changed/replaced.
- **extend** (bool) – pass False to prevent extension of data structure.
- **converter** (Optional[Callable[[*_Field*, Any], Any]]) – converter callable to convert user values between systems.
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index to this node/Records instance.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
Records

Returns
self (this Records instance).

append_record(*root_rec*, *root_idx=()*, *from_rec=None*, *clear_leaves=True*)

add/append Record to this *Records* instance.

Parameters

- **root_rec** (*Record*) – root Record instance of this data structure.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/Records instance.
- **from_rec** (Optional[*Record*]) – *Record* instance to append, if not passed then use a copy of the first/template Record of this *Records* instance, else create new *Record* instance.
- **clear_leaves** (bool) – pass False to not clear the leaf values.

Return type
Record

Returns
added/appended *Record* instance.

compare_records(*records*, *match_fields*, *field_names=()*, *exclude_fields=()*, *record_comparator=None*)

compare the records of this instance with the ones passed in the *records* argument.

Parameters

- **records** (*Records*) – other instance of *Records* to be compared against self.
- **match_fields** (Iterable) – iterable with field names/index paths to determine each *Record* id/pkey.
- **field_names** (Iterable) – iterable with field names/index paths that get compared.
- **exclude_fields** (Iterable) – iterable with field names/index-paths that get excluded from to be compared.

- **record_comparator** (Optional[Callable[[*Record*, *Record*], List[str]]]) – optional callable for additional compare (called for each Record).

Return type
List[str]

Returns
list of differences.

index_match_fields(*match_fields*)

create/initialize match index for this *Records* instance (stored in *match_index* attribute).

Parameters
match_fields (Iterable) – iterable with field names/index paths to determine each *Record* id/pkey.

Return type
Records

Returns
self (this *Records* instance).

leaves(*system=''*, *direction=''*, *flex_sys_dir=True*)

generate leaves/_Field-instances of this *Records* instance.

Parameters

- **system** (str) – system id (pass None to use default system id of the underlying *Record* instances).
- **direction** (str) – direction id (pass None to use default direction of the underlying *Record* instances).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Generator[*_Field*, None, None]

Returns
leaf/_Field-instance generator.

leaf_indexes(**idx_path*, *system=''*, *direction=''*, *flex_sys_dir=True*)

generate leaf-/_Field-index paths for all fields of this *Records* instance.

Parameters

- **idx_path** (Union[int, str]) – index path to be added as base index path (index path to this *Records* instance).
- **system** (str) – system id (pass None to use default system id of the underlying *Record* instances).
- **direction** (str) – direction id (pass None to use default direction of the underlying *Record* instances).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Generator[Tuple[Union[int, str], ...], None, None]

Returns
leaf/_Field-instance generator.

merge_records(*records*, *match_fields=()*)

merge the records passed in *records* into this *Records* instance.

Parameters

- **records** (*Records*) – records to be merged in.
- **match_fields** (Iterable) – match fields used to identify already existing records (merge values in these cases).

Return type
Records

Returns
self.

set_env(*system=''*, *direction=''*, *root_rec=None*, *root_idx=()*)

set the environment (system/direction/action) of each record underneath this *Records* instance.

Parameters

- **system** (Optional[str]) – system id (pass None to leave unchanged).
- **direction** (Optional[str]) – direction id (pass None to leave unchanged).
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index to this node/Record instance.

Return type
Records

Returns
self.

class _Field(*root_rec=None*, *root_idx=()*, *allow_values=False*, ***aspects*)

Bases: object

Internal/Private class used by *Record* to create record field instances.

an instance of *_Field* is representing one record field. the field properties are internally stored within a private dict (_Field._aspects) and are called the 'aspects' of a field.

field aspects get used by a field instance e.g. to: * store field value(s) * define callable(s) to convert, filter or validate field values * associate the root record and root index * store any other user-defined field properties (like sql column expressions, comments, ...)

the FAT_ALL constant contains all pre-defined aspects (see the other FAT_* constants defined at the top of this module). these values are called 'aspect keys' and are used as dict keys in the private dict.

each aspect can additionally have a separate property for each system/direction - in this case the aspect key gets extended with direction/system ids. the two available direction ids are pre-defined by the constants *FAD_FROM* and *FAD_ONTO*. the system ids are not defined in this module, they have to be defined by the application. aspect keys can be compiled with the function *aspect_key()*.

__init__(*root_rec=None*, *root_idx=()*, *allow_values=False*, ***aspects*)

__repr__()

Return repr(self).

__str__()

Return str(self).

__getitem__(*key*)

node_child(*idx_path*, *use_curr_idx=None*, *moan=False*, *selected_sys_dir=None*)

get the node child specified by *idx_path* relative to this *_Field* instance.

Parameters

- **idx_path** (Union[int, str, Tuple[Union[int, str], ...]]) – index path or field name index string.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **moan** (bool) – flag to check data integrity; pass True to raise AssertionError if not.
- **selected_sys_dir** (Optional[dict]) – optional dict to return the currently selected system/direction.

Return type
Union[*Value*, *Values*, *Record*, *Records*, *_Field*, None]

Returns
found node instance or None if not found.

value(**idx_path*, *system=''*, *direction=''*, *flex_sys_dir=False*)

search the Value specified by *idx_path* and return it if found.

Parameters

- **idx_path** (Union[int, str]) – index path of Value.
- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass True to allow fallback to main (non-user/-system) value.

Return type
Union[*Value*, *Values*, *Record*, *Records*, None]

Returns
found Value instance of None if not found.

set_value(*value*, **idx_path*, *system=''*, *direction=''*, *protect=False*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*)

set/replace the Value instance of the node specified by *idx_path*.

Parameters

- **value** (Union[*Value*, *Values*, *Record*, *Records*]) – Value instance to be set/replaced.
- **idx_path** (Union[int, str]) – index path of the Value to be set.
- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **protect** (bool) – pass True to protect existing node value from to be changed/replaced.
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/Record instance.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
_Field

Returns
self (this _Field instance).

val(**idx_path*, *system=''*, *direction=''*, *flex_sys_dir=True*, *use_curr_idx=None*, ***kwargs*)

determine the user/system value referenced by *idx_path*.

Parameters

- **idx_path** (Union[int, str]) – index path items.
- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **kwargs** – extra args (will be passed to underlying data structure).

Return type
Any

Returns
found user/system value, or None if not found or empty string if value was not set yet.

set_val(*val*, **idx_path*, *system=''*, *direction=''*, *flex_sys_dir=True*, *protect=False*, *extend=True*, *converter=None*, *root_rec=None*, *root_idx=()*, *use_curr_idx=None*, *to_value_type=False*)

set the user/system value referenced by *idx_path*.

Parameters

- **val** (Any) – user/system value to be set/replaced.
- **idx_path** (Union[int, str]) – index path of the Value to be set.
- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **protect** (bool) – pass True to protect existing node value from to be changed/replaced.
- **extend** (bool) – pass False to prevent extension of data structure.
- **converter** (Optional[Callable[[*_Field*, Any], Any]]) – converter callable to convert user values between systems.
- **root_rec** (Optional[*Record*]) – root Record instance of this data structure.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/Record instance.
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.
- **to_value_type** (bool) – pass True ensure conversion of *val* to a Value instance.

Return type
_Field

Returns
self (this _Field instance).

leaf_value(*system=''*, *direction=''*, *flex_sys_dir=False*)

determine the leaf value of this field (and optionally system/direction).

Parameters

- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Union[*Value*, *Values*, *Record*, *Records*, None]

Returns
the main or user/system value of this leaf/field or None if deeper value exists and *flex_sys_dir* is False.

used also to check if a deeper located sys field exists in the current data structure.

leaves(*system=''*, *direction=''*, *flex_sys_dir=True*)

generate all sub-leaves/_Field-instances underneath this *_Field* instance.

Parameters

- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Generator[*_Field*, None, None]

Returns
leaf/_Field-instance generator.

leaf_indexes(**idx_path*, *system=''*, *direction=''*, *flex_sys_dir=True*)

generate leaf-/_Field-index paths for all subfields underneath (if exist) or this *_Field* instance.

Parameters

- **idx_path** (Union[int, str]) – index path to this *_Field* instance.
- **system** (str) – system id ('' stands for the main/system-independent value).
- **direction** (str) – direction id ('' stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
Generator[Tuple[Union[int, str], ...], None, None]

Returns
leaf/_Field-instance generator.

find_aspect_key(**aspect_types*, *system=''*, *direction=''*)

search for the passed *aspect_types* in this *_Field* instance.

Parameters

- **aspect_types** (str) – aspect types (dict key prefixes) to search for.
- **system** (Optional[str]) – system id ('' stands for the main/system-independent value).
- **direction** (Optional[str]) – direction id ('' stands for the main/system-independent value).

Return type
Optional[str]

Returns
the full aspect key of the first found aspect that is matching the passed aspect type and optionally also the passed system and direction ids.

the search will be done in the following order: * all passed aspect types including the passed system/direction ids. * all passed aspect types including the passed system and both directions (if direction id get not passed). * all passed aspect types including the passed system and without direction id. * all passed aspect types without system and direction ids.

set_env(*system=''*, *direction=''*, *root_rec=None*, *root_idx=()*)

set the environment (system/direction/action) of each record underneath this *_Field* instance.

Parameters

- **system** (Optional[str]) – system id (don't pass or pass None to leave unchanged).
- **direction** (Optional[str]) – direction id (don't pass or pass None to leave unchanged).
- **root_rec** (Optional[*Record*]) – root Record instance of this field, system and direction.
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index to this node/*_Field* instance.

Return type
_Field

Returns
self (this *_Field* instance).

set_system_root_rec_idx(*system=None*, *direction=None*, *root_rec=None*, *root_idx=()*)

set the root record and index of the data structure where this *_Field* instance is included.

Parameters

- **system** (Optional[str]) – system id (don't pass or pass None to leave unchanged).
- **direction** (Optional[str]) – direction id (don't pass or pass None to leave unchanged).
- **root_rec** (Optional[*Record*]) – new root Record instance of this data structure (pass None to leave unchanged).
- **root_idx** (Optional[Tuple[Union[int, str], ...]]) – root index to this node/Record instance (pass None to determine).

Return type
_Field

Returns
self (this *_Field* instance).

aspect_exists(**aspect_types*, *system=''*, *direction=''*, *flex_sys_dir=False*)

check if aspect exists and return full aspect key (including system/direction) if yes.

Parameters

- **aspect_types** (str) – aspect types (dict key prefixes) to search for.
- **system** (str) – system id (def='' stands for the main/system-independent value).
- **direction** (str) – direction id (def='' stands for the main/system-independent value).

- **flex_sys_dir** (Optional[bool]) – pass False to prevent fallback to system-independent value.

Return type
Optional[str]

Returns
the full aspect key of the first found aspect that is matching the passed aspect type and optionally also the passed system and direction ids.

Returns

aspect_value(**aspect_types*, *system=''*, *direction=''*, *flex_sys_dir=False*)

Parameters

- **aspect_types** (str) – aspect types (dict key prefixes) to search for.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **flex_sys_dir** (Optional[bool]) – pass False to prevent fallback to system-independent value.

Return type
Any

Returns
the value of the first found aspect that is matching the passed aspect type and optionally also the passed system and direction ids or None if not found.

del_aspect(*type_or_key*, *system=''*, *direction=''*)

remove aspect from this field.

Parameters

- **type_or_key** (str) – either FAT_* type or full key (including already the system and direction)-
- **system** (str) – system id string (if type_or_key is a pure FAT_* constant).
- **direction** (str) – direction string FAD_* constant (if type_or_key is a pure FAT_* constant).

Return type
Any

Returns
the aspect value of the removed aspect.

set_aspect(*aspect_value*, *type_or_key*, *system=''*, *direction=''*, *protect=False*, *allow_values=False*)

set/change the value of an aspect identified by *type_or_key*, *system* and *direction*.

Parameters

- **aspect_value** (Any) – the value to set on the aspect.
- **type_or_key** (str) – either FAT_* type or full key (including already the system and direction)-
- **system** (str) – system id string (if type_or_key is a pure FAT_* constant).
- **direction** (str) – direction string FAD_* constant (if type_or_key is a pure FAT_* constant).

- **protect** (bool) – pass True to prevent overwrite of already existing/set aspect value.
- **allow_values** (bool) – pass True to allow change of field value aspect (*FAT_VAL* aspect key).

Return type
_Field

Returns
self (this *_Field* instance).

set_aspects(*allow_values=False*, ***aspects*)

set multiple aspects provided in *aspects*.

Parameters

- **allow_values** (bool) – pass True to allow change of field value aspect (*FAT_VAL* aspect key).
- **aspects** (Any) – dict of aspects where the dict key is the full/complete aspect key.

Return type
_Field

Returns
self (this *_Field* instance).

add_aspects(*allow_values=False*, ***aspects*)

add multiple aspects provided in *aspects*.

Parameters

- **allow_values** (bool) – pass True to allow change of field value aspect (*FAT_VAL* aspect key).
- **aspects** (Any) – dict of aspects where the dict key is the full/complete aspect key.

Return type
_Field

Returns
self (this *_Field* instance).

name(*system=''*, *direction=''*, *flex_sys_dir=True*)

determine one of the names of this field.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.

Return type
str

Returns
main or system-specific name of this field.

del_name(*system*, *direction=''*)

remove system-specific name from this field.

Parameters

- **system** (str) – system id (has to be non-empty system id).
- **direction** (str) – direction id (def=” stands for both directions).

Return type
_Field

Returns
self (this *_Field* instance).

has_name(*name*, *selected_sys_dir=None*)

check if this field has a name identical to the one passed in *name*.

Parameters

- **name** (str) – name to search for.
- **selected_sys_dir** (Optional[dict]) – pass dict to get back the system and direction ids of the found name.

Return type
Optional[str]

Returns
full aspect key of the found name (including system/direction) or None if not found.

set_name(*name*, *system=''*, *direction=''*, *protect=False*)

set/change one of the names of this field.

Parameters

- **name** (str) – the new name of this field.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **protect** (bool) – pass True to prevent overwrite of already set/existing name.

Return type
_Field

Returns
self (this *_Field* instance).

root_rec(*system=''*, *direction=''*)

determine and return root record of this field with given system and direction ids.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Optional[*Record*]

Returns
root record instance or None if not set.

set_root_rec(*rec*, *system=''*, *direction=''*)

set/change the root record of this field, system and direction.

Parameters

- **rec** (*Record*) – root record instance.

- **system**¶ (str) – system id (def=” stands for the main/system-independent value).
- **direction**¶ (str) – direction id (def=” stands for the main/system-independent value).

Return type
_Field

Returns
self (this *_Field* instance).

root_idx(*system=''*, *direction=''*)

return the root index of this field for the specified *system* and *direction* ids.

Parameters

- **system**¶ (str) – system id (def=” stands for the main/system-independent value).
- **direction**¶ (str) – direction id (def=” stands for the main/system-independent value).

Return type
Tuple[Union[int, str], ...]

Returns
root index of this field.

set_root_idx(*idx_path*, *system=''*, *direction=''*)

set/change the root record of this field, system and direction.

Parameters

- **idx_path**¶ (Tuple[Union[int, str], ...]) – root index for this field/system/direction.
- **system**¶ (str) – system id (def=” stands for the main/system-independent value).
- **direction**¶ (str) – direction id (def=” stands for the main/system-independent value).

Return type
_Field

Returns
self (this *_Field* instance).

calculator(*system=''*, *direction=''*)

return the calculation callable for this field, *system* and *direction*.

Parameters

- **system**¶ (str) – system id (def=” stands for the main/system-independent value).
- **direction**¶ (str) – direction id (def=” stands for the main/system-independent value).

Return type
Optional[Callable[[*_Field*], Any]]

Returns
callable used to calculate the value of this field or None if not set/exists/found.

set_calculator(*calculator*, *system=''*, *direction=''*, *protect=False*)

set/change the field value calculator of this field, system and direction.

Parameters

- **calculator**¶ (Callable[[*_Field*], Any]) – new callable used to calculate the value of this field.
- **system**¶ (str) – system id (def=” stands for the main/system-independent value).

- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **protect** (bool) – pass True to prevent overwrite of already set/existing calculator callable.

Return type
_Field

Returns
self (this *_Field* instance).

clear_val(*system=''*, *direction=''*)

return the initial field value (the clear value) of this field, *system* and *direction*.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Any

Returns
the found clear/init value or None if not set/found.

set_clear_val(*clr_val*, *system=''*, *direction=''*)

set/change the clear/init value of this field, *system* and *direction*.

Parameters

- **clr_val** – new clear/init value of this field.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
_Field

Returns
self (this *_Field* instance).

_ensure_system_value(*system*, *direction=''*, *root_rec=None*, *root_idx=()*)

check if a field value for the specified *system*/*direction* exists and if not then create it.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **root_rec** (Optional[*Record*]) – root Record instance of this field, system and direction.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/*_Field* instance.

converter(*system*, *direction=''*)

return the converter callable for this field, *system* and *direction*.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Optional[Callable[[*_Field*, Any], Any]]

Returns
callable used to convert the field value between systems or None if not set.

separate system-specific representations of the field value can be (automatically) converted by specifying a converter callable aspect.

set_converter(*converter*, *system*, *direction=''*, *protect=True*, *root_rec=None*, *root_idx=()*)

set/change the field value converter of this field, system and direction.

Parameters

- **converter** (Callable[[*_Field*, Any], Any]) – new callable used to convert the value of this field between systems.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **protect** (bool) – pass False to allow the overwriting of an already set converter callable.
- **root_rec** (Optional[*Record*]) – root Record instance of this field, system and direction.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/*_Field* instance.

Return type
_Field

Returns
self (this *_Field* instance).

convert(*val*, *system*, *direction*)

convert field value from/onto system.

Parameters

- **val** (Any) – field value to convert.
- **system** (str) – system to convert from/onto.
- **direction** (str) – conversion direction (from or onto - see *FAD_FROM* and *FAD_ONTO*).

Return type
Any

Returns
converted field value.

filterer(*system=''*, *direction=''*)

return the filter callable for this field, *system* and *direction*.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Optional[Callable[[*_Field*], Any]]

Returns
callable used to filter this field/parent-record or None if not set.

set_filterer(*filterer*, *system=''*, *direction=''*, *protect=False*)

set/change the filterer callable of this field, system and direction.

Parameters

- **filterer** (Callable[[*_Field*], Any]) – new callable used to filter this field or the parent record.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **protect** (bool) – pass True to prevent overwrite of already set/existing filter callable.

Return type
_Field

Returns
self (this *_Field* instance).

sql_expression(*system=''*, *direction=''*)

return the sql column expression for this field, *system* and *direction*.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Optional[str]

Returns
sql column expression string if set or None if not set.

set_sql_expression(*sql_expression*, *system=''*, *direction=''*, *protect=False*)

set/change sql column expression of this field, system and direction.

Parameters

- **sql_expression** (str) – new sql column expression used to fetch associated db column of this field.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **protect** (bool) – pass True to prevent overwrite of already set/existing filter callable.

Return type
_Field

Returns
self (this *_Field* instance).

validator(*system=''*, *direction=''*)

return the validation callable for this field, *system* and *direction*.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Optional[Callable[[*_Field*, Any], Any]]

Returns
validation callable if set or None if not set.

set_validator(*validator*, *system=''*, *direction=''*, *protect=False*, *root_rec=None*, *root_idx=()*)
set/change the field value validator of this field, system and direction.

Parameters

- **validator** (Callable[[*_Field*, Any], Any]) – new callable used to validate the value of this field, systems and direction.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **protect** (bool) – pass False to allow the overwriting of an already set converter callable.
- **root_rec** (Optional[*Record*]) – root Record instance of this field, system and direction.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/*_Field* instance.

Return type
_Field

Returns
self (this *_Field* instance).

validate(*val*, *system=''*, *direction=''*)
validate field value for specified *system* and *direction*.

Parameters

- **val** (Any) – field value to validate (if ok to be set as new field value).
- **system** (str) – system id of new field value.
- **direction** (str) – direction id of new field value.

Return type
bool

Returns
True if *val* is ok to be set as new field value else False.

append_record(*system=''*, *direction=''*, *flex_sys_dir=True*, *root_rec=None*, *root_idx=()*)
append new record to the *Records* value of this field/system/direction.

Parameters

- **system** (str) – system id of the field value to extend.
- **direction** (str) – direction id of the field value to extend.
- **flex_sys_dir** (bool) – pass False to prevent fallback to system-independent value.
- **root_rec** (Optional[*Record*]) – root Record instance of this field, system and direction.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/*_Field* instance.

Return type
Record

Returns
added/appended *Record* instance.

clear_leaves(*system=''*, *direction=''*, *flex_sys_dir=True*, *reset_lists=True*)

clear/reset field values and if reset_lists == True also Records/Values lists to one item.

Parameters

- **system** (str) – system of the field value to clear, pass None to clear all field values.
- **direction** (str) – direction of the field value to clear.
- **flex_sys_dir** (bool) – if True then also clear field value if system is given and field has no system value.
- **reset_lists** (bool) – if True/def then also clear Records/lists to one item.

Return type
_Field

Returns
self (this _Field instance).

copy(*deepness=0*, *root_rec=None*, *root_idx=()*, ***kwargs*)

copy the aspects (names, indexes, values, ...) of this field.

Parameters

- **deepness** – deep copy level: <0==see deeper(), 0==only copy current instance, >0==deep copy to deepness value - _Field occupies two deepness: 1st=_Field, 2nd=Value.
- **root_rec** (Optional[*Record*]) – destination root record.
- **root_idx** (Tuple[Union[int, str], ...]) – destination index path (tuple of field names and/or list/Records/Values indexes).
- **kwargs** – additional arguments (will be passed on - most of them used by *Record.copy()*).

Return type
_Field

Returns
new/copied *_Field* instance.

parent(*system=''*, *direction=''*, *value_types=None*)

determine one of the parent ValueType instances in this data structure above of this field.

Parameters

- **system** (str) – system id (def='' stands for the main/system-independent value).
- **direction** (str) – direction id (def='' stands for the main/system-independent value).
- **value_types** (Optional[Tuple[Type[Union[*Value*, *Values*, *Record*, *Records*]], ...]]) – pass tuple of *ValueType* to restrict search to one of the passed types.

Return type
Union[*Value*, *Values*, *Record*, *Records*, None]

Returns
found parent instance or None if not set or if type is not of passed *value_types*.

pull(*from_system*, *root_rec*, *root_idx*)

pull the system-specific value (specified by *from_system*) into the main value of this field.

Parameters

- **from_system** (str) – system id of the system to pull from.
- **root_rec** (*Record*) – root Record instance of this field, system and direction.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/*_Field* instance.

Return type
_Field

Returns
self (this _Field instance).

push(*onto_system*, *root_rec*, *root_idx*)

push the main value of this field onto the system-specific value (specified by *onto_system*).

Parameters

- **onto_system** (str) – system id of the system to pull from.
- **root_rec** (*Record*) – root Record instance of this field, system and direction.
- **root_idx** (Tuple[Union[int, str], ...]) – root index to this node/*_Field* instance.

Return type
_Field

Returns
self (this _Field instance).

string_to_records(*str_val*, *field_names*, *rec_sep=','*, *fld_sep='='*, *system=''*, *direction=''*)

convert formatted string into a *Records* instance containing several *Record* instances.

Parameters

- **str_val** (str) – formatted string to convert.
- **field_names** (Sequence) – list/tuple of field names of each record
- **rec_sep** (str) – character(s) used in *str_val* to separate the records.
- **fld_sep** (str) – character(s) used in *str_val* to separate the field values of each record.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Records

Returns
converted *Records* instance.

record_field_val(**idx_path*, *system=''*, *direction=''*)

get/determine the value of any field specified via *idx_path* within this data structure.

Parameters

- **idx_path** (Union[int, str]) – index path of the field.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Any

Returns
the field value if found else None.

this method has an alias named *rfv()*.

rfv(**idx_path*, *system=''*, *direction=''*)

alias of method *record_field_val()*

Return type
Any

system_record_val(**idx_path*, *system=''*, *direction=''*, *use_curr_idx=None*)

get/determine the current value of a/this field within this data structure.

Parameters

- **idx_path** (Union[int, str]) – index path of the field.
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).
- **use_curr_idx** (Optional[list]) – list of counters to specify if and which current indexes have to be used.

Return type
Any

Returns
the currently selected field value if found else None.

this method has an alias named *srv()*.

srv(**idx_path*, *system=''*, *direction=''*, *use_curr_idx=None*)

alias of method *system_record_val()*

Return type
Any

in_actions(**actions*, *system=''*, *direction=''*)

determine if current data structure is in one of the passed *actions*.

Parameters

- **actions** (str) – tuple of actions (see ACTION_ constants).
- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
bool

Returns
True if the data structure has set one of the passed *actions* else False.

this method has an alias named *ina()*.

ina(**actions*, *system=''*, *direction=''*)

alias of method *in_actions()*

Return type
bool

current_records_idx(*system=''*, *direction=''*)

determine current index of *Records* instance, situated above of this field in this data structure.

Parameters

- **system** (str) – system id (def=” stands for the main/system-independent value).
- **direction** (str) – direction id (def=” stands for the main/system-independent value).

Return type
Union[int, str, None]

Returns
full index path or None if no current index exists above this field.

this method has an alias named *crx()*.

crx(*system=''*, *direction=''*)

alias of method *current_records_idx()*

Return type
Union[int, str, None]

VALUE_TYPES = (<class 'ae.sys_data.Value'>, <class 'ae.sys_data.Values'>, <class 'ae.sys_data.Record'>, <class 'ae.sys_data.Records'>)

tuple of classes/types used for system data values

LIST_TYPES = (<class 'ae.sys_data.Values'>, <class 'ae.sys_data.Records'>)

tuple of classes/types used for system data lists

NODE_TYPES = (<class 'ae.sys_data.Record'>, <class 'ae.sys_data.Records'>)

tuple of classes/types used for system data nodes

NODE_CHILD_TYPES = (<class 'ae.sys_data._Field'>, <class 'ae.sys_data.Record'>)

tuple of classes/types used for system data node children

PARENT_TYPES = (<class 'ae.sys_data.Values'>, <class 'ae.sys_data.Record'>, <class 'ae.sys_data.Records'>)

value types that can have children (Value excluded)

4.20 ae.sys_core_sh

4.20.1 SiHOT PMS system core xml interface

This portion is very old and needs refactoring and much more unit tests.

The classes provided by this portion are allowing the implementation of client and server components to communicate with the Sihot PMS system.

TODO:

- use other xml library because xml.etree the xml modules in the Python standard library are not secure against maliciously constructed data - the problem here is that the xml generated by the Sihot system is not 100% conform to the xml standards.
- refactor SihotXmlParser and inherited classes: migrating the attributes oc, tn, id, rc, hn, ... to a dict.
- inject cae app instance into _SihotTcpClient, RequestXmlHandler and TcpServer (replacing ae.core.po()).

Module Attributes

SDI_SH	Sihot System Interface Id
SH_DEF_SEARCH_FIELD	default search field for external systems (used by sys_data_sh.cl_field_data())
SDF_SH_SERVER_ADDRESS	Sihot Server address or ip
SDF_SH_KERNEL_PORT	Sihot Kernel Interface port
SDF_SH_WEB_PORT	Sihot Web interfaces port
SDF_SH_CLIENT_PORT	Sihot Server client port
SXML_DEF_ENCODING	encoding used by the Sihot xml interface
ERR_MESSAGE_PREFIX_CONTINUE	error message prefix for ignorable errors
TCP_CONNECTION_BROKEN_MSG	error message fragment added if connection is broken

Functions

elem_to_attr(elem)	convert element string to attribute string by converting into lower-case and replacing hyphens with underscores.

Classes

AvailCatInfo(cae[, use_kernel])	build xml request and send it to get available room categories from the Sihot server.
AvailCatInfoResponse(cae)	processing response of CATINFO operation code of the WEB interface
CatRoomResponse(cae)	parser for Sihot room category responses.
CatRooms(cae[, use_kernel])	built room category request and send it to Sihot server.
ConfigDict(cae[, use_kernel])	build and send request for the Sihot configuration settings.
ConfigDictResponse(cae)	parser to process Sihot configuration setting responses.
PostMessage(cae[, use_kernel])	build and send request to add a message into the Sihot system message/notification queue.
Request(cae)	xml parser for generic requests from SIHOT.
RequestXmlHandler(request, client_address, ...)	server component base class to receive xml data from the Sihot system.
ResChange(cae)	xml parser to process Sihot server reservation change notifications.
ResKernelGet(cae)	build and send generic request to the Sihot kernel interface.
ResKernelResponse(cae)	response to the RESERVATION-GET oc/request of the KERNEL interface.
ResResponse(cae)	xml parser for kernel or web interface responses.
RoomChange(cae)	xml parser to process Sihot server room change notifications.
ShSysConnector(system)	connector class for the Sihot system.
SihotXmlBuilder(cae[, use_kernel])	generic class to build and send Sihot xml requests.
SihotXmlParser(cae)	XMLParser interface used by client to parse the responses from the Sihot server.
TcpServer(ip, port, cls_xml_handler[, ...])	tcp server class to process xml sent by the Sihot client.

SDI_SH = 'Sh'

Sihot System Interface Id

SH_DEF_SEARCH_FIELD = 'ShId'

default search field for external systems (used by sys_data_sh.cl_field_data())

SDF_SH_SERVER_ADDRESS = 'shServerIP'

Sihot Server address or ip

SDF_SH_KERNEL_PORT = 'shServerKernelPort'

Sihot Kernel Interface port

SDF_SH_WEB_PORT = 'shServerPort'

Sihot Web interfaces port

SDF_SH_CLIENT_PORT = 'shClientPort'

Sihot Server client port

SXML_DEF_ENCODING = 'cp1252'

encoding used by the Sihot xml interface

ERR_MESSAGE_PREFIX_CONTINUE = 'CONTINUE:'

error message prefix for ignorable errors

TCP_CONNECTION_BROKEN_MSG = 'socket connection broken!'

error message fragment added if connection is broken

_TCP_MAXBUFLEN = 8192

tcp buffer length

_TCP_END_OF_MSG_CHAR = b'\x04'

end-of-message character of the Sihot xml interface

_DEBUG_RUNNING_CHARS = '|/-\\'

progress animation characters for console output

elem_to_attr(*elem*)

convert element string to attribute string by converting into lower-case and replacing hyphens with underscores.

Parameters
elem (str) – element string.

Return type
str

Returns
attribute string of the passed element string.

class _SihotTcpClient(*server_ip*, *server_port*, *timeout=3.6*, *encoding='utf-8'*, *debug_level=0*)

Bases: object

local sihot tcp client used by *SihotXmlBuilder.send_to_server()*.

error_message = ''

received_xml = ''

__init__(*server_ip*, *server_port*, *timeout=3.6*, *encoding='utf-8'*, *debug_level=0*)

send_to_server(*xml*)

send passed xml string to the Sihot server.

Parameters
xml (str) – xml string to send.

Return type
str

Returns
error message or empty string if no errors occurred.

_receive_response(*sock*)

receive response from Sihot server.

Parameters
sock – used socket for the connection to the Sihot server.

Return type
str

Returns
received string.

class RequestXmlHandler(*request*, *client_address*, *server*)

Bases: BaseRequestHandler

server component base class to receive xml data from the Sihot system.

error_message = ''

notify()

print error message to console output.

handle()

receive xml string sent by the Sihot system to this server component.

abstract handle_xml(*xml_from_client*)

abstract method to be implemented by the inheriting class.

Parameters
xml_from_client (bytes) – xml request sent from Sihot xml client as a bytes string.

Return type
bytes

Returns
xml response to the Sihot xml client as bytes string.

class _ThreadedServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)

Bases: ThreadingMixIn, TCPServer

local tcp server threading mixin class used by *TcpServer* class.

class TcpServer(*ip*, *port*, *cls_xml_handler*, *debug_level=0*)

Bases: object

tcp server class to process xml sent by the Sihot client.

__init__(*ip*, *port*, *cls_xml_handler*, *debug_level=0*)

run(*display_animation=False*)

run/start the server.

Parameters
display_animation (bool) – pass True to display progress animation at console output.

class SihotXmlParser(*cae*)

Bases: object

XMLParser interface used by client to parse the responses from the Sihot server.

__init__(*cae*)

parse_xml(*xml*)

parse the xml response string sent by the Sihot server.

Parameters
xml (str) – xml string to parse.

get_xml()

get the xml string to be parsed.

Return type
str

Returns
xml string to be parsed.

start(*tag*, *_attrib*)

parse next opening xml tag; called for each opening tag.

Parameters

- **tag** (str) – tag string.
- **_attrib** (Dict[str, str]) – attribute string (not used in Sihot xml elements).

Return type
Optional[str]

Returns
None if the tag got parsed and recognized/processed else the unrecognized tag string.

data(*data*)

process parsed data string; called on each chunk (separated by XMLParser on spaces, special chars, ...).

Parameters
data (str) – data string chunk.

Return type
Optional[str]

Returns
None if data chunk string got parsed and recognized else the unprocessed data string.

end(*tag*)

parser detected end tag of element; called for each closing tag.

Parameters
tag (str) – closing element tag string.

Return type
Optional[str]

Returns
closing element tag string.

`close()`
end of xml string reached; called when all data has been parsed.

Return type
SihotXmlParser

Returns
this *SihotXmlParser* instance.

`server_error()`
get the server error code string.

Return type
str

Returns
'0' if no error occurred, else the Sihot error return code as string.

`server_err_msg()`
get the server error message.

Return type
str

Returns
empty string if no error occurred, else the Sihot error message string.

`class Request(cae)`
Bases: *SihotXmlParser*

xml parser for generic requests from SIHOT.

`get_operation_code()`
return the Sihot operation code of the xml request string sent to the Sihot server.

Return type
str

Returns
Sihot operation code string.

`class RoomChange(cae)`
Bases: *SihotXmlParser*

xml parser to process Sihot server room change notifications.

`__init__(cae)`

`class ResChange(cae)`
Bases: *SihotXmlParser*

xml parser to process Sihot server reservation change notifications.

`__init__(cae)`

`start(tag, attrib)`
parser detected start tag of next/new xml element; called for each opening tag.

Return type
None

data(*data*)

process parsed element data chunk.

Return type
None

class ResResponse(*cae*)

Bases: *SihotXmlParser*

xml parser for kernel or web interface responses.

__init__(*cae*)

class AvailCatInfoResponse(*cae*)

Bases: *SihotXmlParser*

processing response of CATINFO operation code of the WEB interface

__init__(*cae*)

data(*data*)

process parsed element data chunk.

Return type
Optional[str]

class CatRoomResponse(*cae*)

Bases: *SihotXmlParser*

parser for Sihot room category responses.

__init__(*cae*)

end(*tag*)

parser detected end tag of element.

class ConfigDictResponse(*cae*)

Bases: *SihotXmlParser*

parser to process Sihot configuration setting responses.

__init__(*cae*)

end(*tag*)

parser detected end tag of Sihot config element.

Return type
Optional[str]

class ResKernelResponse(*cae*)

Bases: *SihotXmlParser*

response to the RESERVATION-GET oc/request of the KERNEL interface.

__init__(*cae*)

class SihotXmlBuilder(*cae*, *use_kernel=False*)

Bases: object

generic class to build and send Sihot xml requests.

tn: *str* = '1'

__init__(*cae*, *use_kernel=False*)

create an instance of this class.

Parameters

- **cae** (*ConsoleApp*) – instance of the running *ConsoleApp* app.
- **use_kernel** (bool) – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

beg_xml(*operation_code*, *add_inner_xml=''*, *transaction_number=''*)

create a new xml request string including xml header, operation code and transaction number.

Parameters

- **operation_code** (str) – Sihot operation code of the new xml element.
- **add_inner_xml** (str) – inner xml block/elements.
- **transaction_number** (str) – Sihot transaction number string.

end_xml()

terminate a Sihot xml request string.

add_tag(*tag*, *val=''*)

add a new xml element tag with the passed value.

send_to_server(*response_parser=None*)

send the built xml request to the Sihot server.

Parameters

response_parser (Optional[*SihotXmlParser*]) – used parser to parse the response from the Sihot server (def=SihotXmlParser).

Return type

str

Returns

error string or empty string if no errors occurred.

static new_tag(*tag*, *val=''*, *opening=True*, *closing=True*)

create new xml element with the passed tag and value.

Parameters

- **tag** (str) – tag of the new xml element.
- **val** (Any) – value of the new xml element.
- **opening** (bool) – pass False to NOT add the element opening tag.
- **closing** (bool) – pass False to NOT add the element closing tag.

Returns

new xml element or element part/fragment.

static convert_value_to_xml_string(*value*)

convert any element value type to the corresponding xml string, replacing & < > characters with escapes.

Parameters

value (Any) – element value to be converted.

Return type
str

Returns
element value as string.

property xml: str

property to determine the currently built xml string.

Getter
return built xml string.

Setter
change xml string.

class AvailCatInfo(*cae*, *use_kernel=False*)

Bases: *SihotXmlBuilder*

build xml request and send it to get available room categories from the Sihot server.

avail_rooms(*hotel_id=''*, *room_cat=''*, *from_date=datetime.date(2024, 3, 19)*, *to_date=datetime.date(2024, 3, 19)*)

determine available rooms for the specified hotel, room category and date range.

Parameters

- **hotel_id** (str) – Sihot hotel id or empty string to get available rooms of all hotels.
- **room_cat** (str) – Sihot room category or empty string to get available rooms of all categories.
- **from_date** (date) – start date of the date range: defaulting to today.
- **to_date** (date) – end date of the date range; defaulting to today.

Return type
Union[dict, str]

Returns
Sihot response as dict created by *AvailCatInfoResponse* with the results or an error message string if an error occurred.

response: Optional[SihotXmlParser]

class CatRooms(*cae*, *use_kernel=False*)

Bases: *SihotXmlBuilder*

built room category request and send it to Sihot server.

get_cat_rooms(*hotel_id='1'*, *from_date=datetime.date(2024, 3, 19)*, *to_date=datetime.date(2024, 3, 19)*, *scope=''*)

determine configured room categories of the Sihot system for the specified hotel and date range.

Parameters

- **hotel_id** (str) – Sihot hotel id to get the room categories.
- **from_date** (date) – start date of the date range: defaulting to today.
- **to_date** (date) – end date of the date range; defaulting to today.
- **scope** (str) – scope string to request additional information - see Sihot documentation.

Return type
Union[dict, str]

Returns
Sihot response as dict created by *CatRoomResponse* with the results or an error message string if an error occurred.

response: Optional[*SihotXmlParser*]

class ConfigDict(*cae*, *use_kernel=False*)

Bases: *SihotXmlBuilder*

build and send request for the Sihot configuration settings.

get_key_values(*config_type*, *hotel_id='1'*, *language='EN'*)

determine a configuration setting of the Sihot system for the specified hotel and language.

Parameters

- **config_type** (str) – Sihot config setting type - see Sihot documentation.
- **hotel_id** (str) – Sihot hotel id to get the configuration setting.
- **language** (str) – language id for the configuration description texts - see Sihot documentation.

Return type
Union[dict, str]

Returns
Sihot response as dict created by *ConfigDictResponse* with the results or an error message string if an error occurred.

response: Optional[*SihotXmlParser*]

class PostMessage(*cae*, *use_kernel=False*)

Bases: *SihotXmlBuilder*

build and send request to add a message into the Sihot system message/notification queue.

post_message(*msg*, *level=3*, *system='sys_core_sh_module'*)

build and send request to add a message into the Sihot system message/notification queue.

Parameters

- **msg** (str) – message text string to add to the Sihot system messages.
- **level** (int) – importance level.
- **system** (str) – message sender system id/string.

Return type
str

Returns
error message or empty string if no error occurred.

response: Optional[*SihotXmlParser*]

class ResKernelGet(*cae*)

Bases: *SihotXmlBuilder*

build and send generic request to the Sihot kernel interface.

__init__(*cae*)

create an instance of this class.

Parameters

- **cae** (*ConsoleApp*) – instance of the running *ConsoleApp* app.
- **use_kernel** – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

fetch_res_no(*obj_id*, *scope='GET'*)

determine reservation and guest data for the passed reservation object id.

Parameters

- **obj_id** (str) – Sihot reservation object id.
- **scope** – search scope string (see 7.3.1.2 in Sihot KERNEL interface doc V 9.0).

Return type
Tuple[Any, ...]

Returns
either the reservation ids as tuple of (hotel_id, res_id, sub_id, gds_no) or the tuple (None, "error") if the reservation was not found.

response: Optional[*SihotXmlParser*]

class ShSysConnector(*system*)

Bases: *SystemConnectorBase*

connector class for the Sihot system.

connect()

not needed - lazy connection

Return type
str

static clients_match_field_init(*match_fields*)

check and return the match field for a client search.

Parameters
match_fields (Sequence) – tuple/list of length 1 with the match field name.

Return type
str

Returns
match field name or an error message if an error occurred.

last_err_msg: str

last system connection error message(s)

4.21 ae.sys_data_sh

4.21.1 Sihot system data interface

This portion is very old and needs refactoring and much more unit tests.

The sys record data classes provided by this portion are based on client and server components (to communicate with the Sihot PMS system) provided by the *ae.sys_core_sh* portion of the ae namespace package.

TODO:

- split into several small modules.
- enhance type annotation.
- add much more unit tests.

Module Attributes

SH_CLIENT_MAP	default map for ClientFromSihot.elem_fld_map instance.
SH_RES_MAP	reservation map

Functions

add_sh_options(cae[, client_port, ...])	add config/command line options.
client_data(cae, obj_id)	fetch the client data record from Sihot for the client specified by its Sihot object id.
complete_res_data(rec)	complete reservation data row (rec) with default values.
convert_date_from_sh(xml_string)	convert date string sent by the Sihot system into a datetime.date value.
convert_date_onto_sh(date)	convert datetime.date value into a date string in the format of the Sihot system.
date_range_chunks(date_from, date_till, ...)	split date range into manageable chunks respecting the passed maximum days length and yield them back.
elem_path_join(elem_names)	convert list of element names to element path.
gds_no_to_ids(cae, hotel_id, gds_no)	determine all reservation ids for a reservation with the passed GDS number.
gds_no_to_obj_id(cae, hotel_id, gds_no)	determine the Sihot object id of a reservation identified by the GDS number.
hotel_and_res_id(shd)	determine hotel and reservation ids.
obj_id_to_res_no(cae, obj_id)	using RESERVATION-GET oc from KERNEL interface (see 7.3 in SIHOT KERNEL interface doc).
pax_count(shd)	determine the number of PAX of the passed reservation record.
print_sh_options(cae)	print the current config options values to the console.
res_no_to_ids(cae, hotel_id, res_id, sub_id)	determine all reservation ids for a reservation with the passed reservation number.
res_no_to_obj_id(cae, hotel_id, res_id, sub_id)	determine the Sihot object id of a reservation identified by their reservation number.
res_search(cae, date_from[, date_till, ...])	search reservations with the criteria specified by the parameters.

Classes

BulkFetcherBase(cae[, add_kernel_port])	helper base class for bulk data fetches.
ClientFetch(cae)	build xml and send to the Sihot system to fetch a client record.
ClientFromSihot(cae[, elem_map])	extended xml parser converting client data directly into a sys data Record structure.
ClientSearch(cae)	search client.
ClientToSihot(cae)	extended xml builder class to send/push client data to Sihot.
FldMapXmlBuilder(cae[, use_kernel, elem_map])	extended xml builder base class.
FldMapXmlParser(cae, elem_map)	extended xml parser converting data directly into a sys data Record structure.
GuestBulkFetcher(cae[, add_kernel_port])	WIP/NotUsed/NoTests: the problem is with GUEST-SEARCH is that there is no way to bulk fetch all guests because the search criteria is not providing range search to split in slices.
ResBulkFetcher(cae[, allow_future_arrivals])	reservation bulk fetch.
ResFetch(cae[, use_kernel])	fetch reservation.
ResFromSihot(cae)	extended xml parser converting reservation data directly into a sys data Record structure.
ResSearch(cae[, use_kernel])	search reservation.
ResSender(cae)	helper class to send reservation records to Sihot.
ResToSihot(cae)	extended xml builder to send reservation data to Sihot.

convert_date_from_sh(*xml_string*)

convert date string sent by the Sihot system into a datetime.date value.

Return type
date

convert_date_onto_sh(*date*)

convert datetime.date value into a date string in the format of the Sihot system.

Return type
str

SH_CLIENT_MAP = (('OBJID', 'ShId', None, <function <lambda>>), ('MATCHCODE', 'AcuId'), ('T-SALUTATION', 'Salutation'), ('T-TITLE', 'Title'), ('T-GUEST', 'GuestType', '1'), ('NAME-1', 'Surname'), ('NAME-2', 'Forename'), ('STREET', 'Street'), ('PO-BOX', 'POBox'), ('ZIP', 'Postal'), ('CITY', 'City'), ('T-COUNTRY-CODE', 'Country'), ('T-STATE', 'State', None, <function <lambda>>), ('T-LANGUAGE', 'Language'), ('T-NATION', 'Nationality', None, <function <lambda>>), ('COMMENT', 'Comment'), ('COMMUNICATION/', None, None, <function <lambda>>), ('PHONE-1', 'Phone'), ('PHONE-2', 'WorkPhone'), ('FAX-1', 'Fax'), ('EMAIL-1', 'Email'), ('EMAIL-2', 'EmailB'), ('MOBIL-1', 'MobilePhone'), ('MOBIL-2', 'MobilePhoneB'), ('/COMMUNICATION', None, None, <function <lambda>>), ('ADD-DATA/', None, None, <function <lambda>>), ('T-PERSON-GROUP', None, '1A'), ('D-BIRTHDAY', 'DOB', None, None, <function <lambda>>, <function <lambda>>), ('INTERNET-PASSWORD', 'Password'), ('MATCH-ADM', 'RciId'), ('MATCH-SM', 'SfId'), ('/ADD-DATA', None, None, <function <lambda>>))

default map for ClientFromSihot.elem_fld_map instance. as read-only constant by AcuClientToSihot using the SIHOT KERNEL interface because SiHOT WEB V9 has missing fields: initials (CD_INIT1/2) and profession (CD_INDUSTRY1/2).

```
SH_RES_MAP = (('SIHOT-Document.ID', 'ResHotelId'), ('ARESLIST/',), ('RESERVATION/',),
('RESERVATION.RES-HOTEL', 'ResHotelId'), ('RESERVATION.RES-NR', 'ResId', None, <function
<lambda>>), ('RESERVATION.SUB-NR', 'ResSubId', None, <function <lambda>>),
('RESERVATION.OBJID', 'ResObjId', None, <function <lambda>>), ('GDSNO', 'ResGdsNo'),
('RESERVATION.GUEST-ID', 'ShId'), ('RESERVATION.GUEST-OBJID', 'ShId'),
('RESERVATION.MATCHCODE', 'AcuId'), ('RESERVATION.NAME', 'Surname'), ('VOUCHERNUMBER',
'ResVoucherNo', None, <function <lambda>>), ('EXT-KEY', 'ResGroupNo', None, <function
<lambda>>), ('FLAGS', None, 'IGNORE-OVERBOOKING'), ('RT', 'ResStatus'), ('CAT',
'ResRoomCat'), ('PCAT', 'ResPriceCat', None, <function <lambda>>), ('ALLOTMENT-EXT-NO',
'ResAllotmentNo', '', <function <lambda>>), ('PAYMENT-INST', 'ResAccount', None,
<function <lambda>>), ('SALES-DATE', 'ResBooked', None, <function <lambda>>, <function
<lambda>>, <function <lambda>>), ('RATE-SEGMENT', 'ResRateSegment', None, <function
<lambda>>, ''), ('RATE/', None, None, <function <lambda>>), ('RATE.R', 'ResBoard', None,
<function <lambda>>), ('RATE.ISDEFAULT', None, 'Y', <function <lambda>>), ('/RATE', None,
None, <function <lambda>>), ('RATE/', None, None, <function <lambda>>), ('RATE.R',
'ResRateBoard', None, <function <lambda>>), ('RATE.ISDEFAULT', None, 'Y', <function
<lambda>>), ('RATE.DAYS/', None, None, <function <lambda>>), ('RATE.DAYS.D', ('ResRates',
0, 'RateDay'), None, <function <lambda>>, <function <lambda>>, <function <lambda>>),
('RATE.DAYS.PRICE', ('ResRates', 0, 'RateAmount'), None, <function <lambda>>),
('RATE./DAYS', None, None, <function <lambda>>), ('/RATE', None, None, <function
<lambda>>), ('RATE/', None, None, <function <lambda>>), ('RATE.R', None, 'GSC', <function
<lambda>>), ('RATE.ISDEFAULT', None, 'N', <function <lambda>>), ('/RATE', None, None,
<function <lambda>>), ('RESCHANNELLIST/', None, None, <function <lambda>>),
('RESCHANNEL/', None, None, <function <lambda>>), ('RESCHANNEL.IDX', None, 1, <function
<lambda>>), ('RESCHANNEL.MATCHCODE', None, 'RCI', <function <lambda>>),
('RESCHANNEL.ISPRICEOWNER', None, 1, <function <lambda>>), ('RESCHANNEL.IDX', None, 1,
<function <lambda>>), ('RESCHANNEL.MATCHCODE', None, 'MAR01', <function <lambda>>),
('RESCHANNEL.ISPRICEOWNER', None, 1, <function <lambda>>), ('RESCHANNEL.IDX', None, 2,
<function <lambda>>), ('RESCHANNEL.MATCHCODE', None, 'TSP', <function <lambda>>),
('RESCHANNEL.ISPRICEOWNER', None, 1, <function <lambda>>), ('/RESCHANNEL', None, None,
<function <lambda>>), ('/RESCHANNELLIST', None, None, <function <lambda>>), ('ARR',
'ResArrival', None, None, <function <lambda>>, <function <lambda>>), ('DEP',
'ResDeparture', None, None, <function <lambda>>, <function <lambda>>), ('NOROOMS', None,
1), ('NOPAX', 'ResAdults', None, None, <function <lambda>>, <function <lambda>>),
('NOCHILDS', 'ResChildren', None, <function <lambda>>, <function <lambda>>, <function
<lambda>>), ('TEC-COMMENT', 'ResLongNote', None, <function <lambda>>), ('COMMENT',
'ResNote', None, <function <lambda>>), ('MARKETCODE-NO', 'ResMktSegment', None, <function
<lambda>>), ('MARKETCODE', 'ResMktSegment'), ('SOURCE', 'ResSource', None, <function
<lambda>>), ('NN', 'ResMktGroupNN', None, <function <lambda>>), ('CHANNEL',
'ResMktGroup', None, <function <lambda>>), ('EXT-REFERENCE', 'ResFlightArrComment', None,
<function <lambda>>), ('ARR-TIME', 'ResFlightETA'), ('PICKUP-TIME-ARRIVAL',
'ResFlightETA', None, <function <lambda>>), ('PICKUP-TYPE-ARRIVAL', None, 1, <function
<lambda>>), ('PERSON/', None, None, <function <lambda>>), ('PERSON.GUEST-ID',
('ResPersons', 0, 'PersShId'), None, <function <lambda>>), ('PERSON.MATCHCODE',
('ResPersons', 0, 'PersAcuId'), None, <function <lambda>>), ('PERSON.NAME',
('ResPersons', 0, 'PersSurname'), None, <function <lambda>>), ('PERSON.NAME2',
('ResPersons', 0, 'PersForename'), None, <function <lambda>>), ('PERSON.AUTO-GENERATED',
('ResPersons', 0, 'AutoGen'), '1', <function <lambda>>), ('PERSON.ROOM-SEQ',
('ResPersons', 0, 'RoomSeq'), None, <function <lambda>>), ('PERSON.ROOM-PERS-SEQ',
('ResPersons', 0, 'RoomPersSeq'), None, <function <lambda>>), ('PERSON.PERS-TYPE',
('ResPersons', 0, 'TypeOfPerson'), None, <function <lambda>>), ('PERSON.RN',
('ResPersons', 0, 'RoomNo'), None, <function <lambda>>), ('PERSON.DOB', ('ResPersons', 0,
'PersDOB'), None, <function <lambda>>, <function <lambda>>, <function <lambda>>),
('PERSON.COUNTRY-CODE', ('ResPersons', 0, 'PersCountry'), None, <function <lambda>>),
('PERSON.EMAIL', ('ResPersons', 0, 'PersEmail'), None, <function <lambda>>),
('PERSON.LANG', ('ResPersons', 0, 'PersLanguage'), None, <function <lambda>>),
('PERSON.PHONE', ('ResPersons', 0, 'PersPhone'), None, <function <lambda>>),
('PERSON.PERS-RATE.R', ('ResPersons', 0, 'Board'), None, <function <lambda>>),
('PERSON.PICKUP-COMMENT-ARRIVAL', ('ResPersons', 0, 'FlightArrComment'), None, <function
<lambda>>), ('PERSON.PICKUP-TIME-ARRIVAL', ('ResPersons', 0, 'FlightETA'), None,
```

reservation map

add_sh_options(*cae*, *client_port=0*, *add_kernel_port=False*, *add_maps_and_kernel_usage=False*)

add config/command line options.

Parameters

- **cae** (*ConsoleApp*) – app instance.
- **client_port** (int) – client port.
- **add_kernel_port** (bool) – pass True to add also port option for the kernel interface.
- **add_maps_and_kernel_usage** (bool) – pass True to add also options for kernel switch and maps.

print_sh_options(*cae*)

print the current config options values to the console.

client_data(*cae*, *obj_id*)

fetch the client data record from Sihot for the client specified by its Sihot object id.

Parameters

- **cae** (*ConsoleApp*) – app instance.
- **obj_id** (str) – Sihot client object id.

Return type
Record

Returns
client record data structure.

complete_res_data(*rec*)

complete reservation data row (rec) with default values.

Parameters
rec (*Record*) – reservation data Record instance.

Return type
Record

Returns
completed reservation data Record instance.

un-changed fields:
ResRoomNo, ResNote, ResLongNote, ResFlightArrComment (flight no...), ResAllotmentNo, ResVoucherNo.

mandatory fields:
ShId or AcuId or Surname (to specify the orderer of the reservation), ResHotelId, ResArrival, ResDeparture, ResRoomCat, ResMktSegment, ResGdsNo, ResAdults, ResChildren.

optional fields:
ResPersons0PersSurname and ResPersons0PersForename (surname and forename) ResPersons1PersSurname and ResPersons1PersForename (...)

optional auto-populated fields:
see the default values - specified in default_values dict underneath.

elem_path_join(*elem_names*)

convert list of element names to element path.

Parameters
elem_names¶ (List[str]) – list of element names.

Return type
str

Returns
element path.

hotel_and_res_id(*shd*)

determine hotel and reservation ids.

Return type
Tuple[Optional[str], Optional[str]]

pax_count(*shd*)

determine the number of PAX of the passed reservation record.

Return type
int

date_range_chunks(*date_from*, *date_till*, *fetch_max_days*)

split date range into manageable chunks respecting the passed maximum days length and yield them back.

Return type
Iterator[Tuple[date, date]]

gds_no_to_ids(*cae*, *hotel_id*, *gds_no*)

determine all reservation ids for a reservation with the passed GDS number.

Return type
Dict[str, Any]

gds_no_to_obj_id(*cae*, *hotel_id*, *gds_no*)

determine the Sihot object id of a reservation identified by the GDS number.

Return type
str

res_no_to_ids(*cae*, *hotel_id*, *res_id*, *sub_id*)

determine all reservation ids for a reservation with the passed reservation number.

Return type
Union[str, Dict[str, Any]]

res_no_to_obj_id(*cae*, *hotel_id*, *res_id*, *sub_id*)

determine the Sihot object id of a reservation identified by their reservation number.

Return type
str

res_search(*cae*, *date_from*, *date_till=None*, *mkt_sources=None*, *mkt_groups=None*, *max_los=28*, *search_flags=''*, *search_scope=''*, *chunk_pause=1*)

search reservations with the criteria specified by the parameters.

Parameters

- **cae**¶ – instance of the application environment specifying searched Sihot server.

- **date_from** (date) – date of first day of included arrivals.
- **date_till** (Optional[date]) – date of last day of included arrivals.
- **mkt_sources** (Optional[List[str]]) – list of market source codes.
- **mkt_groups** (Optional[List[str]]) – list of market group codes.
- **max_los** (int) – integer with maximum length of stay.
- **search_flags** (str) – string with search flag words (separated with semicolon).
- **search_scope** (str) – string with search scope words (separated with semicolon).
- **chunk_pause** (int) – integer with seconds to pause between fetch of date range chunks.

Return type
Union[str, *Records*]

Returns
string with error message if error or Records/list of Sihot reservations (Record instances).

obj_id_to_res_no(*cae*, *obj_id*)

using RESERVATION-GET oc from KERNEL interface (see 7.3 in SIHOT KERNEL interface doc).

Parameters

- **cae** (*ConsoleApp*) – Console App Environment instance.
- **obj_id** (str) – Sihot Reservation Object Id.

Return type
tuple

Returns
reservation ids as tuple of (hotel_id, res_id, sub_id, gds_no) or (None, "error") if not found

_strip_err_msg(*error_msg*)

Return type
str

class FldMapXmlParser(*cae*, *elem_map*)

Bases: *SihotXmlParser*

extended xml parser converting data directly into a sys data Record structure.

__init__(*cae*, *elem_map*)

clear_rec()
clear the record data.

property rec: *Record*
return the record data.

start(*tag*, *attrib*)
process start of new xml element.

Return type
Optional[str]

data(*data*)
process data of xml element.

Return type
Optional[str]

end(*tag*)

process end of xml element.

Return type
Optional[str]

class ClientFromSihot(*cae*, *elem_map=(('OBJID', 'ShId', None, <function <lambda>>), ('MATCHCODE', 'AcuId'), ('T-SALUTATION', 'Salutation'), ('T-TITLE', 'Title'), ('T-GUEST', 'GuestType', '1'), ('NAME-1', 'Surname'), ('NAME-2', 'Forename'), ('STREET', 'Street'), ('PO-BOX', 'POBox'), ('ZIP', 'Postal'), ('CITY', 'City'), ('T-COUNTRY-CODE', 'Country'), ('T-STATE', 'State', None, <function <lambda>>), ('T-LANGUAGE', 'Language'), ('T-NATION', 'Nationality', None, <function <lambda>>), ('COMMENT', 'Comment'), ('COMMUNICATION/', None, None, <function <lambda>>), ('PHONE-1', 'Phone'), ('PHONE-2', 'WorkPhone'), ('FAX-1', 'Fax'), ('EMAIL-1', 'Email'), ('EMAIL-2', 'EmailB'), ('MOBIL-1', 'MobilePhone'), ('MOBIL-2', 'MobilePhoneB'), ('/COMMUNICATION', None, None, <function <lambda>>), ('ADD-DATA/', None, None, <function <lambda>>), ('T-PERSON-GROUP', None, '1A'), ('D-BIRTHDAY', 'DOB', None, None, <function <lambda>>, <function <lambda>>), ('INTERNET-PASSWORD', 'Password'), ('MATCH-ADM', 'RciId'), ('MATCH-SM', 'SfId'), ('/ADD-DATA', None, None, <function <lambda>>))*)

Bases: *FldMapXmlParser*

extended xml parser converting client data directly into a sys data Record structure.

__init__(*cae*, *elem_map=(('OBJID', 'ShId', None, <function <lambda>>), ('MATCHCODE', 'AcuId'), ('T-SALUTATION', 'Salutation'), ('T-TITLE', 'Title'), ('T-GUEST', 'GuestType', '1'), ('NAME-1', 'Surname'), ('NAME-2', 'Forename'), ('STREET', 'Street'), ('PO-BOX', 'POBox'), ('ZIP', 'Postal'), ('CITY', 'City'), ('T-COUNTRY-CODE', 'Country'), ('T-STATE', 'State', None, <function <lambda>>), ('T-LANGUAGE', 'Language'), ('T-NATION', 'Nationality', None, <function <lambda>>), ('COMMENT', 'Comment'), ('COMMUNICATION/', None, None, <function <lambda>>), ('PHONE-1', 'Phone'), ('PHONE-2', 'WorkPhone'), ('FAX-1', 'Fax'), ('EMAIL-1', 'Email'), ('EMAIL-2', 'EmailB'), ('MOBIL-1', 'MobilePhone'), ('MOBIL-2', 'MobilePhoneB'), ('/COMMUNICATION', None, None, <function <lambda>>), ('ADD-DATA/', None, None, <function <lambda>>), ('T-PERSON-GROUP', None, '1A'), ('D-BIRTHDAY', 'DOB', None, None, <function <lambda>>, <function <lambda>>), ('INTERNET-PASSWORD', 'Password'), ('MATCH-ADM', 'RciId'), ('MATCH-SM', 'SfId'), ('/ADD-DATA', None, None, <function <lambda>>))*)

end(*tag*)

process end of xml element.

Return type
Optional[str]

class ResFromSihot(*cae*)

Bases: *FldMapXmlParser*

extended xml parser converting reservation data directly into a sys data Record structure.

__init__(*cae*)

end(*tag*)

process end of xml element.

Return type
Optional[str]

class ClientFetch(*cae*)

Bases: *SihotXmlBuilder*

build xml and send to the Sihot system to fetch a client record.

__init__(*cae*)

create an instance of this class.

Parameters

- **cae**¶ (*ConsoleApp*) – instance of the running *ConsoleApp* app.
- **use_kernel**¶ – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

fetch_client(*obj_id*, *field_names=()*)

return Record with guest data OR str with error message in case of error.

Return type
Union[str, *Record*]

response: Optional[*SihotXmlParser*]

class ClientSearch(*cae*)

Bases: *SihotXmlBuilder*

search client.

__init__(*cae*)

create an instance of this class.

Parameters

- **cae**¶ – instance of the running *ConsoleApp* app.
- **use_kernel**¶ – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

search_clients(*matchcode=''*, *exact_matchcode=True*, *name=''*, *forename=''*, *surname=''*, *guest_no=''*, *email=''*, *guest_type=''*, *flags='FIND-ALSO-DELETED-GUESTS'*, *order_by=''*, *limit=0*, *field_names=('ShId',)*, ***kwargs*)

invoke the client search.

Return type
Union[str, list, *Records*]

client_id_by_matchcode(*matchcode*)

determine client id for a client identified by its match code.

Return type
Optional[str]

response: Optional[*SihotXmlParser*]

class ResFetch(*cae*, *use_kernel=False*)

Bases: *SihotXmlBuilder*

fetch reservation.

fetch_res(*ho_id*, *gds_no=''*, *res_id=''*, *sub_id=''*, *scope='USEISODATE'*)

invoke request and fetch of reservation data.

Return type
Union[str, *Record*]

fetch_by_gds_no(*ho_id*, *gds_no*, *scope='USEISODATE'*)

fetch reservation identified by their GDS number.

Return type
Union[str, *Record*]

fetch_by_res_id(*ho_id*, *res_id*, *sub_id*, *scope='USEISODATE'*)

fetch reservation identified by their reservation number and sub-number.

Return type
Union[str, *Record*]

response: Optional[*SihotXmlParser*]

class ResSearch(*cae*, *use_kernel=False*)

Bases: *SihotXmlBuilder*

search reservation.

search_res(*hotel_id=''*, *from_date=datetime.date(2024, 3, 19)*, *to_date=datetime.date(2024, 3, 19)*, *matchcode=''*, *name=''*, *gds_no=''*, *flags=''*, *scope=''*, *guest_id=''*)

invoke search of reservation.

Return type
Union[str, *Records*]

response: Optional[*SihotXmlParser*]

class FldMapXmlBuilder(*cae*, *use_kernel=False*, *elem_map=None*)

Bases: *SihotXmlBuilder*

extended xml builder base class.

__init__(*cae*, *use_kernel=False*, *elem_map=None*)

create an instance of this class.

Parameters

- **cae** (*ConsoleApp*) – instance of the running *ConsoleApp* app.
- **use_kernel** (bool) – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

get_warnings()

get warning messages as concatenated string.

Return type
str

wipe_warnings()

wipe all collected warning messages.

prepare_rec(*rec*)

prepare record to push to Sihot.

prepare_map_xml(*rec*, *include_empty_values=True*)

prepare and return xml string.

Return type
str

response: Optional[*SihotXmlParser*]

class ClientToSihot(*cae*)

Bases: *FldMapXmlBuilder*

extended xml builder class to send/push client data to Sihot.

__init__(*cae*)

create an instance of this class.

Parameters

- **cae**¶ – instance of the running *ConsoleApp* app.
- **use_kernel**¶ – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

static _complete_client_data(*rec*)

prepare_rec(*rec*)

prepare client record for the push to Sihot.

_prepare_guest_xml(*rec*, *fld_name_suffix=''*)

prepare extra guest data.

_prepare_guest_link_xml(*mc1*, *mc2*)

prepare guest links.

_send_link_to_sihot(*pk1*, *pk2*)

Return type
str

_send_person_to_sihot(*rec*, *first_person=''*)

send client data of one person to Sihot, passing AcuId of first person to send 2nd person.

Return type
str

send_client_to_sihot(*rec*)

send client data to Sihot.

Return type
str

response: Optional[*SihotXmlParser*]

_warning_msgs: List[str]

class ResToSihot(*cae*)

Bases: *FldMapXmlBuilder*

extended xml builder to send reservation data to Sihot.

__init__(*cae*)

create an instance of this class.

Parameters

- **cae** (*ConsoleApp*) – instance of the running *ConsoleApp* app.
- **use_kernel** – pass True to use the Sihot kernel interface (False==use Sihot sxml interface).

_add_sihot_configs(*rec*)

prepare_rec(*rec*)

prepare reservation data to be sent/pushed to Sihot system.

_prepare_res_xml(*rec*)

_sending_res_to_sihot(*rec*)

Return type
str

_handle_error(*rec*, *err_msg*)

Return type
str

_ensure_clients_exist_and_updated(*rec*, *ensure_client_mode*)

Return type
str

send_res_to_sihot(*rec*, *ensure_client_mode=0*)

send reservation to Sihot system.

Return type
str

static res_id_label()

reservation ids log and debug message label.

Return type
str

static res_id_values(*rec*)

reservation ids log and debug message data.

Return type
str

res_id_desc(*rec*, *err_msg*, *separator='\\n\\n'*, *indent=8*)

extended reservation ids log and debug message data.

Return type
str

wipe_gds_errors()

wipe collected gds errors.

response: Optional[*SihotXmlParser*]

_warning_msgs: List[str]

class BulkFetcherBase(*cae*, *add_kernel_port=True*)

Bases: object

helper base class for bulk data fetches.

__init__(*cae*, *add_kernel_port=True*)

add_options()

add command line option.

load_options()

load command line option.

print_options()

print command line option to console output.

class GuestBulkFetcher(*cae*, *add_kernel_port=True*)

Bases: *BulkFetcherBase*

WIP/NotUsed/NoTests: the problem is with GUEST-SEARCH is that there is no way to bulk fetch all guests because the search criteria is not providing range search to split in slices. Fetching all 600k clients is resulting in a timeout error after 30 minutes (see Sihot interface SDF_SH_TIMEOUT/'shTimeout' option value)

fetch_all()

fetch and return all found records.

Return type
Records

class ResBulkFetcher(*cae*, *allow_future_arrivals=True*)

Bases: *BulkFetcherBase*

reservation bulk fetch.

__init__(*cae*, *allow_future_arrivals=True*)

add_options()

add command line options.

load_options()

load command line options.

print_options()

print command line options.

date_range_str()

determine date range as string value.

Return type
str

fetch_all()

fetch bulk reservation data.

Return type
Records

class ResSender(*cae*)

Bases: *ResToSihot*

helper class to send reservation records to Sihot.

send_rec(*rec*)

send reservation record.

Return type
Tuple[str, str]

response: Optional[*SihotXmlParser*]

_warning_msgs: List[str]

_gds_errors: Dict[str, Tuple[*Record*, str]]

get_res_no()

determine reservation number of sent reservation.

Return type
tuple

4.22 ae.db_core

4.22.1 database connection and data manipulation base classes

This module is providing generic base classes and helper methods to implement and integrate any database driver that is compatible to the **Python DB API**. The base classes are providing already all attributes and methods to handle database connections, data selections and manipulations.

Additionally this package is defining generalized interfaces to switch dynamically between databases without the need to adapt the code of your application - simply by selecting another database connector class within your *configuration files*.

basic usage

The abstract base class *DbBase* allows you to implement a db-specific class with few lines of code. Simply inherit from *DbBase* and implement a *connect()* method that is setting the *conn* instance attribute to a **Python DB API** compatible connection object.

For a fictive specific database driver package *SpecificDbDriver* that is providing a *connect* method, a minimal example would look like:

```
from ae.db_core import DbBase
...
class SpecificDb(DbBase):
    def connect(self) -> str:
        self.last_err_msg = ''
        try:
           self.conn = SpecificDbDriver.connect(**self.connect_params())
        except Exception as ex:
            self.last_err_msg = f"SpecificDb-connect() error: {ex} for {self}"
        else:
```

(continues on next page)

(continued from previous page)

```
            self.create_cursor()
        return self.last_err_msg
```

The helper method *connect_params()* provided by this module is helping you to merge any specified database system credentials and features into a dict to be passed as connection parameters to the database-specific connection class/function/factory of the database driver.

Another helper *create_cursor()* provided by *DbBase* can be used in the *connect()* method to create a Python DB API-compatible cursor object and assign it to the *curs* attribute.

In this implementation your class SpecificDb does automatically also inherit useful methods from *DbBase* to execute *INSERT*, *DELETE*, *UPDATE* and *UPSERT* commands, to run *SELECT* queries and to call *stored procedures*.

Hint: Examples of an implementation of a specific database class are e.g. the portion packages *ae.db_pg* for Postgres and *ae.db_ora* for Oracle.

---

connection parameters

DbBase supports the following connection parameter keys:

- **user** : user name or database account name.
- **password** : user account password.
- **dbname** : name of the database to connect.
- **host** : host name or IP address of the database server.
- **port**: port number of the database server.

Some database drivers are using default values for most of these connection parameters, if they are not specified. The mandatory connection parameters can differ for different database drivers. Most of them need at least a user name and password.

Hint: Most database drivers like e.g. psycopg2 or pg8000 for Postgres databases are supporting all of these connection parameter keys. For others like e.g. cx_Oracle which are not supporting some of them (like e.g. *host* and *dbname*) you may also need to overwrite the *connect_params()* method to convert/adopt unsupported connection parameter keys.

Alternatively you can provide the connection parameters needed by the database driver connector with a single database url string in the SQLAlchemy Database URL format:

```
dialect+driver://user:password@host:port/dbname
```

So if your connection parameters dict - respective your database system credentials and features - providing a *url* key then *connect_params()* will parse and split the url into separate connection parameters.

Note: When you specify a connection parameter as separate key and also within the *url* value then the separate key value will have preference. The scheme part (dialect+driver) including the following colon character of the URL string are not evaluated and can be omitted.

---

bind variables format

Bind variables in your sql query strings have to be formatted in the **named parameter style**. If your database driver is only supporting the *pyformat* parameter style, then overload the __init__ method and set the *param_style* to `'pyformat'` and all sql query strings will be automatically converted by *DbBase* before they get sent to the database:

```
class SpecificDb(DbBase):
    def __init__(self, ...)
        super().__init__(...)
        self.param_style = 'pyformat'
    ...
```

Module Attributes

NAMED_BIND_VAR_PREFIX	character to mark a bind variable in a sql query
CHK_BIND_VAR_PREFIX	bind variable name prefix, to allow for same column/-name a new/separate value in e.g.

Functions

connect_args_from_params(conn_params)	split dict with database connection parameters into credentials and features.

Classes

DbBase(system)	abstract database connector base class for the *ae* namespace database system layers.

NAMED_BIND_VAR_PREFIX: str = ':'

character to mark a bind variable in a sql query

CHK_BIND_VAR_PREFIX: str = 'CV_'

bind variable name prefix, to allow for same column/-name a new/separate value in e.g. the SET clause and an old value in the WHERE clause. Gets added to bind variables in filters/chk_values and extra where clauses.

connect_args_from_params(*conn_params*)

split dict with database connection parameters into credentials and features.

Parameters
conn_params (Dict[str, Any]) – connection params dict.

Return type
Tuple[Dict[str, str], List[str]]

Returns
tuple of credentials dict and features list.

_normalize_col_values(*col_values*)

convert empty strings into real None values.

Parameters
col_values¶ (Dict[str, Any]) – dict of column values (dict key is the column name).

Return type
Dict[str, Any]

Returns
col_values dict where empty values got replaced with None (also in original dict).

_prepare_in_clause(*sql*, *bind_vars=None*, *additional_col_values=None*)

replace list bind variables used in an IN clause with separate bind variables for each list item.

Parameters

- **sql**¶ (str) – query to be executed.
- **bind_vars**¶ (Optional[Dict[str, Any]]) – dict of all available bind variables for the execution.
- **additional_col_values**¶ (Optional[Dict[str, Any]]) – additional bind variables.

Return type
Tuple[str, Dict[str, Any]]

Returns
tuple of adapted query string and joined bind variables.

_rebind(*chk_values=None*, *where_group_order=''*, *bind_vars=None*, *extra_bind=None*)

merge where_group_order string with chk_values filter dict and merge/rename bind variables.

Parameters

- **chk_values**¶ (Optional[Dict[str, Any]]) – dict with column_name: value items, used to filter/restrict the resulting rows.

  This method compiles this dict into a sql WHERE clause expression. If you also passed additional sql clauses into the *where_group_order* then it will be merged with the compiled expression. The names of the sql parameters and related bind variables are build from the column names/keys of this dict, and will be prefixed with *CHK_BIND_VAR_PREFIX*.

  Passing None or a empty dict to this and to the *extra_bind* arguments will disable any filtering. If only this argument is None/empty then the first item of the *extra_bind* dict will be used as filter.

- **where_group_order**¶ (str) – sql part with optional WHERE/GROUP/ORDER clauses (the part after the WHERE), including bind variables in the *named parameter style*.
- **bind_vars**¶ (Optional[Dict[str, Any]]) – dict with bind variables (variable name has to be prefixed in *where_group_order* with *CHK_BIND_VAR_PREFIX*).
- **extra_bind**¶ (Optional[Dict[str, Any]]) – additional dict with bind variables (variable name has NOT to be prefixed/adapted in *where_group_order* argument).

Return type
Tuple[Optional[Dict[str, Any]], str, Dict[str, Any]]

Returns
tuple of corrected/rebound values of chk_values, where_group_order and bind_vars.

class DbBase(*system*)

Bases: SystemConnectorBase, ABC

abstract database connector base class for the *ae* namespace database system layers.

This class inherits from SystemConnectorBase the following attributes:

system: instance of the related ae-sys_core.SystemBase class.

console_app: instance of the application using this database system.

last_err_msg: the error message string of the last error or an empty string if no error occurred in the last database action/operation.

__init__(*system*)

create instance of generic database object (base class for real database like e.g. postgres or oracle).

Parameters

system (SystemBase) – SystemBase instance. Providing useful attributes, like e.g.:

credentials: dict with database driver specific account credentials.

features: list of features. Features will also be passed as connection parameters to the database driver. The main differences to credentials are that a feature without any value will be interpreted as boolean True and that features can have any value that can be specified as a literal (like int or even datetime).

console_app: instance of the application using this database system.

conn

database driver connection

curs

database driver cursor

param_style: str

database driver bind variable/parameter style

abstract connect()

sub-class has to implement this connect method

Return type

str

_adapt_sql(*sql*, *bind_vars*)

replace the parameter style of bind variables from *pyformat* into *named*.

Parameters

- **sql** (str) – query to scan for named bind variables.
- **bind_vars** (Dict[str, Any]) – dict of all available bind variables.

Return type

str

Returns

adapted query string.

Note: For database drivers - like psycopg2 - that are support only the *pyformat* parameter style syntax (in the format %(bind_var)s) the sql query string will be adapted, by converting all bind variables from the parameter style *named* into *pyformat*.

The returned query will be unchanged for all other database drivers (that are directly supporting the *named* parameter style).

---

call_proc(*proc_name*, *proc_args*, *ret_dict=None*)

execute stored procedure on database server.

Parameters

- **proc_name** (str) – name of the stored procedure.
- **proc_args** (Sequence) – tuple of parameters/arguments passed to the stored procedure.
- **ret_dict** (Optional[Dict[str, Any]]) – optional dict - if passed then the dict item with the key *return* will be set/updated to the value returned from the stored procedure/database.

Return type
str

Returns
empty string if no error occurred else the error message.

close(*commit=True*)

close the connection to the database driver and server.

Parameters
commit (bool) – pass False to prevent commit (and also execute rollback) before closure of connection.

Return type
str

Returns
empty string if no error occurred else the error message.

connect_params()

merges self.system.credentials with self.system.features into a database driver connection parameters dict.

Return type
Dict[str, Any]

Returns

new dict with the items from self.system.credentials and then extended with the entries of self.system.features.

Credential values are always of type str, only features can be of any type. Features without a specified value are set True. All keys of the returned dict will be in lower-case.

create_cursor()

allow sub-class to create Python DB API-conform database driver cursor

cursor_description()

return description text if opened cursor or None if cursor is closed or not yet opened.

Return type
Optional[str]

fetch_all()

fetch all the rows found from the last executed SELECT query.

Return type
Sequence[Tuple]

Returns
empty list on error or if query result is empty, else a list of database rows.

`fetch_value`(*col_idx=0*)

fetch the value of a column of the first/next row of the found rows of the last SELECT query.

Parameters
col_idx (`int`) – index of the column with the value to fetch and return.

Return type
`Any`

Returns
value of the column at index *col_idx*.

`execute_sql`(*sql*, *commit=False*, *bind_vars=None*)

execute sql query with optional bind variables.

Parameters

- **sql** (`str`) – sql query to execute.
- **commit** (`bool`) – pass True to execute a COMMIT command directly after the query execution.
- **bind_vars** (`Optional`[`Dict`[`str`, `Any`]]) – optional dict with bind variables.

Return type
`str`

Returns
empty string if no error occurred else the error message.

`delete`(*table_name*, *chk_values=None*, *where_group_order=''*, *bind_vars=None*, *commit=False*)

execute a DELETE command against a table.

Parameters

- **table_name** (`str`) – name of the database table.
- **chk_values** (`Optional`[`Dict`[`str`, `Any`]]) – dict of column names/values to identify the record(s) to delete.
- **where_group_order** (`str`) – extra sql added after the WHERE clause (merged with chk_values by `_rebind()`). This string can include additional WHERE expressions with extra bind variables.
- **bind_vars** (`Optional`[`Dict`[`str`, `Any`]]) – dict of extra bind variables (key=variable name, value=value).
- **commit** (`bool`) – bool value to specify if commit should be done. Pass True to commit.

Return type
`str`

Returns
last error message or empty string if no errors occurred.

`insert`(*table_name*, *col_values*, *returning_column=''*, *commit=False*)

execute an INSERT command to add one record to a database table.

Parameters

- **table_name** (`str`) – name of the database table.

- **col_values** (Dict[str, Any]) – dict of inserted column values with the column name as key.
- **returning_column** (str) – name of column which value will be returned by next fetch_all/fetch_value() call.
- **commit** (bool) – bool value to specify if commit should be done. Pass True to commit.

Return type
str

Returns
last error message or empty string if no errors occurred.

select(*from_join=''*, *cols=()*, *chk_values=None*, *where_group_order=''*, *bind_vars=None*, *hints=''*)

execute a SELECT query against a database table.

Parameters

- **from_join** (str) – name(s) of the involved database table(s), optional with JOIN clause(s). Passing an empty string results in a SELECT statement without the FROM keyword.
- **cols** (Sequence[str]) – sequence of the column names that will be selected and included in the resulting data-rows.
- **chk_values** (Optional[Dict[str, Any]]) – dict of column names/values to identify selected record(s).
- **where_group_order** (str) – extra sql added after the WHERE clause (merged with chk_values by `_rebind()`). This string can include additional WHERE expressions with extra bind variables, ORDER BY and GROUP BY expressions. These special/extra bind variables have to be specified in the *bind_vars* argument and have to be prefixed with the string *'CV_'* in the WHERE clause (see also the *CHK_BIND_VAR_PREFIX* data constant in this module).
- **bind_vars** (Optional[Dict[str, Any]]) – dict of extra bind variables (key=variable name, value=value).
- **hints** (str) – optional SELECT optimization hint string.

Return type
str

Returns
last error message or empty string if no errors occurred. Use the methods *fetch_all()* or *fetch_value()* to retrieve the resulting data-rows.

update(*table_name*, *col_values*, *chk_values=None*, *where_group_order=''*, *bind_vars=None*, *commit=False*, *locked_cols=()*)

execute an UPDATE command against a database table.

Parameters

- **table_name** (str) – name of the database table.
- **col_values** (Dict[str, Any]) – dict of inserted/updated column values with the column name as key.
- **chk_values** (Optional[Dict[str, Any]]) – dict of column names/values to identify affected record(s). If not passed then the first name/value of *col_values* is used as primary key check/filter value.

- **where_group_order** (str) – extra sql added after the WHERE clause (merged with chk_values by _rebind()). This string can include additional WHERE expressions with extra bind variables, ORDER BY and GROUP BY expressions. These special/extra bind variables have to be specified in the *bind_vars* argument and have to be prefixed with the string *'CV_'* (see also the *CHK_BIND_VAR_PREFIX* data constant in this module).
- **bind_vars** (Optional[Dict[str, Any]]) – dict of extra bind variables (key=variable name, value=value).
- **commit** (bool) – bool value to specify if commit should be done. Pass True to commit.
- **locked_cols** (Sequence[str]) – list of column names not be overwritten on update of column value is not empty.

Return type
str

Returns
last error message or empty string if no errors occurred.

upsert(*table_name*, *col_values*, *chk_values*, *where_group_order=''*, *bind_vars=None*, *returning_column=''*, *commit=False*, *locked_cols=()*, *multiple_row_update=True*)

execute an INSERT or UPDATE command against a record of a database table (UPDATE if record already exists).

Parameters

- **table_name** (str) – name of the database table.
- **col_values** (Dict[str, Any]) – dict of inserted/updated column values with the column name as key.
- **chk_values** (Dict[str, Any]) – dict of column names/values to identify affected record(s), also used to check if record already exists (and data has to updated instead of inserted). If not passed then the first name/value of col_values is used as primary key check/filter value.
- **where_group_order** (str) – extra sql added after the WHERE clause (merged with chk_values by _rebind()). This string can include additional WHERE expressions with extra bind variables, ORDER BY and GROUP BY expressions. These special/extra bind variables have to be specified in the *bind_vars* argument and have to be prefixed with the string *'CV_'* in the query string (see also the *CHK_BIND_VAR_PREFIX* data constant in this module).
- **bind_vars** (Optional[Dict[str, Any]]) – dict of extra bind variables (key=variable name, value=value).
- **returning_column** (str) – name of column which value will be returned by next fetch_all/fetch_value() call.
- **commit** (bool) – bool value to specify if commit should be done. Pass True to commit record changes.
- **locked_cols** (Sequence[str]) – list of column names not be overwritten on update of column value is not empty.
- **multiple_row_update** (bool) – allow update of multiple records with the same chk_values.

Return type
str

Returns
last error message or empty string if no errors occurred.

`commit`(*reset_last_err_msg=False*)

commit the current transaction if the database driver supports/implements a commit method.

Parameters
reset_last_err_msg¶ (bool) – pass True to reset the last error message of this instance before the commit.

Return type
str

Returns
last error message or empty string if no error happened.

`rollback`(*reset_last_err_msg=False*)

roll the current transaction back if the DB driver supports transactions and does have a rollback method.

Parameters
reset_last_err_msg¶ (bool) – pass True to reset the last error message of this instance before the rollback.

Return type
str

Returns
last error message or empty string if no error happened.

`get_row_count`()

determine rowcount of last executed query.

Return type
int

Returns
the number of affected rows of the last query.

`selected_column_names`()

determine the column names fo the last executed SELECT query.

Return type
List[str]

Returns
list of column names - order by column index.

`static thread_lock_init`(*table_name*, *chk_values*)

created named locking instance for passed table and filter/check expression.

Return type
NamedLocks

Returns
NamedLocks instance for this table and filter.

4.23 ae.db_ora

4.23.1 database system core layer to access Oracle databases

the class *OraDb* of this namespace portion is a thin layer that is extending the *DbBase* of the module *ae.db_core* to connect to an Oracle database.

Hint: this namespace portion is using the cx_Oracle package as the database driver. the cx_Oracle package has to have at least version 5 or higher.

basic usage of the oracle database layer

to create an instance of the class *OraDb* you first have to create a *SystemBase* instance. this can be done either programmatically by providing an application instance (of the class *ConsoleApp* or an inherited sub-class of it) plus any database parameters, like required credentials and any database configuration features/options:

```
app = ConsoleApp()
system = SystemBase('system-id', app, dict(User='user name', Password='password`, ...), ...)
```

alternatively provide all system-specific info within the *ae config files* and let *UsedSystems* load it:

```
system = used_systems['system-id']
```

finally pass the database parameters in *system* to create an instance of *OraDb*:

```
ora_db = OraDb(system)
```

then call the *connect()* method of this instance to connect to the Oracle database server:

```
error_message = ora_db.connect()
if error_message:
    print(error_message)
```

if the connection could not be established then *connect()* is returning an error message string. if the return value is an empty string then you can use all the methods provided by *DbBase*, like e.g. *update()*:

```
error_message = ora_db.update('my_table`, {'my_col': 'new value'})
if error_message:
    print(error_message)
    error_message = ora_db.rollback()
```

an explicit call of *rollback()* is only needed if you use transactions. in this case you should also use *commit()* at the end of each transaction to store any data updates:

```
error_message = ora_db.commit()
```

alternatively you can use the *commit* argument that is provided by the *DbBase* DML methods: by passing a *True* value to this argument, the method will automatically execute a *commit()* call for you if no error occurred in the DML method:

```
error_message = ora_db.update('table`, {'column': 369}, commit=True)
```

finally after all database actions are done you can close the connection to the databases server with the *close()* method:

```
error_message = ora_db.close()
```

Classes

OraDb(system)	Oracle database class, based on *DbBase*

class OraDb(*system*)

Bases: *DbBase*

Oracle database class, based on *DbBase*

__init__(*system*)

create instance of oracle database object.

Parameters
system (*SystemBase*) – instance of a *SystemBase* class.

SystemBase (defined in the module *ae.sys_core*) is providing the credentials and features, which get retrieved from *config files*, then converted by *connect_params()* into *connection parameters* to connect to the Postgres database.

if you experiencing the following unicode encoding error:

```
'charmap' codec can't decode byte 0x90 in position 2: character maps to
↪<undefined>
```

don't try to create a type handler like recommended in some places - still got same error after adding the following method to replace the self.conn.outputtypehandler of the database driver:

```
def output_type_handler(cursor, name, default_type, size, precision, scale):
    if default_type in (cx_Oracle.STRING, cx_Oracle.FIXED_CHAR):
        return cursor.var(cx_Oracle.NCHAR, size, cursor.arraysize)
```

luckily, finally found workaround by setting the following OS environment variable to the character set of the used Oracle server (here UTF8):

```
os.environ["NLS_LANG"] = ".AL32UTF8"
```

connect()

connect this instance to the database driver.

Return type
str

prepare_ref_param(*value*)

prepare special Oracle reference parameter.

Parameters
value (Union[datetime, int, float, str]) – the input value passed into the reference parameter of the called stored procedure.

Return type
Any

Returns
a handle to the reference variable.

the following code snippet shows how to use this method together with *get_value()* to retrieve the returned value of a reference parameter:

```
ora_db = OraDb(...)
*ref_var* = ora_db.prepare_ref_param("input_value")
err_msg = ora_db.call_proc('STORED_PROCEDURE', (*ref_var*, ...))
if not err_msg:
    output_value = ora_db.get_value(*ref_var*)
```

static get_value(*var*)

get output value from a reference variable passed into a stored procedure.

Parameters
var¶ – handle to a reference variable.

Return type
Any

Returns
output value of the reference variable.

static set_value(*var*, *value*)

set the input value of a reference variable to pass into a stored procedure.

Parameters

- **var**¶ (Any) – handle to the reference variable to set.
- **value**¶ (Union[datetime, int, float, str]) – value to set as input value of the reference variable.

param_style: str

database driver bind variable/parameter style

last_err_msg: str

last system connection error message(s)

4.24 ae.db_pg

4.24.1 postgres database layer

this module provides the class *PostgresDb* to connect and interact with a Postgres database server.

the communication with the database driver is done with great help of the psycopg2 pypi package.

basic usage of ae.db_pg

to create an instance of the class *PostgresDb* you first have to create a *SystemBase* instance. this can be done either programmatically by providing an application instance (of the class `ConsoleApp` or an inherited subclass of it) plus any database parameters, like required credentials and any database configuration features/options:

```
app = ConsoleApp()
system = SystemBase('system-id', app, dict(User='user name', Password='password`,␣
↪Dbname=...), ...)
```

alternatively provide all system-specific info within the *ae config files* and let *UsedSystems* load it:

```
system = used_systems['system-id']
```

finally pass the database parameters in *system* to create an instance of *PostgresDb*:

```
pg_db = PostgresDb(system)
```

then call the *connect()* method of this instance to connect to the Postgres database server:

```
error_message = pg_db.connect()
if error_message:
    print(error_message)
```

if the connection could not be established then *connect()* is returning an error message string. if the return value is an empty string then you can use all the methods provided by *DbBase*, like e.g. *update()*:

```
error_message = pg_db.update('my_table', {'my_col': 'new value'})
if error_message:
    print(error_message)
    error_message = pg_db.rollback()
```

an explicit call of *rollback()* is only needed if you use transactions (autocommit is False). in this case you should also use *commit()* at the end of each transaction to store any data updates:

```
error_message = pg_db.commit()
```

alternatively you can use the *commit* argument that is provided by the *DbBase* DML methods: by passing a *True* value to this argument, the method will automatically execute a *commit()* call for you if no error occurred in the DML method:

```
error_message = pg_db.update('table`, {'column': 369}, commit=True)
```

finally after all database actions are done you can close the connection to the databases server with the *close()* method:

```
error_message = pg_db.close()
```

Classes

PostgresDb(system)	an instance of this class represents a Postgres database.

class PostgresDb(*system*)

Bases: DbBase

an instance of this class represents a Postgres database.

__init__(*system*)

create instance of Postgres database object

Parameters
system (SystemBase) – instance of a SystemBase class.

SystemBase (defined in the module ae.sys_core) is providing the credentials and features, which get retrieved from *config files*, then converted by connect_params() into *connection parameters* to connect to the Postgres database.

for connections via SSL to the Postgres server you have to add either the connection parameters **sslmode**, **sslcert** and **sslkey** or **sslrootcert** and **sslcrl** (depending on the configuration of your server).

features : optional list of features.

param_style: str

database driver bind variable/parameter style

connect()

connect this instance to the Postgres database server, using the credentials provided at instantiation.

Return type
str

Returns
error message in case of error or empty string if not.

execute_sql(*sql*, *commit=False*, *bind_vars=None*, *auto_commit=False*)

execute sql query or sql command.

Parameters

- **sql** (str) – sql query or command to execute.
- **commit** (bool) – pass True to commit (after INSERT or UPDATE queries).
- **bind_vars** (Optional[Dict[str, Any]]) – dict of bind variables (key=variable name, value=value).
- **auto_commit** (bool) – pass True activate auto-commit-mode for this postgres session.

Return type
str

Returns
last error message or empty string if no errors occurred.

Hint: overwriting generic execute_sql for Postgres because if auto_commit is False then a db error is invalidating the connection until it gets rolled back (optionally to a save-point). unfortunately psycopg2

does not provide/implement save-points. could be done alternatively with execute("SAVEPOINT NonAutoCommErrRollback") but RELEASE/ROLLBACK makes it complicated (see also https://stackoverflow.com/questions/2370328/continuing-a-transaction-after-primary-key-violation-error):

```
save_point = None if auto_commit else self.conn.setSavepoint(
↪'NonAutoCommErrRollback')
super().execute_sql(sql, commit=commit, auto_commit=auto_commit, bind_vars=bind_
↪vars)
if save_point:
    if self.last_err_msg:
        self.conn.rollback(save_point)
    else:
        self.conn.releaseSavepoint(save_point)
return self.last_err_msg
```

therefore KISS - a simple rollback will do it also.

last_err_msg: *str*

last system connection error message(s)

4.25 ae.transfer_service

4.25.1 transfer client and server services

this ae portion is providing client and server services to transfer files and text messages between two devices in the same local network.

the number of parallel running file and message transfers is only limited by the available resources of the involved devices.

if a file transfers gets interrupted it can be recovered later and without the need to resend the already transferred file content.

standard file paths - like e.g. the documents or downloads folders - are getting automatically adopted to the specific path structures of each involved device and operating system.

transfer service life cycle

the transfer service can be invoked in different ways: standalone as a separate process or attached and embedded into a controlling application.

run transfer service in standalone mode

execute this module to run the transfer services as a separate standalone process via:

```
python transfer_service.py [--bind=...] [--port=...] [--buf_len=...]
```

the following command line options are overwriting the default server address and socket buffer length (see also *service_factory()*):

- 'bind' to restrict the incoming connections to an ip address/range (overwriting the default *SERVER_BIND*).

- 'port' to specify the socket port (overwriting the default port *SERVER_PORT*).
- 'buf_len' to specify the socket buffer length (overwriting the default buffer length *SOCKET_BUF_LEN*).

after that the transfer service will be able to receive files send from another process or device.

Note: on Android a standalone transfer service has to be started as android service.

run transfer service attached to any app

alternatively you can run the transfer service server in a separate thread within respectively attached to your application:

```
from ae.transfer_service import service_factory

transfer_service_app = service_factory()
transfer_service_app.set_option('port', 12345, save_to_config=False)
transfer_service_app.set_option('buf_len', 34567, save_to_config=False)
transfer_service_app.start_server(threaded=True)
```

pause or stop transfer service

to manually pause the transfer service, store the app instance of the transfer service app (*transfer_service_app* in the example above) and call its *stop_server()* method:

```
transfer_service_app.stop_server()
```

to fully stop the transfer service and terminate the transfer service app call instead its *shutdown()* method:

```
transfer_service_app.shutdown()
```

Hint: the *shutdown()* method of the base app instance (*AppBase*) automatically ensures a clean shutdown of the transfer service server on app quit/exit.

send file to another transfer service server

to send files from one transfer server to another running transfer server, a separate client process has to be started on the device storing the file to be send.

to initiate the file transfer the client process has to make a tcp connection to the transfer server running on the same device, specifying the path of the file to send and the remote ip of the receiving transfer server and finally call the remote procedure *send_file* like shown in the following example:

```
import socket
from ae.transfer_service import connect_and_request

request_kwargs = dict(method_name='send_file', file_path='path_to_file/file_name.ext',␣
↪remote_ip='192.168.3.123')
with socket.socket() as sock:
```

(continues on next page)

(continued from previous page)

```
    response_kwargs = connect_and_request(sock, request_kwargs)

if 'error' in response_kwargs:
    # handle error (e.g. display to user or add to a log file)
```

if the file transfer failed then the transfer kwargs dict returned by *connect_and_request()* will contain an *error* key containing the error message text.

implemented remote procedures

the following remote procedures are provided by the transfer service server:

- *cancel_request*: cancel running file transfer.
- *pending_requests*: get log info the progress/status of all currently running file transfers.
- *recv_file*: receive file from other transfer service server.
- *recv_message*: receive text message from other transfer service server.
- *send_file*: send file to other transfer service server.
- *send_message*: send text message to other transfer service server.

Hint: the demo app ComPartY is using all provided remote procedures.

Module Attributes

CONNECTION_TIMEOUT	default timeout (in seconds) to connect and request a server process
CONNECT_ERR_PREFIX	error message string prefix if error happened directly in *connect_and_request()* helper (no protocol error).
ENCODING_KWARGS	default encoding and encoding error handling strategy
SERVER_BIND	setting BIND to '' or None to allow connections for all available interfaces
SERVER_PORT	server listening port
SHUTDOWN_TIMEOUT	timeout (in seconds) to shutdown/stop the console app
SOCKET_BUF_LEN	buf length for socket receives and sends
TRANSFER_KWARGS_DATE_TIME_NAME_PARTS	kwarg name suffix for values automatically converted
TRANSFER_KWARGS_LINE_END_CHAR	end of line/command character as string
TRANSFER_KWARGS_LINE_END_BYTE	
TransferKwargs	command/action format of requests and responses
requests_lock	locking requests TransferKwargs in *TransferServiceApp.reqs_and_logs*
server_app	transfer service server app

Functions

clean_log_str(log_str)	remove high-commas and backslashes from the passed string to add it to the logs (preventing duplicates).
connect_and_request(sock, request_kwargs[, ...])	connect to remote, send first command/action and return response as transfer kwargs dict.
recv_bytes(sock[, buf_len])	receive all bytes from the passed client socket instance until connection lost or line end reached.
service_factory([task_id_func])	create server app instance including the command line options *bind* and *port*.
transfer_kwargs_error(transfer_kwargs, err_msg)	add/append error to transfer kwargs dict without overwriting any previous error.
transfer_kwargs_from_literal(transfer_kwargs_lit	convert dict literal (created with transfer_kwargs_literal()) to transfer kwargs dict.
transfer_kwargs_literal(transfer_kwargs)	convert dict to str literal to be sent via sockets (re-instantiable via transfer_kwargs_from_literal()).
transfer_kwargs_update(*variables, **kwargs)	update multiple transfer dicts with the same keys/values (locking with requests_lock).

Classes

ThreadedTCPRequestHandler(request, ...)	server request handler.
TransferServiceApp([app_title, app_name, ...])	server service app class

CONNECTION_TIMEOUT = 2.7

default timeout (in seconds) to connect and request a server process

CONNECT_ERR_PREFIX = 'transfer_service.connect_and_request() exception '

error message string prefix if error happened directly in *connect_and_request()* helper (no protocol error).

ENCODING_KWARGS = {'encoding': 'UTF-8', 'errors': 'ignore'}

default encoding and encoding error handling strategy

SERVER_BIND = ''

setting BIND to '' or None to allow connections for all available interfaces

SERVER_PORT = 36969

server listening port

SHUTDOWN_TIMEOUT = 3.9

timeout (in seconds) to shutdown/stop the console app

SOCKET_BUF_LEN = 16384

buf length for socket receives and sends

TRANSFER_KWARGS_DATE_TIME_NAME_PARTS = ('_date', '_time')

kwarg name suffix for values automatically converted

TRANSFER_KWARGS_LINE_END_CHAR = '\n'

end of line/command character as string

TRANSFER_KWARGS_LINE_END_BYTE = b'\n'

TransferKwargs

command/action format of requests and responses

alias of `Dict[str, Any]`

requests_lock = <unlocked _thread.lock object>

locking requests TransferKwargs in *TransferServiceApp.reqs_and_logs*

clean_log_str(*log_str*)

remove high-commas and backslashes from the passed string to add it to the logs (preventing duplicates).

Parameters
log_str (Union[str, bytes]) – log string or bytes array to clean up.

Return type
str

Returns
cleaned log string.

connect_and_request(*sock*, *request_kwargs*, *buf_len=16384*, *timeout=2.7*)

connect to remote, send first command/action and return response as transfer kwargs dict.

Parameters

- **sock** (socket) – new socket instance.
- **request_kwargs** (Dict[str, Any]) – first/initial request kwargs dict. if the key *'server_address'* is not provided (with the server address as (host, ip) tuple), then ('localhost', SERVER_PORT) is used. if the key 'local_ip' is not specified then the local ip address will be used.
- **buf_len** (int) – socket buffer length. if not passed then *SOCKET_BUF_LEN* will be used. pass zero/0 to use the buf length defined via the 'buf_len' command line option.
- **timeout** (Optional[float]) – timeout in seconds or None to use socket/system default timeout. if not passed then the default timeout specified by *CONNECTION_TIMEOUT* will be used.

Return type
Dict[str, Any]

Returns
response transfer kwargs dict.

recv_bytes(*sock*, *buf_len=16384*)

receive all bytes from the passed client socket instance until connection lost or line end reached.

Parameters

- **sock** (socket) – socket instance.
- **buf_len** (int) – socket buffer length. if not passed then *SOCKET_BUF_LEN* will be used. pass zero/0 to use the buf length defined via the 'buf_len' command line option.

Return type
bytes

Returns
received bytes.

service_factory(*task_id_func=None*)

create server app instance including the command line options *bind* and *port*.

Parameters
task_id_func (Optional[Callable[[str, str, str], str]]) – callable to return an unique id for a transfer request task.

Return type
TransferServiceApp

Returns
transfer service app instance.

transfer_kwargs_error(*transfer_kwargs*, *err_msg*)

add/append error to transfer kwargs dict without overwriting any previous error. :type _sphinx_paramlinks_ae.transfer_service.transfer_kwargs_error.transfer_kwargs: Dict[str, Any] :param _sphinx_paramlinks_ae.transfer_service.transfer_kwargs_error.transfer_kwargs: request/response transfer kwargs dict. :type _sphinx_paramlinks_ae.transfer_service.transfer_kwargs_error.err_msg: str :param _sphinx_paramlinks_ae.transfer_service.transfer_kwargs_error.err_msg: error message to add.

transfer_kwargs_from_literal(*transfer_kwargs_lit*)

convert dict literal (created with transfer_kwargs_literal()) to transfer kwargs dict.

Parameters
transfer_kwargs_lit (str) – request/response dict literal string.

Return type
Dict[str, Any]

Returns
transfer kwargs dict.

transfer_kwargs_literal(*transfer_kwargs*)

convert dict to str literal to be sent via sockets (re-instantiable via transfer_kwargs_from_literal()).

Note: to ensure security (and prevent injections) only the following basic types can be used: *int*, *float*, *boolean*, *str*, *bytes*, *list*, *tuple* and *dict*. date/time values are only allowed as dict value and of the type *datetime.datetime*; additionally the key of this dict value has to contain one of the fragments defined in *TRANSFER_KWARGS_DATE_TIME_NAME_PARTS*.

Parameters
transfer_kwargs (Dict[str, Any]) – request/response transfer kwargs dict.

Return type
str

Returns
literal string of transfer kwargs dict terminated with TRANSFER_KWARGS_LINE_END_CHAR.

transfer_kwargs_update(**variables*, ***kwargs*)

update multiple transfer dicts with the same keys/values (locking with requests_lock).

Parameters

- **variables** – transfer kwargs variables/dicts.
- **kwargs** – kwargs to update.

class ThreadedTCPRequestHandler(*request*, *client_address*, *server*)

Bases: StreamRequestHandler

server request handler.

self.rfile is a file-like object created by the handler; to use e.g. readline() instead of raw recv() likewise, self.wfile is a file-like object used to write back to the client.

handle()

handle a single request

class TransferServiceApp(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*, *cfg_opt_eval_vars=None*, *additional_cfg_files=()*, *cfg_opt_val_stripper=None*, *formatter_class=None*, *epilog=''*, ***logging_params*)

Bases: *ConsoleApp*

server service app class

reqs_and_logs: List[Dict[str, Any]]

list of transfer_kwargs of currently processed requests

server_instance: Optional[ThreadingTCPServer]

server class instance

server_thread: Optional[Thread]

server thread (main or separate thread)

cancel_request(*request_kwargs*, *handler*)

cancel running request.

Parameters

- **request_kwargs** (Dict[str, Any]) – request transfer kwargs dict, specifying via *rt_id_to_cancel* the request to cancel.
- **handler** (StreamRequestHandler) – request handler instance.

Return type
Dict[str, Any]

Returns
response kwargs, having *error* key if request could not be found/cancelled.

static id_of_task(*action*, *object_type*, *object_key*)

compile the id of a transfer request task.

Parameters

- **action** (str) – action or log level string.
- **object_type** (str) – task object type ('log' for log entries, 'file' for file transfers, else 'message').
- **object_key** (str) – task key (file path for file transfers, message string for messages and timestamp for log entries).

Return type
str

Returns
unique key identifying the task/log-entry.

log(*log_level*, *message*)

print log message and add it to reqs_and_logs (to be read by controller app).

Note: please note that you have to use `print()` or one of the console print methods of the `AppBase` (like e.g. `verbose_out()`, respective self.vpo) instead of this method for the logging of low level transport methods/functions (like e.g. *pending_requests()*, *response_to_request()*, *handle()* or *recv_bytes()*). this will prevent the duplication of a log message, because each call of this method creates a new entry in *reqs_and_logs* which will be sent to the controlling app via the low level transport methods (which would recursively grow the sent messages until the system freezes), especially if the transfer kwargs are included into the log message.

Parameters

- **log_level** (`str`) – 'print' always prints, 'debug' prints if self.debug, 'verbose' prints if self.verbose.
- **message** (`str`) – message to print.

pending_requests(*request_kwargs*, *handler*)

determine currently running/pending server requests and debug log messages (in debug mode only).

Parameters

- **request_kwargs** (`Dict[str, Any]`) – request transfer kwargs dict.
- **handler** (`StreamRequestHandler`) – request handler class instance.

Return type
`Dict[str, Any]`

Returns
response transfer kwargs dict keys.

recv_file(*request_kwargs*, *handler*)

receive binary file content from client.

Parameters

- **request_kwargs** (`Dict[str, Any]`) – request transfer kwargs dict with the following keys:
  - *'file_path'*: file path (can contain path placeholders).
  - *'total_bytes'*: total file length in bytes.
  - *'series_file'*: optionally, pass any value to ensures new file name.
- **handler** (`StreamRequestHandler`) – request handler class instance.

Return type
`Dict[str, Any]`

Returns

response transfer kwargs dict keys (request kwargs extended with additional keys):

- *'error'*: error message string if an error occurred.
- *'transferred_bytes'*: start offset on recovered transfer (previously received bytes).

recv_message(*request_kwargs*, *handler*)

receive message from client/peer.

Return type
Dict[str, Any]

response_to_request(*request_lit*, *handler*)

process request to this server and return response string.

Note: this method is running in a separate thread (created by the server to process this request).

Parameters

- **request_lit** (str) – request string, which is a dict literal with *'method_name'* key.
- **handler** (StreamRequestHandler) – stream request handler instance.

Return type
str

Returns
response string with transfer kwargs as literal. if an error occurred then the returned response kwargs contains an *'error'* key which stores the related error message.

send_file(*request_kwargs*, *handler*)

send binary file content to remote server.

Return type
Dict[str, Any]

send_message(*request_kwargs*, *handler*)

send message to remote server.

Return type
Dict[str, Any]

shutdown(*exit_code=0*, *timeout=None*)

overwritten to stop a running transfer service server/threads on shutdown of this app instance.

Parameters

- **exit_code** (Optional[int]) – set application OS exit code - see *shutdown()*.
- **timeout** (Optional[float]) – timeout float value in seconds - see *shutdown()*.

start_server(*threaded=False*)

start server and run until main app *stop_server()*.

Parameters
threaded (bool) – optionally pass True to use separate thread for the server process.

Return type
str

Returns
empty string/"" if server instance/thread got started else error message string.

stop_server()

stop/pause transfer service server - callable also if not running to reset/prepare this app instance.

server_app: *TransferServiceApp* | None = None
transfer service server app

4.26 ae.sideloading_server

4.26.1 sideloading server

this ae namespace portion provides a simple http server to download a file to another device in the same local network, using any web browser.

by adding this module to your android app you can distribute e.g. the APK file of your app directly to another android device to install it there - without requiring an app store or an internet connection.

The implementation is inspired by the accepted answer to https://stackoverflow.com/questions/18543640/how-would-i-create-a-python-web-server-that-downloads-a-file-on-any-get-request.

Useful debugging tools can be found at https://www.maketecheasier.com/transferring-files-using-python-http-server.

sideloading server lifecycle

The sideloading server provided by this ae namespace portion can be started by first creating an instance of the sideloading app and then call its *start_server()* method:

```
from ae.sideloading_server import server_factory

sideloading_server_app = server_factory()
sideloading_server_app.start_server()
```

By calling *start_server()* without arguments a default file mask (*DEFAULT_FILE_MASK*) will be used, which specifies the APK file of the main app situated in the *Downloads* folder of the device.

The web address that has to be entered in the web browser of the receiving device can be determined with the *server_url()* method:

```
client_url = sideloading_server_app.server_url()
```

While the server is running the *client_progress()* method is providing the progress state of any currently running sideloading task. More detailed info of the running sideloading process can be gathered by calling the *fetch_log_entries()* method.

to temporarily stop the sideloading server simple call the *stop_server()* method. On app exit additionally call the method *shutdown()*.

Hint: The ae namespace portion *kivy_sideloading* is providing a package for an easy integration of this sideloading server into your kivy app.

An example usage of this sideloading server and the *ae.kivy_sideloading* package is integrated into the demo apps GlslTester, Lisz and ComPartY.

Module Attributes

DEFAULT_FILE_MASK	default glob file mask to sideloading file
FILE_COUNT_MISMATCH	err msg part of *SideloadingServerApp.start_server()*
SERVER_BIND	setting BIND to '' or None to allow connections for all local devices
SERVER_PORT	http server listening port (a port number under 1024 requires root privileges)
SOCKET_BUF_LEN	buf length for socket sends in *response_to_request()*
SHUTDOWN_TIMEOUT	timeout(in seconds) to shut down/stop the console app and http sideloading server
SideloadingKwargs	command/log format of sideloading requests

Functions

server_factory([task_id_func])	create server app instance.
update_handler_progress([handler, ...])	default progress update callback.

Classes

SideloadingServerApp([app_title, app_name, ...])	server service app class
SimpleHTTPRequestHandler(request, ...)	server request handler.

DEFAULT_FILE_MASK = '{downloads}/{main_app_name}*.apk'

default glob file mask to sideloading file

FILE_COUNT_MISMATCH = ' files found matching '

err msg part of *SideloadingServerApp.start_server()*

SERVER_BIND = ''

setting BIND to '' or None to allow connections for all local devices

SERVER_PORT = 36996

http server listening port (a port number under 1024 requires root privileges)

SOCKET_BUF_LEN = 16384

buf length for socket sends in *response_to_request()*

SHUTDOWN_TIMEOUT = 3.9

timeout(in seconds) to shut down/stop the console app and http sideloading server

SideloadingKwargs

command/log format of sideloading requests

alias of `Dict[str, Any]`

server_factory(*task_id_func=None*)

create server app instance.

Parameters
task_id_func (Optional[Callable[[str, str, str], str]]) – callable to return an unique id for a transfer request task.

Return type
SideloadingServerApp

Returns
sideloading server app instance.

update_handler_progress(*handler=None*, *transferred_bytes=-3*, *total_bytes=-3*, ***_kwargs*)

default progress update callback.

Parameters

- **handler** (Optional[*SimpleHTTPRequestHandler*]) – server request handler - containing the attributes to be updated.
- **transferred_bytes** (int) – already transferred bytes or error code.
- **total_bytes** (int) – total sideloading bytes.
- **_kwargs** – additional/optional kwargs (e.g. client_ip).

class SimpleHTTPRequestHandler(*request*, *client_address*, *server*)

Bases: BaseHTTPRequestHandler

server request handler.

progress_total: int = -1

total file size in bytes to be transferred

progress_transferred: int = -1

transferred bytes in this sideloading progress thread

do_GET()

handler callback for a http GET command request.

log_request(*code='-'*, *size='-'*)

overwrite to prevent console output if not in debug mode.

class SideloadingServerApp(*app_title=''*, *app_name=''*, *app_version=''*, *sys_env_id=''*, *debug_level=0*, *multi_threading=False*, *suppress_stdout=False*, *cfg_opt_eval_vars=None*, *additional_cfg_files=()*, *cfg_opt_val_stripper=None*, *formatter_class=None*, *epilog=''*, ***logging_params*)

Bases: *ConsoleApp*

server service app class

client_handlers: Dict[str, *SimpleHTTPRequestHandler*]

handler for each active client connection

file_path: str

path to the found sideloading file

load_requests: List[Dict[str, Any]]

sideloading requests and error/debug log

progress_callback(***_kwargs*)

progress event callback

server_instance: Optional[ThreadingHTTPServer]

server class instance

server_thread: Optional[Thread]

server thread (main or separate thread)

client_progress(*client_ip*)

determine sideloading progress (transferred bytes) for the client with the passed ip address.

This method can be used alternatively to the callback *progress_callback*.

Parameters
client_ip (str) – client ip address.

Return type
Tuple[int, int]

Returns
tuple of two int values: status and file size of the sideloading file. A positive status value signifies the number of already transferred bytes. A negative value in this first int signals an error. Possibles error codes are: * -1 if sideloading task did not start yet * -2 if not exists or sideloading task has already finished * -3 if the default Please note that the file size given in the second int can be 0 if an error has occurred or if the sideloading task is not yet fully created/started.

static id_of_task(*action*, *object_type*, *object_key*)

compile the id of the sideloading request task.

Parameters

- **action** (str) – action or log level string.
- **object_type** (str) – task object type (currently only 'log' is implemented/supported).
- **object_key** (str) – task key (for log entries the timestamp of the log entry is used).

Return type
str

Returns
unique key identifying the task/log-entry.

log(*log_level*, *message*)

print log message and optionally add it to the requests (to be read by controller app).

Parameters

- **log_level** (str) – 'print' always prints, 'debug' prints if *self.debug* is True and 'verbose' prints a message to the log if *self.verbose* is True.
- **message** (str) – message to print.

fetch_log_entries()

collect last log entries, return them and remove them from this app instance.

Return type
Dict[str, Dict[str, Any]]

Returns
Dict[request_tasks_id, sideloading_request_kwargs] with fetched requests log entries.

response_to_request(*handler*)

process request to this server and return response string.

Note: this method may run in a separate thread (created by the server to process this request).

Parameters

handler¶ (*SimpleHTTPRequestHandler*) – stream request handler instance.

server_url()

determine an url string to put into the address field of any browser to start sideloading.

Return type

str

Returns

server url string.

shutdown(*exit_code=0*, *timeout=None*)

overwritten to stop any running sideloading server instance/threads on shutdown of this app instance.

Parameters

- **exit_code**¶ (Optional[int]) – set application OS exit code - see *shutdown()*.
- **timeout**¶ (Optional[float]) – timeout float value in seconds - see *shutdown()*.

start_server(*file_mask=''*, *threaded=False*, *progress=<function update_handler_progress>*)

start http file sideloading server to run until *stop_server()* get called.

Parameters

- **file_mask**¶ (str) – optional glob.glob file mask to specify the sideloading file. If not passed then the *DEFAULT_FILE_MASK* will be used which specifies an APK file of the main app situated in the *Downloads* folder of the device. The sideloading server will only be started if the file mask matches exactly one file. The file path of the matched file can be accessed via the *file_path* attribute.
- **threaded**¶ (bool) – optionally pass True to use separate thread to run the server instance.
- **progress**¶ (Callable) – optional callback executed for each transferred/side-loaded chunk. If not specified then the default callback method *update_handler_progress()* will be called to update the progress attributes of the request handler, which can be polled via the *client_progress()* method.

Return type

str

Returns

empty string/"" if server instance/thread got started else the error message string.

Note: if *threaded* is *True* then the *progress* callback get executed in its own thread.

stop_server()

stop/pause sideloading server - callable also if not running to reset/prepare this app instance.

4.27 ae.gui_app

4.27.1 base class for python applications with a graphical user interface

the abstract base class *MainAppBase* provided by this ae namespace portion allows the integration of any Python GUI framework into the ae namespace.

on overview about the available GUI-framework-specific ae namespace portion implementations can be found in the documentation of the ae namespace portion *ae.lisz_app_data*.

extended console application environment

the abstract base class *MainAppBase* inherits directly from the ae namespace class *ae console application environment class*. the so inherited helper methods are useful to log, configure and control the run-time of your GUI app via command line arguments.

Hint: please see the documentation of *config options* and *config files* in the *ae.console* namespace portion/module for more detailed information.

MainAppBase adds on top of the *ConsoleApp* the concepts of *application events*, *application status* and *application flow*, explained further down.

application events

the events described in this section are fired on application startup and shutdown. additional events get fired e.g. in relation to the app states (documented further down in the section *app state events*) or on start or stop of an *app tour*.

the following application events are fired exactly one time at startup in the following order:

- *on_app_init*: fired **after** `ConsoleApp` app instance got initialized (detected config files) and **before** the image and sound resources and app states get loaded and the GUI framework app class instance gets initialized.
- *on_app_run*: fired **from within** the method *run_app()*, **after** the parsing of the command line arguments and options, and **before** all portion resources got imported.
- *on_app_build*: fired **after** all portion resources got loaded/imported, and **before** the framework event loop of the used GUI framework gets started.
- *on_app_started*: fired **after** all app initializations, and the start of and the initial processing of the framework event loop.

Note: the application events *on_app_build* and *on_app_started* have to be fired by the used GUI framework.

Hint: depending on the used gui framework there can be more app start events. e.g. the *ae.kivy.apps* module fires the events *on_app_built()* and *on_app_start()* (all of them fired after *on_app_run()* and *on_app_build()*). see also *kivy application events*.

---

when an application gets stopped then the following events get fired in the following order:

- *on_app_exit*: fired **after* framework win got closed and just **before** the event loop of the GUI framework will be stopped and the app shutdown.

- *on_app_quit*: fired **after** the event loop of the GUI framework got stopped and before the `AppBase.shutdown()` method will be called.

Note: the *on_app_exit* events will only be fired if the app is explicitly calling the *stop_app()* method.

Hint: depending on the used gui framework there can be more events. e.g. the *apps* module fires the events `on_app_stop()` and clock tick later *on_app_stopped()* (both of them before `on_app_quit()` get fired). see also *kivy application events*.

application status

any application- and user-specific configurations like e.g. the last window position/size, the app theme/font/language or the last selected flow within your app, could be included in the application status.

this namespace portion introduces the section *aeAppState* in the app *config files*, where any status values can be stored persistently to be recovered on the next startup of your application.

Hint: the section name *aeAppState* is declared by the *APP_STATE_SECTION_NAME* constant. if you need to access this config section directly then please use this constant instead of the hardcoded section name.

app state variables

this module is providing/pre-defining the following application state variables:

- *app_state_version*
- *flow_id*
- *flow_id_ink*
- *flow_path*
- *flow_path_ink*
- *font_size*
- *lang_code*
- *light_theme*
- *selected_item_ink*
- *sound_volume*
- *unselected_item_ink*
- *vibration_volume*
- *win_rectangle*

which app state variables are finally used by your app project is (fully data-driven) depending on the app state *config variables* detected in all the *config files* that are found/available at run-time of your app. the names of all the available application state variables can be determined with the main app helper method *app_state_keys()*.

if no config-file is provided then this package ensures at least the proper initialization of the following app state variables:

- *font_size*
- *lang_code*
- *light_theme*
- *win_rectangle*

if your application is e.g. supporting a user-defined font size, using the provided/pre-defined app state variable *font_size*, then it has to call the method `change_app_state()` with the argument of *app_state_name* set to *font_size* every time when the user has changed the font size of your app.

Hint: the two built-in app state variables are *flow_id* and *flow_path* will be explained detailed in the next section.

---

the `load_app_states()` method is called on instantiation from the implemented main app class to load the values of all app state variables from the *config files*, and is then calling :meth:~MainAppBase.setup_app_states` for pass them into their corresponding instance attributes.

use the main app instance attribute to read/get the actual value of a single app state variable. the actual values of all app state variables as a dict is determining the method `retrieve_app_states()`, and can be saved into the *config files* for the next app run via the method `save_app_states()` - this could be done e.g. after the app state has changed or at least on quiting the application.

always call the method `change_app_state()` to change an app state value to ensure:

(1) the propagation to any duplicated (observable/bound) framework property and

(2) the event notification of the related (optionally declared) main app instance method.

app state constants

this module is also providing some pre-defined constants that can be optionally used in your application in relation to the app states data store and for the app state config variables *app_state_version*, *font_size* and *light_theme*:

- *APP_STATE_SECTION_NAME*
- *APP_STATE_VERSION_VAR_NAME*
- *MIN_FONT_SIZE*
- *MAX_FONT_SIZE*
- *THEME_LIGHT_BACKGROUND_COLOR*
- *THEME_LIGHT_FONT_COLOR*
- *THEME_DARK_BACKGROUND_COLOR*
- *THEME_DARK_FONT_COLOR*

app state events

there are three types of notification events get fired in relation to the app state variables, using the method names:

- *on_<app_state_name>*: fired if the user of the app is changing the value of an app state variable.
- *on_<app_state_name>_save*: fired if an app state gets saved to the config file.
- *on_app_state_version_upgrade*: fired if the user upgrades a previously installed app to a higher version.

the method name of the app state change notification event consists of the prefix `on_` followed by the variable name of the app state. so e.g. on a change of the *font_size* app state the notification event *on_font_size* will be fired/called (if exists as a method of the main app instance). these events don't provide any event arguments.

the second event gets fired for each app state value just after the app states getting retrieved from the app class instance, and before they get stored into the main config file. the method name of this event includes also the name of the app state with the suffix *_save*, so e.g. for the app state *flow_id* the event method name will result in `on_app_state_flow_id_save()`. this event is providing one event argument with the value of the app state. if the event method returns a value that is not *None* then this value will be stored/saved.

the third event gets fired on app startup when the app got upgraded to a higher version of the app state variable APP_STATE_VERSION_VAR_NAME (*app_state_version*). it will be called providing the version number for each version to upgrade, starting with the version of the previously installed main config file, until the upgrade version of the main config file get reached. so if e.g. the previously installed app state version was 3 and the new version number is 6 then this event will be fired 3 times with the argument 3, 4 and 5. it can be used e.g. to change or add app state variables or to adapt the app environment.

application flow

to control the current state and UX flow (or context) of your application, and to persist it until the next app start, `MainBaseApp` provides two *app state variables*: *flow_id* to store the currently working flow and *flow_path* to store the history of nested flows.

an application flow is represented by an id string that defines three things: (1) the action to enter into the flow, (2) the data or object that gets currently worked on and (3) an optional key string that is identifying/indexing a widget or data item of your application context/flow.

Note: never concatenate a flow id string manually, use the *id_of_flow()* function instead.

---

the flow id is initially an empty string. as soon as the user is starting a new work flow or the current selection your application should call the method `change_flow()` passing the flow id string into the *new_flow_id* argument to change the app flow.

for more complex applications you can specify a path of nested flows. this flow path gets represented by the app state variable *flow_path*, which is a list of flow id strings.

to enter into a deeper/nested flow you simply call `change_flow()` with one of the actions defined in *ACTIONS_EXTENDING_FLOW_PATH*.

to go back to a previous flow in the flow path call `change_flow()` passing one of the actions defined in *ACTIONS_REDUCING_FLOW_PATH*.

application flow change events

the flow actions specified by *ACTIONS_CHANGING_FLOW_WITHOUT_CONFIRMATION* don't need a flow change confirmation event handler:

- *'enter'* or *'leave'* extend/reduce the flow path.
- *'focus'* pass/change the input focus.
- *'suggest'* for autocompletion or other suggestions.

all other flow actions need to confirmed to be changed by *change_flow()*, either by a custom flow change confirmation method/event-handler or by declaring a popup class. the name of the event handler and of the popup class gets determined from the flow id.

Hint: the name of the flow change confirmation method that gets fired when the app want to change the flow (via the method *change_flow()*) gets determined by the function *flow_change_confirmation_event_name()*, whereas the name of the popup class get determined by the function *flow_popup_class_name()*.

---

if the flow-specific change confirmation event handler does not exist or returns in a boolean *False* or *None* then *on_flow_change()* will be called. if this call also returns *False* then the action of the new flow id will be searched within *ACTIONS_CHANGING_FLOW_WITHOUT_CONFIRMATION* and if not found then the flow change will be rejected and *change_flow()* returns *False*.

if in contrary either the flow change confirmation event handler exists and does return *True* or *on_flow_change()* returns True or the flow action of the new flow id is in *ACTIONS_CHANGING_FLOW_WITHOUT_CONFIRMATION* then the flow id and path will be changed accordingly.

after the flow id/path change confirmation the method *change_flow()* checks if the optional event_kwargs key *changed_event_name* got specified and if yes then it calls this method.

finally, if a confirmed flow change results in a *'focus'* flow action then the event *on_flow_widget_focused* will be fired. this event can be used by the GUI framework to set the focus to the widget associated with the new focus flow id.

flow actions *'open'* and *'close'*

to display an instance of a properly named popup class, simply initiate the change the app flow to an appropriate flow id (with an *'open'* flow action). in this case no change confirmation event handler is needed, because *on_flow_change()* is then automatically opening the popup.

when the popup is visible the flow path will be extended with the respective flow id.

calling the *close* method of the popup will hide it. on closing the popup the flow id will be reset and the opening flow id will be removed from the flow path.

all popup classes are providing the events *on_pre_open*, *on_open*, *on_pre_dismiss* and *on_dismiss*. the *on_dismiss* event handler can be used for data validation: returning a non-False value from it will cancel the close.

Hint: see the documentation of each popup class for more details on the features of popup classes (for Kivy apps e.g. *FlowDropDown*, *FlowPopup* or *FlowSelector*).

---

key press events

to provide key press events to the applications that will use the new GUI framework you have to catch the key press events of the framework, convert/normalize them and then call the *key_press_from_framework()* with the normalized modifiers and key args.

the *modifiers* arg is a string that can contain several of the following sub-strings, always in the alphabetic order (like listed below):

- Alt
- Ctrl
- Meta
- Shift

the *key* arg is a string that is specifying the last pressed key. if the key is not representing a single character but a command key, then *key* will be one of the following strings:

- escape
- tab
- backspace
- enter
- del
- enter
- up
- down
- right
- left
- home
- end
- pgup
- pgdown

on call of *key_press_from_framework()* this method will try to dispatch the key press event to your application. first it will check the app instance if it has declared a method with the name *on_key_press_of_<modifiers>_<key>* and if so it will call this method.

if this method does return False (or any other value resulting in False) then method *key_press_from_framework()* will check for a method with the same name in lower-case and if exits it will call this method.

if also the second method does return False, then it will try to call the event method *on_key_press* of the app instance (if exists) with the modifiers and the key as arguments.

if the *on_key_press* method does also return False then *key_press_from_framework()* will finally pass the key press event to the original key press handler of the GUI framework for further processing.

integrate new gui framework

to integrate a new Python GUI framework you have to declare a new class that inherits from *MainAppBase* and implements at least the abstract method *init_app()*.

additionally and to load the resources of the app (after the portions resources got loaded) the event *on_app_build* has to be fired, executing the *MainAppBase.on_app_build()* method. this could be done directly from within *init_app()* or by redirecting one of the events of the app instance of the GUI framework.

a minimal implementation of the *init_app()* method would look like the following:

```
def init_app(self):
    self.call_method('on_app_build')
    return None, None
```

most GUI frameworks are providing classes that need to be instantiated on application startup, like e.g. the instance of the GUI framework app class, the root widget or layout of the main GUI framework window(s). to keep a reference to these instances within your main app class you can use the attributes *framework_app*, *framework_root* and *framework_win* of the class *MainAppBase*.

the initialization of the attributes *framework_app*, *framework_root* and *framework_win* is optional and can be done e.g. within *init_app()* or in the *on_app_build* application event fired later by the framework app instance.

Note: if *framework_win* is set to a window instance, then the window instance has to provide a *close* method, which will be called automatically by the *stop_app()*.

---

a typical implementation of a framework-specific main app class looks like:

```
from new_gui_framework import NewFrameworkApp, MainWindowClassOfNewFramework

class NewFrameworkMainApp(MainAppBase):
    def init_app(self):
        self.framework_app = NewFrameworkAppClass()
        self.framework_win = MainWindowClassOfNewFramework()

        # return callables to start/stop the event loop of the GUI framework
        return self.framework_app.start, self.framework_app.stop
```

in this example the *on_app_build* application event gets fired either from within the *start* method of the framework app instance or by an event provided by the GUI framework.

init_app() will be executed only once at the main app class instantiation. only the main app instance has to initialize the GUI framework to prepare the app startup and has to return at least a callable to start the event loop of the GUI framework.

Hint: although not recommended because of possible namespace conflicts, one could e.g. alternatively integrate the framework application class as a mixin to the main app class.

---

to initiate the app startup the `run_app()` method has to be called from the main module of your app project. *run_app()* will then start the GUI event loop by calling the first method that got returned by *init_app()*.

optional configuration and extension

most of the base implementation helper methods can be overwritten by either the inheriting framework portion or directly by user main app class.

base resources for your gui app

this portion is also providing base resources for commonly used images and sounds.

the image file resources provided by this portion are taken from:

- iconmonstr.

the sound files provides by this portion are taken from:

- Erokia at freesound.org.
- plasterbrain at freesound.org.

Hint: the i18n translation texts of this module are provided by the ae namespace portion *ae.gui_help*.

TODO: implement OS-independent detection of dark/light screen mode and automatic notification on day/night mode switch. - see https://github.com/albertosottile/darkdetect for MacOS, MSWindows and Ubuntu - see https://github.com/kvdroid/Kvdroid/blob/master/kvdroid/tools/darkmode.py for Android

Module Attributes

APP_STATE_SECTION_NAME	config section name to store app state
APP_STATE_VERSION_VAR_NAME	config variable name to store the current application state version
COLOR_BLACK	!= 0/1 to differentiate from framework pure black/white colors
THEME_DARK_BACKGROUND_COLOR	dark theme background color in rgba(0.0 .
THEME_DARK_FONT_COLOR	dark theme font color in rgba(0.0 .
THEME_LIGHT_BACKGROUND_COLOR	light theme background color in rgba(0.0 .
THEME_LIGHT_FONT_COLOR	light theme font color in rgba(0.0 .
MIN_FONT_SIZE	minimum (s.a.
MAX_FONT_SIZE	
ACTIONS_EXTENDING_FLOW_PATH	flow actions that are extending the flow path.
ACTIONS_REDUCING_FLOW_PATH	flow actions that are shrinking/reducing the flow paths.
ACTIONS_CHANGING_FLOW_WITHOUT_CONFIRMATION	flow actions that are processed without the need to be confirmed.
FLOW_KEY_SEP	separator character between flow action/object and flow key
FLOW_ACTION_RE	regular expression detecting invalid characters in flow action string
FLOW_OBJECT_RE	regular expression detecting invalid characters in flow object string
HIDDEN_GLOBALS	tuple of global/module variable names that are hidden in *global_variables()*
PORTIONS_IMAGES	register of image files found in portions/packages at import time
PORTIONS_SOUNDS	register of sound files found in portions/packages at import time
USER_NAME_MAX_LEN	maximal length of a username/id (restricting *user_id*)
AppStateType	app state config variable type
EventKwargsType	change flow event kwargs type
ColorRGB	color red, green and blue parts
ColorRGBA	ink is rgb color and alpha
ColorOrInk	color or ink type

Functions

ellipse_polar_radius(ell_a, ell_b, radian)	calculate the radius from polar for the given ellipse and radian.
ensure_tap_kwargs_refs(init_kwargs, tap_widget)	ensure that the passed widget.__init__ kwargs dict contains a reference to itself within kwargs['tap_kwargs'].
flow_action(flow_id)	determine the action string of a flow_id.
flow_action_split(flow_id)	split flow id string into action part and the rest.
flow_change_confirmation_event_name(flow_id)	determine the name of the event method for the change confirmation of the passed flow_id.
flow_class_name(flow_id, name_suffix)	determine class name for the given flow id and class name suffix.
flow_key(flow_id)	return the key of a flow id.
flow_key_split(flow_id)	split flow id into action with object and flow key.
flow_object(flow_id)	determine the object string of the passed flow_id.
flow_path_id(flow_path[, path_index])	determine the flow id of the newest/last entry in the flow_path.
flow_path_strip(flow_path)	return copy of passed flow_path with all non-enter actions stripped from the end.
flow_popup_class_name(flow_id)	determine name of the Popup class for the given flow id.
id_of_flow(action[, obj, key])	create flow id string.
register_package_images()	call from module scope of the package to register/add image/img resources path.
register_package_sounds()	call from module scope of the package to register/add sound file resources.
replace_flow_action(flow_id, new_action)	replace action in given flow id.
update_tap_kwargs(widget[, popup_kwargs])	update or simulate widget's tap_kwargs property and return the updated dictionary (for kv rule of tap_kwargs).

Classes

MainAppBase(**console_app_kwargs)	abstract base class to implement a GUIApp-conform app class

APP_STATE_SECTION_NAME = 'aeAppState'

config section name to store app state

APP_STATE_VERSION_VAR_NAME = 'app_state_version'

config variable name to store the current application state version

COLOR_BLACK = [0.009, 0.006, 0.003, 1.0]

!= 0/1 to differentiate from framework pure black/white colors

THEME_DARK_BACKGROUND_COLOR = [0.009, 0.006, 0.003, 1.0]

dark theme background color in rgba(0.0 … 1.0)

THEME_DARK_FONT_COLOR = [0.999, 0.996, 0.993, 1.0]

dark theme font color in rgba(0.0 … 1.0)

THEME_LIGHT_BACKGROUND_COLOR = [0.999, 0.996, 0.993, 1.0]

light theme background color in rgba(0.0 … 1.0)

THEME_LIGHT_FONT_COLOR = [0.009, 0.006, 0.003, 1.0]

light theme font color in rgba(0.0 … 1.0)

MIN_FONT_SIZE = 15.0

minimum (s.a. *min_font_size*) and

MAX_FONT_SIZE = 99.0

ACTIONS_EXTENDING_FLOW_PATH = ['add', 'confirm', 'edit', 'enter', 'open', 'show', 'suggest']

flow actions that are extending the flow path.

ACTIONS_REDUCING_FLOW_PATH = ['close', 'leave']

flow actions that are shrinking/reducing the flow paths.

ACTIONS_CHANGING_FLOW_WITHOUT_CONFIRMATION = ['', 'enter', 'focus', 'leave', 'suggest']

flow actions that are processed without the need to be confirmed.

FLOW_KEY_SEP = ':'

separator character between flow action/object and flow key

FLOW_ACTION_RE = re.compile('[a-z0-9]+')

regular expression detecting invalid characters in flow action string

FLOW_OBJECT_RE = re.compile('[A-Za-z0-9_]+')

regular expression detecting invalid characters in flow object string

HIDDEN_GLOBALS = ('ABC', 'abstractmethod', '_add_base_globals', 'Any', '__builtins__', '__cached__', 'Callable', '_d_', 'Dict', '__doc__', '__file__', 'List', '__loader__', 'module_globals', '__name__', 'Optional', '__package__', '__path__', '__spec__', 'Tuple', 'Type', '__version__')

tuple of global/module variable names that are hidden in *global_variables()*

```
PORTIONS_IMAGES = {'add_item':  [RegisteredFile('/home/docs/checkouts/readthedocs.org/
user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/img/add_item.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/add_item.png')], 'close_popup':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/close_popup.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/close_popup.png')], 'copy_node':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/copy_node.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/copy_node.png')], 'cut_node':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/cut_node.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/cut_node.png')], 'delete_item':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/delete_item.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/delete_item.png')], 'drag_item':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/drag_item.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/drag_item.png')], 'edit_font_size':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/enaml_app/img/edit_font_size.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/enaml_app/img/light_1/edit_font_size.png')], 'edit_item':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/edit_item.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/edit_item.png')], 'enter_item':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/enter_item.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/enter_item.png')], 'export_node':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/export_node.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/export_node.png')], 'filter_off':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/filter_off.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/filter_off.png')], 'filter_on':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/filter_on.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/filter_on.png')], 'font_size':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/kivy_user_prefs/img/light_1/font_size.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/kivy_user_prefs/img/font_size.png')], 'help_circled':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/help_circled.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/help_circled.png')], 'help_icon':
[RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/help_icon.png'),
RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/
python3.9/site-packages/ae/gui_app/img/light_1/help_icon.png')], 'icon_view':
```

register of image files found in portions/packages at import time

PORTIONS_SOUNDS = {'added': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/added.wav')], 'debug_draw': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/debug_draw.wav')], 'debug_save': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/debug_save.wav')], 'deleted': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/deleted.wav')], 'edited': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/edited.wav')], 'enter_item': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/enter_item.wav')], 'error': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/error.wav')], 'filter_off': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/filter_off.wav')], 'filter_on': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/filter_on.wav')], 'leave_item': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/leave_item.wav')], 'touched': [RegisteredFile('/home/docs/checkouts/readthedocs.org/user_builds/ae/envs/stable/lib/python3.9/site-packages/ae/gui_app/snd/touched.wav')]}

register of sound files found in portions/packages at import time

USER_NAME_MAX_LEN = 12

maximal length of a username/id (restricting *user_id*)

AppStateType

app state config variable type

alias of Dict[str, Any]

EventKwargsType

change flow event kwargs type

alias of Dict[str, Any]

ColorRGB

color red, green and blue parts

alias of Union[Tuple[float, float, float], List[float]]

ColorRGBA

ink is rgb color and alpha

alias of Union[Tuple[float, float, float, float], List[float]]

ColorOrInk

color or ink type

alias of Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]]

ellipse_polar_radius(*ell_a*, *ell_b*, *radian*)

calculate the radius from polar for the given ellipse and radian.

Parameters

- **ell_a** (float) – ellipse x-radius.
- **ell_b** (float) – ellipse y-radius.
- **radian** (float) – radian of angle.

Return type
float

Returns
ellipse radius at the angle specified by radian.

ensure_tap_kwargs_refs(*init_kwargs*, *tap_widget*)

ensure that the passed widget.__init__ kwargs dict contains a reference to itself within kwargs['tap_kwargs'].

Parameters

- **init_kwargs** (Dict[str, Any]) – kwargs of the widgets __init__ method.
- **tap_widget** (Any) – reference to the tap widget.

this alternative version is only 10 % faster but much less clean than the current implementation:

```
if 'tap_kwargs' not in init_kwargs:
    init_kwargs['tap_kwargs'] = {}
tap_kwargs = init_kwargs['tap_kwargs']

if 'tap_widget' not in tap_kwargs:
    tap_kwargs['tap_widget'] = tap_widget

if 'popup_kwargs' not in tap_kwargs:
    tap_kwargs['popup_kwargs'] = {}
popup_kwargs = tap_kwargs['popup_kwargs']
if 'opener' not in popup_kwargs:
    popup_kwargs['opener'] = tap_kwargs['tap_widget']
```

flow_action(*flow_id*)

determine the action string of a flow_id.

Parameters
flow_id (str) – flow id.

Return type
str

Returns
flow action string.

flow_action_split(*flow_id*)

split flow id string into action part and the rest.

Parameters
flow_id (str) – flow id.

Return type
Tuple[str, str]

Returns
tuple of (flow action string, flow obj and key string)

flow_change_confirmation_event_name(*flow_id*)

determine the name of the event method for the change confirmation of the passed flow_id.

Parameters
flow_id¶ (str) – flow id.

Return type
str

Returns
tuple with 2 items containing the flow action and the object name (and id).

flow_class_name(*flow_id*, *name_suffix*)

determine class name for the given flow id and class name suffix.

Parameters

- **flow_id**¶ (str) – flow id.
- **name_suffix**¶ (str) – class name suffix.

Return type
str

Returns
name of the class. please note that the flow action *open* will not be added to the returned class name.

flow_key(*flow_id*)

return the key of a flow id.

Parameters
flow_id¶ (str) – flow id string.

Return type
str

Returns
flow key string.

flow_key_split(*flow_id*)

split flow id into action with object and flow key.

Parameters
flow_id¶ (str) – flow id to split.

Return type
Tuple[str, str]

Returns
tuple of (flow action and object string, flow key string).

flow_object(*flow_id*)

determine the object string of the passed flow_id.

Parameters
flow_id¶ (str) – flow id.

Return type
str

Returns
flow object string.

`flow_path_id`(*flow_path*, *path_index=-1*)

determine the flow id of the newest/last entry in the flow_path.

Parameters

- **flow_path** (List[str]) – flow path to get the flow id from.
- **path_index** (int) – index in the flow_path.

Return type
str

Returns
flow id string or empty string if flow path is empty or index does not exist.

`flow_path_strip`(*flow_path*)

return copy of passed flow_path with all non-enter actions stripped from the end.

Parameters
flow_path (List[str]) – flow path list to strip.

Return type
List[str]

Returns
stripped flow path copy.

`flow_popup_class_name`(*flow_id*)

determine name of the Popup class for the given flow id.

Parameters
flow_id (str) – flow id.

Return type
str

Returns
name of the Popup class. please note that the action *open* will not be added to the returned class name.

`id_of_flow`(*action*, *obj=''*, *key=''*)

create flow id string.

Parameters

- **action** (str) – flow action string.
- **obj** (str) – flow object (defined by app project).
- **key** (str) – flow index/item_id/field_id/... (defined by app project).

Return type
str

Returns
complete flow_id string.

`register_package_images`()

call from module scope of the package to register/add image/img resources path.

no parameters needed because we use here *stack_var()* helper function to determine the module file path via the *__file__* module variable of the caller module in the call stack. in this call we have to overwrite the default value (*SKIPPED_MODULES*) of the *skip_modules* parameter to not skip ae portions that are providing package

resources and are listed in the *SKIPPED_MODULES*, like e.g. *ae.gui_app* and *ae.gui_help* (passing empty string '' to overwrite default skip list).

register_package_sounds()

call from module scope of the package to register/add sound file resources.

no parameters needed because we use here *stack_var()* helper function to determine the module file path via the *__file__* module variable of the caller module in the call stack. in this call we have to overwrite the default value (*SKIPPED_MODULES*) of the *skip_modules* parameter to not skip ae portions that are providing package resources and are listed in the *SKIPPED_MODULES*, like e.g. *ae.gui_app ae.gui_help* (passing empty string '' to overwrite default skip list).

replace_flow_action(*flow_id*, *new_action*)

replace action in given flow id.

Parameters

- **flow_id** (str) – flow id.
- **new_action** (str) – action to be set/replaced within passed flow id.

Returns
flow id with new action and object/key from passed flow id.

update_tap_kwargs(*widget*, *popup_kwargs=None*, ***tap_kwargs*)

update or simulate widget's tap_kwargs property and return the updated dictionary (for kv rule of tap_kwargs).

Parameters

- **widget** (Any) – widget with tap_kwargs property to be updated.
- **popup_kwargs** (Optional[Dict[str, Any]]) – dict with items to update popup_kwargs key of tap_kwargs
- **tap_kwargs** – additional tap_kwargs items to update.

Return type
Dict[str, Any]

Returns

class MainAppBase(***console_app_kwargs*)

Bases: *ConsoleApp*, ABC

abstract base class to implement a GUIApp-conform app class

app_state_version: int = 0

version number of the app state variables in <config>.ini

flow_id: str = ''

id of the current app flow (entered by the app user)

flow_id_ink: Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]] = [0.99, 0.99, 0.69, 0.69]

rgba color for flow id / drag&drop node placeholder

flow_path_ink: Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]] = [0.99, 0.99, 0.39, 0.48]

rgba color for flow_path/drag&drop item placeholder

font_size: float = 21.0

font size used toolbar and flow screens

lang_code: `str = ''`

optional language code (e.g. 'es_ES' for Spanish)

light_theme: `bool = False`

True=light theme/background, False=dark theme

selected_item_ink: `Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]] = [0.69, 1.0, 0.39, 0.18]`

rgba color for list items (selected)

unselected_item_ink: `Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]] = [0.39, 0.39, 0.39, 0.18]`

rgba color for list items (unselected)

sound_volume: `float = 0.12`

sound volume of current app (0.0=mute, 1.0=max)

vibration_volume: `float = 0.3`

vibration volume of current app (0.0=mute, 1.0=max)

win_rectangle: `tuple = (0, 0, 1920, 1080)`

window coordinates (x, y, width, height)

framework_app: `Any = None`

app class instance of the used GUI framework

framework_win: `Any = None`

window instance of the used GUI framework

framework_root: `Any = None`

app root layout widget

image_files: `Optional[FilesRegister] = None`

image/icon files

sound_files: `Optional[FilesRegister] = None`

sound/audio files

__init__(***console_app_kwargs*)

create instance of app class.

Parameters

console_app_kwargs – kwargs to be passed to the __init__ method of `ConsoleApp`.

_exit_code: `int`

init by stop_app() and passed onto OS by run_app()

_last_focus_flow_id: `str`

id of the last valid focused window/widget/item/context

flow_path: `List[str]`

list of flow ids, reflecting recent user actions

_start_event_loop: `Optional[Callable]`

callable to start event loop of GUI framework

_stop_event_loop: `Optional[Callable]`

callable to start event loop of GUI framework

_init_default_user_cfg_vars()

init user default config variables.

override this method to add module-/app-specific config vars that can be set individually per user.

abstract init_app(*framework_app_class=None*)

initialize framework app instance and root window/layout, return GUI event loop start/stop methods.

Parameters

framework_app_class (Optional[Any]) – class to create app instance (optionally extended by app project).

Return type

Tuple[Optional[Callable], Optional[Callable]]

Returns

tuple of two callable, the 1st to start and the 2nd to stop/exit the GUI event loop.

app_state_keys()

determine current config variable names/keys of the app state section *APP_STATE_SECTION_NAME*.

Return type

Tuple[str, ...]

Returns

tuple of all app state item keys (config variable names).

static backup_config_resources()

backup config files and image/sound/translations resources to {ado}<now_str>.

config files are collected from {ado}, {usr} or {cwd} (the first found file name only - see/sync-with *ae.console.ConsoleApp.add_cfg_files()*).

resources are copied from {ado} or {cwd} (only the first found resources root path).

Return type

str

change_app_state(*app_state_name*, *state_value*, *send_event=True*, *old_name=''*)

change app state to *state_value* in self.<app_state_name> and app_states dict.

Parameters

- **app_state_name** (str) – name of the app state to change.
- **state_value** (Any) – new value of the app state to change.
- **send_event** (bool) – pass False to prevent send/call of the main_app.on_<app_state_name> event.
- **old_name** (str) – pass to add state to the main config file: old state name to rename/migrate or *UNSET* to only add a new app state variable with the name specified in *app_state_name*.

change_observable(*name*, *value*, *is_app_state=False*)

change observable attribute/member/property in framework_app instance (and shadow copy in main app).

Parameters

- **name** (str) – name of the observable attribute/member or key of an observable dict property.
- **value** (Any) – new value of the observable.

- **is_app_state**¶ (bool) – pass True for an app state observable.

change_flow(*new_flow_id*, ***event_kwargs*)

try to change/switch the current flow id to the value passed in *new_flow_id*.

Parameters

- **new_flow_id**¶ (str) – new flow id (maybe overwritten by flow change confirmation event handlers by assigning a flow id to event_kwargs['flow_id']).
- **event_kwargs**¶ – optional args to pass additional data or info onto and from the flow change confirmation event handler.

  the following keys are currently supported/implemented by this module/portion (additional keys can be added by the modules/apps using this method):

  – *changed_event_name*: optional main app event method name to be called if the flow got confirmed and changed.

  – *count*: optional number used to render a pluralized help text for this flow change (this number gets also passed to the help text formatter by/in *change_flow()*).

  – *edit_widget*: optional widget instance for edit/input.

  – *flow_id*: process *flow_path* as specified by the *new_flow_id* argument, but then overwrite this flow id with this event arg value to set *flow_id*.

  – *popup_kwargs*: optional dict passed to the Popup *__init__* method, like e.g. dict(opener=opener_widget_of_popup, data=...).

  – *popups_to_close*: optional sequence of widgets to be closed by this method after flow change got confirmed.

  – 'reset_last_focus_flow_id': pass *True* to reset the last focus flow id, pass *False* or *None* to ignore the last focus id (and not use to set flow id) or pass a flow id string value to change the last focus flow id to the passed value.

  – *tap_widget*: optional tapped button widget instance (initiating this flow change).

  some of these keys get specified directly on the call of this method, e.g. via *tap_kwargs* or *tap_kwargs*, where others get added by the flow change confirmation handlers/callbacks.

Return type
bool

Returns

True if flow got confirmed by a declared custom flow change confirmation event handler (either event method or Popup class) of the app and changed accordingly, else False.

some flow actions are handled internally independent of the return value of a custom event handler, like e.g. *'enter'* or *'leave'* will always extend or reduce the flow path and the action *'focus'* will give the indexed widget the input focus (these exceptions are configurable via *ACTIONS_CHANGING_FLOW_WITHOUT_CONFIRMATION*).

static class_by_name(*class_name*)

search class name in framework modules as well as in app main.py to return class object.

Parameters
class_name¶ (str) – name of the class.

Return type
Optional[Type]

Returns
class object with the specified class name or *UNSET* if not found.

`static dpi_factor()`

dpi scaling factor - override if the used GUI framework supports dpi scaling.

Return type
float

`close_popups(classes=())`

close all opened popups (starting with the foremost popup).

Parameters
classes (tuple) – optional class filter - if not passed then only the first foremost widgets underneath the app win with an *open* method will be closed. pass tuple to restrict found popup widgets to certain classes. like e.g. by passing (Popup, DropDown, FlowPopup) to get all popups of an app (in Kivy use Factory.WidgetClass if widget is declared only in kv lang).

`find_image(image_name, height=32.0, light_theme=True)`

find best fitting image in img app folder (see also *img_file()* for easier usage).

Parameters
- **image_name** (str) – name of the image (file name without extension).
- **height** (float) – preferred height of the image/icon.
- **light_theme** (bool) – preferred theme (dark/light) of the image.

Return type
Optional[*RegisteredFile*]

Returns
image file object (RegisteredFile/CachedFile) if found else None.

`find_sound(sound_name)`

find sound by name.

Parameters
sound_name (str) – name of the sound to search for.

Return type
Optional[*RegisteredFile*]

Returns
cached sound file object (RegisteredFile/CachedFile) if sound name was found else None.

`find_widget(match)`

search the widget tree returning the first matching widget in reversed z-order (top-/foremost first).

Parameters
match (Callable[[Any], bool]) – callable called with the widget as argument, returning True if widget matches.

Return type
Optional[Any]

Returns
first found widget in reversed z-order (top-most widget first).

`flow_path_action`(*flow_path=None*, *path_index=-1*)

determine the action of the last (newest) entry in the flow_path.

Parameters

- **flow_path** (Optional[List[str]]) – optional flow path to get the flow action from (default=self.flow_path).
- **path_index** (int) – optional index in the flow_path (default=-1).

Return type
str

Returns
flow action string or empty string if flow path is empty or index does not exist.

`global_variables`(***patches*)

determine generic/most-needed global variables to evaluate expressions/macros.

Parameters
patches – dict of variable names and values to add/replace on top of generic globals.

Return type
Dict[str, Any]

Returns
dict of global variables patched with *patches*.

`img_file`(*image_name*, *font_size=None*, *light_theme=None*)

shortcutting *find_image()* method w/o bound property to get image file path.

Parameters

- **image_name** (str) – image name (file name stem).
- **font_size** (Optional[float]) – optional font size in pixels.
- **light_theme** (Optional[bool]) – optional theme (True=light, False=dark).

Return type
str

Returns
file path of image file or empty string if image file not found.

`key_press_from_framework`(*modifiers*, *key*)

dispatch key press event, coming normalized from the UI framework.

Parameters

- **modifiers** (str) – modifier keys.
- **key** (str) – key character.

Return type
bool

Returns
True if key got consumed/used else False.

`load_app_states`()

load application state from the config files to prepare app.run_app

load_images()

load images from app folder img.

load_sounds()

load audio sounds from app folder snd.

load_translations(*lang_code*)

load translation texts for the passed language code.

Parameters
lang_code¶ (str) – the new language code to be set (passed as flow key). empty on first app run/start.

mix_background_ink()

remix background ink if one of the basic back colours change.

on_app_build()

default/fallback flow change confirmation event handler.

on_app_exit()

default/fallback flow change confirmation event handler.

on_app_init()

default/fallback flow change confirmation event handler.

on_app_quit()

default/fallback flow change confirmation event handler.

on_app_run()

default/fallback flow change confirmation event handler.

on_app_started()

app initialization event - the last one on app startup.

on_debug_level_change(*level_name*, *_event_kwargs*)

debug level app state change flow change confirmation event handler.

Parameters

- **level_name**¶ (str) – the new debug level name to be set (passed as flow key).
- **_event_kwargs**¶ (Dict[str, Any]) – unused event kwargs.

Return type
bool

Returns
True to confirm the debug level change.

on_flow_change(*flow_id*, *event_kwargs*)

checking if exists a Popup class for the new flow and if yes then open it.

Parameters

- **flow_id**¶ (str) – new flow id.
- **event_kwargs**¶ (Dict[str, Any]) – optional event kwargs; the optional item with the key *popup_kwargs* will be passed onto the *__init__* method of the found Popup class.

Return type
bool

Returns
True if Popup class was found and displayed.

this method is mainly used as the last fallback clicked flow change confirmation event handler of a FlowButton.

`on_flow_id_ink()`

redirect flow id back ink app state color change event handler to actualize mixed_back_ink.

`on_flow_path_ink()`

redirect flow path back ink app state color change event handler to actualize mixed_back_ink.

`static on_flow_popup_close(_flow_key, _event_kwargs)`

default popup close handler of FlowPopup widget, ensuring update of *flow_path*.

Parameters

- **_flow_key** (str) – unused flow key.
- **_event_kwargs** (Dict[str, Any]) – unused popup args.

Return type
bool

Returns
always returning True.

`on_lang_code_change(lang_code, _event_kwargs)`

language app state change flow change confirmation event handler.

Parameters

- **lang_code** (str) – the new language code to be set (passed as flow key). empty on first app run/start.
- **_event_kwargs** (Dict[str, Any]) – unused event kwargs.

Return type
bool

Returns
True to confirm the language change.

`on_light_theme_change(_flow_key, event_kwargs)`

app theme app state change flow change confirmation event handler.

Parameters

- **_flow_key** (str) – flow key.
- **event_kwargs** (Dict[str, Any]) – event kwargs with key *'light_theme'* containing True|False for light|dark theme.

Return type
bool

Returns
True to confirm change of flow id.

`on_selected_item_ink()`

redirect selected item back ink app state color change event handler to actualize mixed_back_ink.

on_unselected_item_ink()

redirect unselected item back ink app state color change event handler to actualize mixed_back_ink.

on_user_add(*user_name*, *event_kwargs*)

called on close of UserNameEditorPopup to check user input and create/register the current os user.

Parameters

- **user_name** (str) – new user id.
- **event_kwargs** (Dict[str, Any]) – optionally pass True in the *unique_user_name* key to prevent duplicate usernames.

Return type
bool

Returns
True if user got registered else False.

play_beep()

make a short beep sound, should be overwritten by GUI framework.

play_sound(*sound_name*)

play audio/sound file, should be overwritten by GUI framework.

Parameters
sound_name (str) – name of the sound to play.

play_vibrate(*pattern=(0.0, 0.09, 0.21, 0.3, 0.09, 0.09, 0.21, 0.09)*)

play vibrate pattern, should be overwritten by GUI framework.

Parameters
pattern (Tuple) – optional tuple of pause and vibrate time sequence - use error pattern if not passed.

open_popup(*popup_class*, ***popup_kwargs*)

open Popup/DropDown, calling the *open*/*show* method of the instance created from the passed popup class.

Parameters

- **popup_class** (Type) – class of the Popup/DropDown widget/window.
- **popup_kwargs** – args to instantiate and show/open the popup.

Return type
Any

Returns
created and displayed/opened popup class instance.

Hint: overwrite this method if framework is using different method to open popup window or if a widget in the Popup/DropDown need to get the input focus.

---

popups_opened(*classes=()*)

determine all popup-like container widgets that are currently opened.

Parameters
classes (Tuple) – optional class filter - if not passed then only the widgets underneath win/root with an *open* method will be added. pass tuple to restrict found popup widgets to certain classes. like e.g. by passing (Popup, DropDown, FlowPopup) to get all popups of an ae/Kivy app (in Kivy use Factory.WidgetClass if widget is declared only in kv lang).

Return type
List

Returns
list of the foremost opened/visible popup class instances (children of the app window), matching the `classes` or having an *open* method, ordered by their z-coordinate (most front widget first).

`retrieve_app_states()`

determine the state of a running app from the main app instance and return it as dict.

Return type
Dict[str, Any]

Returns
dict with all app states available in the config files.

`run_app()`

startup main and framework applications.

`save_app_states()`

save app state in config file.

Return type
str

Returns
empty string if app status could be saved into config files else error message.

`setup_app_states(app_states)`

put app state variables into main app instance to prepare framework app.run_app.

Parameters
app_states (Dict[str, Any]) – dict of app states.

`show_message(message, title='', is_error=True)`

display (error) message popup to the user.

Parameters

- **message** (str) – message string to display.
- **title** (str) – title of message box.
- **is_error** (bool) – pass False to not emit error tone/vibration.

`stop_app(exit_code=0)`

quit this application.

Parameters
exit_code (int) – optional exit code.

`upgraded_config_app_state_version()`

determine app state version of an app upgrade.

Return type
int

Returns
value of app state variable APP_STATE_VERSION_VAR_NAME if the app got upgraded (and has a config file from a previous app installation), else 0.

widget_by_attribute(*att_name*, *att_value*)

determine the first (top-most) widget having the passed attribute name and value.

Parameters

- **att_name** (str) – name of the attribute of the searched widget.
- **att_value** (str) – attribute value of the searched widget.

Return type
Optional[Any]

Returns
widget that has the specified attribute with the specified value or None if not found.

widget_by_flow_id(*flow_id*)

determine the first (top-most) widget having the passed flow_id.

Parameters
flow_id (str) – flow id value of the searched widget's *tap_flow_id*/*focus_flow_id* attribute.

Return type
Optional[Any]

Returns
widget that has a *tap_flow_id*/*focus_flow_id* attribute with the value of the passed flow id or None if not found.

widget_by_app_state_name(*app_state_name*)

determine the first (top-most) widget having the passed app state name (app_state_name).

Parameters
app_state_name (str) – app state name of the widget's *app_state_name* attribute.

Return type
Optional[Any]

Returns
widget that has a *app_state_name* attribute with the passed app state name or None if not found.

widget_children(*wid*, *only_visible=False*)

determine the children of widget or its container (if exists) in z-order (top-/foremost first).

Parameters

- **wid** (Any) – widget to determine the children from.
- **only_visible** (bool) – pass True to only return visible widgets.

Return type
List

Returns
list of children widgets of the passed widget.

static widget_pos(*wid*)

return the absolute window x and y position of the passed widget.

Parameters
wid – widget to determine the position of.

Return type
Tuple[float, float]

Returns
tuple of x and y screen/window coordinate.

`widgets_enclosing_rectangle`(*widgets*)

calculate the minimum bounding rectangle all the passed widgets.

Parameters
widgets (Union[list, tuple]) – list/tuple of widgets to determine the minimum bounding rectangle for.

Return type
Tuple[float, float, float, float]

Returns
tuple of floats with the x, y, width, height values of the bounding rectangle.

`static widget_size`(*wid*)

return the size (width and height) in pixels of the passed widget.

Parameters
wid – widget to determine the size of.

Return type
Tuple[float, float]

Returns
tuple of width and height in pixels.

`static widget_visible`(*wid*)

determine if the passed widget is visible (has width and height and (visibility or opacity) set).

Parameters
wid (Any) – widget to determine visibility of.

Return type
bool

Returns
True if widget is visible (or visibility cannot be determined), False if hidden.

`win_pos_size_change`(**win_pos_size*)

screen resize handler called on window resize or when app will exit/stop via closed event.

Parameters
win_pos_size – window geometry/coordinates: x, y, width, height.

4.28 ae.gui_help

4.28.1 main app base class with context help for flow and app state changes

the class *HelpAppBase* provided by this namespace portion is extending your application with a context-sensitive help functionality.

the data-driven approach allows ad-hoc-changes of your app's help texts without the need of code changes or recompilation. this gets achieved within *HelpAppBase* by overriding the main app class methods *change_flow()* and *change_app_state()*.

so to add help support to the widgets of your app you only need to add/provide the help texts with a help id that is matching the value of the `help_id` attribute of the widget you need help for.

additionally, you can provide a separate i18n translation message file for each of the supported languages to make your help texts multilingual.

help layout implementation example

HelpAppBase inherits from *MainAppBase* while still being independent of the used GUI framework/library.

Note: the user interface for this help system has to be provided externally on top of this module. it can either be implemented directly in your app project or in a separate framework-specific module.

use *HelpAppBase* as base class of the GUI framework specific main application class and implement the abstract methods *init_app()* and *ensure_top_most_z_index()*:

```
from ae.gui_help import HelpAppBase

class MyMainApp(HelpAppBase):
    def init_app(self, framework_app_class=None):
        self.framework_app = framework_app_class()
        ...
        return self.framework_app.run, self.framework_app.stop

    def ensure_top_most_z_index(self, widget):
        framework_method_to_push_widget_to_top_most(widget)
        ...
```

to activate the help mode the widget to display the help texts have to be assigned to the main app attribute *help_layout* and to the framework app property *help_layout* via the *change_observable()* method:

```
main_app.change_observable('help_layout', HelpScreenContainerOrWindow())
```

the *help_layout* property is also used as a flag of the help mode activity. by assigning *None* to these attributes the help mode will be deactivated:

```
main_app.change_observable('help_layout', None)
```

use the attribute *help_activator* to provide and store a widget that allows the user to toggle the help mode activation. the *help_display()* is using it as fallback widget if no help target (or widget to be explained) got found.

Hint: the de-/activation method *help_activation_toggle()* together with the classes *HelpBehavior*, *HelpToggler* and *Tooltip* are demonstrating a typical implementation of help activator and help text tooltip widgets.

additional helper functions

the helper functions *anchor_layout_x()*, *anchor_layout_y()*, *anchor_points()* and *anchor_spec()* are calculating framework-independent the position and direction of the targeting tooltip anchor arrow and its layout box.

flow change context message id

the message id to identify the help texts for each flow button is composed by the *id_of_flow_help()*, using the prefix marker string defined by the module variable *FLOW_HELP_ID_PREFIX* followed by the flow id of the flow widget.

for example the message id for a flow button with the flow action *'open'*, the object *'item'* and the (optional) flow key *'123456'* is resulting in the following help text message id:

```
'help_flow#open_item:123456'
```

if there is no need for a detailed message id that is taking the flow key into account, then simply create a help text message id without the flow key.

the method *help_display()* does first search for a message id including the flow key in the available help text files and if not found it will automatically fall back to use a message id without the flow key:

```
'help_flow#open_item'
```

Hint: more information regarding the flow id you find in the doc string of the module *ae.gui_app* in the section *application flow*.

---

application state change context message id

the message ids for app state change help texts are using the prefix marker string defined by the module variable *APP_STATE_HELP_ID_PREFIX*, followed by the name of the app state and are composed via the method *id_of_state_help()*.

pluralize-able help texts

each message id can optionally have several help texts for their pluralization. for that simply add a *count* item to the *help_vars* property of the help target widget and then define a help text for the all the possible count cases in your message text file like shown in the following example:

```
{
    'message_id': {
                    'zero':      "help text if {count} == 0",    # {count} will be␣
↪replaced with `'0'`
                    'one':       "help text if count == 1",
                    'many':      "help text if count > 1",
                    'negative':  "help text if count < 0",
                    '':          "fallback help text if count is None",
                    },
   ...
}
```

the provided *count* value can also be included/displayed in the help text, like shown in the *'zero'* count case of the example.

pre- and post-change help texts

to display a different help message before and after the change of the flow id or the app state define a message dict with the keys *''* (an empty string) and *'after'* like shown in the following example:

```
{
    'message_id': {
                   '':      "help text displayed before the flow/app-state change.",
                   'after': "help text displayed after the flow/app-state change",
                   },
   ...
}
```

if you want to move/change the help target to another widget after a change, then use instead of *'after'* the *'next_help_id'* message dict key:

```
{
    'message_id': {
                   '':              "help text before the change",
                   'next_help_id':  "help_flow#next_flow_id",
                   },
   ...
}
```

in this case the help target will automatically change to the widget specified by the flow id in the *'next_help_id'* key, if the user was tapping the second time on the first/initial help target widget.

i18n help texts

the displayed help messages related to the message id will automatically get translated into the default language of the current system/environment.

the declaration and association of message ids and their related help messages is done with the help of the namespace portion *ae.i18n*.

Hint: more details on these and other features of this help system, e.g. the usage of f-strings in the help texts, is documented in the doc string of the *ae.i18n* module.

a more complex example app demonstrating the features of this context help system can be found in the repository of the kivy lisz demo app.

4.28.2 app tours

the following classes provided by this portion building a solid fundament to implement tours for your app:

- *TourBase* - abstract base class of all app tours.
- *TourDropdownFromButton* - abstract base class for tours on dropdowns.
- *OnboardingTour* - minimal app onboarding tour, extendable with app specific tour pages.
- *UserPreferencesTour* - minimal user preferences dropdown tour.

app tour start and stop events

the following main app event methods get called (if they exist) in relation to the start/stop of an app tour:

- *on_tour_init*: fired when the app tour instance got initialized and the app states backup got saved.
- *on_tour_start*: fired after tour start() method get called; delay id configurable via *tour_start_delay* page data.
- *on_tour_exit*: fired after an app tour got finished and the app states got restored to the values of the tour start. fired delayed letting UI events get processed; delay seconds configurable via the *tour_exit_delay* page data value.

UI-specific implementation

to complete the implementation of the app tours, the UI-specific framework has to provide a tour layout class, which is highlighting the widget explained and to display a tooltip and the tour page texts.

Hint: the *TourOverlay* class is a quite complete implementation of a tour layout class.

optionally for the 'user_registration' page of the *OnboardingTour* an open username editor flow has to be implemented.

Module Attributes

CLOSE_POPUP_FLOW_ID	flow id to close opened dropdown/popup
APP_STATE_HELP_ID_PREFIX	message id prefix for app state change help texts
FLOW_HELP_ID_PREFIX	message id prefix for flow change help texts
TOUR_PAGE_HELP_ID_PREFIX	message id prefix of tour page text/dict
TOUR_START_DELAY_DEF	default value of tour start delay in seconds
TOUR_EXIT_DELAY_DEF	default value of tour exit delay in seconds
IGNORED_HELP_FLOWS	tuple of flow ids never search/show help text for
IGNORED_HELP_STATES	tuple of app state names never searched help for
REGISTERED_TOURS	map(name: class) of all registered tour classes
AnchorSpecType	(see return value of *anchor_spec()*)
ExplainedMatcherType	single explained widget matcher type

Functions

anchor_layout_x(anchor_spe, layout_width, ...)	calculate x position of the layout box of an anchor.
anchor_layout_y(anchor_spe, layout_height, ...)	calculate y position of the layout box of an anchor.
anchor_points(font_size, anchor_spe)	recalculate points of the anchor triangle drawing.
anchor_spec(wid_x, wid_y, wid_width, ...)	calculate anchor center pos (x, y) and anchor direction to the targeted widget.
help_id_tour_class(help_id)	determine the tour class if passed help id has attached tour pages.
help_sub_id(help_id)	determine sub id (flow id, tour id or app state name) of the current/specified/passed help id.
id_of_flow_help(flow_id)	compose help id for specified flow id.
id_of_state_help(app_state_name)	compose help id for app state name/key.
id_of_tour_help(page_id)	compose help id for specified tour page id.
register_tour_class(tour_class)	register app tour class.
tour_help_translation(page_id)	determine help translation for the passed page id (flow id or app state name).
tour_id_class(tour_id)	determine the tour class of the passed tour id.
translation_short_help_id(help_id)	check if a help text exists for the passed help id.
widget_page_id(wid)	determine tour page id of passed widget.

Classes

HelpAppBase(**console_app_kwargs)	main app help base class.
OnboardingTour(main_app)	onboarding tour for first app start.
TourBase(main_app)	abstract tour base class, automatically registering sub-classes as app tours.
TourDropdownFromButton(main_app)	generic tour base class to auto-explain a dropdown menu, starting with the button opening the dropdown.
UserPreferencesTour(main_app)	user preferences menu tour.

CLOSE_POPUP_FLOW_ID = 'close_flow_popup'

flow id to close opened dropdown/popup

APP_STATE_HELP_ID_PREFIX = 'help_app_state#'

message id prefix for app state change help texts

FLOW_HELP_ID_PREFIX = 'help_flow#'

message id prefix for flow change help texts

TOUR_PAGE_HELP_ID_PREFIX = 'tour_page#'

message id prefix of tour page text/dict

TOUR_START_DELAY_DEF = 0.15

default value of tour start delay in seconds

TOUR_EXIT_DELAY_DEF = 0.45

default value of tour exit delay in seconds

IGNORED_HELP_FLOWS = ('close_flow_popup',)

tuple of flow ids never search/show help text for

IGNORED_HELP_STATES = ('flow_id', 'flow_path', 'win_rectangle')

tuple of app state names never searched help for

REGISTERED_TOURS = {'AnimatedOnboardingTour': <class 'ae.kivy.tours.AnimatedOnboardingTour'>, 'OnboardingTour': <class 'ae.gui_help.OnboardingTour'>, 'SideloadingMenuTour': <class 'ae.kivy_sideloading.SideloadingMenuTour'>, 'TourDropdownFromButton': <class 'ae.gui_help.TourDropdownFromButton'>, 'UserPreferencesTour': <class 'ae.gui_help.UserPreferencesTour'>}

map(name: class) of all registered tour classes

AnchorSpecType

(see return value of *anchor_spec()*)

alias of Tuple[float, float, str]

ExplainedMatcherType

single explained widget matcher type

alias of Union[Callable[[Any], bool], str]

anchor_layout_x(*anchor_spe*, *layout_width*, *win_width*)

calculate x position of the layout box of an anchor.

Parameters

- **anchor_spe** (Tuple[float, float, str]) – *AnchorSpecType* instance (*anchor_spec()* return) with anchor position/direction.
- **layout_width** (float) – anchor layout width.
- **win_width** (float) – app window width.

Return type
float

Returns
absolute x coordinate within the app window of anchor layout.

anchor_layout_y(*anchor_spe*, *layout_height*, *win_height*)

calculate y position of the layout box of an anchor.

Parameters

- **anchor_spe** (Tuple[float, float, str]) – *AnchorSpecType* tuple with anchor position and direction.
- **layout_height** (float) – anchor layout height.
- **win_height** (float) – app window height.

Return type
float

Returns
absolute y coordinate in the app window of anchor layout.

anchor_points(*font_size*, *anchor_spe*)

recalculate points of the anchor triangle drawing.

Parameters

- **font_size** (float) – font_size to calculate size (radius == hypotenuse / 2) of the anchor triangle.
- **anchor_spe** (Tuple[float, float, str]) – anchor specification tuple: x/y coordinates and direction - see *anchor_spec()* return.

Return type
Tuple[float, ...]

Returns
tuple of the three x and y coordinates of the anchor triangle edges.

anchor_spec(*wid_x*, *wid_y*, *wid_width*, *wid_height*, *win_width*, *win_height*)

calculate anchor center pos (x, y) and anchor direction to the targeted widget.

Parameters

- **wid_x** (float) – absolute x coordinate in app window of targeted widget.
- **wid_y** (float) – absolute y coordinate in app window of targeted widget.
- **wid_width** (float) – width of targeted widget.
- **wid_height** (float) – height of targeted widget.
- **win_width** (float) – app window width.
- **win_height** (float) – app window height.

Return type
Tuple[float, float, str]

Returns

tooltip anchor specification tuple (*AnchorSpecType*) with the three items:

- anchor_x (anchor center absolute x coordinate in window),
- anchor_y (anchor center absolute y coordinate in window) and
- anchor_dir (anchor direction: 'r'=right, 'i'=increase-y, 'l'=left, 'd'=decrease-y)

Note: the direction in the y-axis got named increase for higher y values and *decrease* for lower y values to support different coordinate systems of the GUI frameworks.

e.g. Kivy has the y-axis zero value at the bottom of the app window, whereas in enaml/Qt it is at the top.

---

help_id_tour_class(*help_id*)

determine the tour class if passed help id has attached tour pages.

Parameters
help_id (str) – help id to determine the tour class from.

Return type
Optional[Any]

Returns
tour class of an existing tour for the passed help id or None if no associated tour exists.

help_sub_id(*help_id*)

determine sub id (flow id, tour id or app state name) of the current/specified/passed help id.

opposite of *id_of_flow_help()* / *id_of_state_help()* / *id_of_tour_help()*.

Parameters
help_id¶ (str) – help id to extract the sub id from.

Return type
str

Returns
flow id, tour id, app state name or empty string if help id does not contain a sub id.

id_of_flow_help(*flow_id*)

compose help id for specified flow id.

Parameters
flow_id¶ (str) – flow id to make help id for.

Return type
str

Returns
help id for the specified flow id.

id_of_state_help(*app_state_name*)

compose help id for app state name/key.

Parameters
app_state_name¶ (str) – name of the app state variable.

Return type
str

Returns
help id for the specified app state.

id_of_tour_help(*page_id*)

compose help id for specified tour page id.

Parameters
page_id¶ (str) – tour page id to make help id for.

Return type
str

Returns
help id for the specified tour page.

register_tour_class(*tour_class*)

register app tour class.

Parameters
tour_class¶ (Type[*TourBase*]) – tour class to register.

tour_help_translation(*page_id*)

determine help translation for the passed page id (flow id or app state name).

Parameters
page_id¶ (str) – tour page id.

Return type
Union[str, Dict[str, str], None]

Returns
help translation text/dict (if exists) or None if translation not found.

tour_id_class(*tour_id*)

determine the tour class of the passed tour id.

Parameters
tour_id¶ (str) – tour/flow id to determine tour class for.

Return type
Optional[Any]

Returns
tour class of an existing tour for the passed tour id or None if no tour exists.

translation_short_help_id(*help_id*)

check if a help text exists for the passed help id.

Parameters
help_id¶ (str) – help id to check if a translation/help texts exists.

Return type
Tuple[Union[str, Dict[str, str], None], str]

Returns
tuple of translation text/dict (if exists) and maybe shortened help id(removed detail) or tuple of (None, help_id) if translation not found.

widget_page_id(*wid*)

determine tour page id of passed widget.

Parameters
wid¶ (Optional[Any]) – widget to determine tour page id from (can be None).

Return type
str

Returns
tour page id or empty string if widget has no page id or is None.

class TourBase(*main_app*)

Bases: object

abstract tour base class, automatically registering subclasses as app tours.

subclass this generic, UI-framework-independent base class to bundle pages of a tour and make sure that the attr:*~TourBase.page_ids* and *page_data* attributes are correctly set. a UI-framework-dependent tour overlay/layout instance, created and assigned to main_app.tour_layout, will automatically create an instance of your tour-specific subclass on tour start.

classmethod __init_subclass__(***kwargs*)

register tour class; called on declaration of tour subclass.

__init__(*main_app*)

auto_switch_pages: Union[bool, int]

enable/disable automatic switch of tour pages.

set to *True*, *1* or *-1* to automatically switch tour pages; *True* and *1* will switch to the next page until the last page is reached, while *-1* will switch back to the previous pages until the first page is reached; *-1* and *1* automatically toggles at the first/last page the to other value (endless ping-pong until back/next button gets pressed by the user).

the seconds to display each page before switching to the next one can be specified via the item value of the the dict *page_data* dict with the key *'next_page_delay'*.

page_data: Dict[str, Any]

additional/optional help variables (in *help_vars* key), tour and page text/layout/timing settings.

the class attribute values are default values for all tour pages and get individually overwritten for each tour page by the i18n translations attributes on tour page change via *load_page_data()*.

supported/implemented dict keys:

- *app_flow_delay*: time in seconds to wait until app flow change is completed (def=1.2, >0.9 for auto-width).
- *back_text*: caption of tour previous page button (def=get_text('back')).
- *fade_out_app*: set to 0.0 to prevent the fade out of the app screen (def=1.0).
- *help_vars*: additional help variables, e.g. *help_translation* providing context help translation dict/text.
- *next_text*: caption of tour next page button (def=get_text('next')).
- *next_page_delay*: time in seconds to read the current page before next request_auto_page_switch() (def=9.6).
- *page_update_delay*: time in seconds to wait until tour layout/overlay is completed (def=0.9).
- *tip_text* or '' (empty string): tour page tooltip text fstring message text template. alternatively put as first character a '=' character followed by a tour page flow id to initialize the tip_text to the help translation text of the related flow widget, and the *self* help variable to the related flow widget instance.
- *tour_start_delay*: seconds between tour.start() and on_tour_start main app event (def=TOUR_START_DELAY_DEF).
- *tour_exit_delay*: seconds between tour.stop() and the on_tour_exit main app event (def=TOUR_EXIT_DELAY_DEF).

pages_explained_matchers: Dict[str, Union[Callable[[Any], bool], str, Tuple[Union[Callable[[Any], bool], str], ...]]]

matchers (specified as callable or id-string) to determine the explained widget(s) of each tour page.

each key of this dict is a tour page id (for which the explained widget(s) will be determined).

the value of each dict item is a matcher or a tuple of matchers. each matcher specifies a widget to be explained/targeted/highlighted. for matcher tuples the minimum rectangle enclosing all widgets get highlighted.

the types of matchers, to identify any visible widget, are:

- *find_widget()* matcher callable (scanning framework_win.children)
- evaluation expression resulting in *find_widget()* matcher callable
- widget id string, declared via kv lang, identifying widget in framework_root.ids
- page id string, compiled from widgets app state/flow/focus via *widget_page_id()* to identify widget

page_ids: List[str]

list of tour page ids, either initialized via this class attribute or dynamically.

page_idx: int

index of the current tour page (in *page_ids*)

last_page_idx: Optional[int]

last tour page index (*None* on tour start)

main_app
shortcut to main app instance

layout
tour overlay layout instance

top_popup
top most popup widget (in app simulation)

backup_app_states()
backup current states of this app, including flow.

cancel_auto_page_switch_request(*reset=True*)
cancel auto switch callback if requested, called e.g. from tour layout/overlay next/back buttons.

cancel_delayed_setup_layout_call()
cancel delayed setup layout call request.

property last_page_id: str | None
determine last displayed tour page id.

load_page_data()
load page before switching to it (and maybe reload after preparing app flow and before setup of layout).

next_page()
switch to next tour page.

prev_page()
switch to previous tour page.

request_auto_page_switch()
initiate automatic switch to next tour page.

restore_app_states()
restore app states of this app - saved via *backup_app_states()*.

setup_app_flow()
setup app flow and load page data to prepare a tour page.

setup_explained_widget()
determine and set the explained widget for the actual tour page.

Return type
list

Returns
list of explained widget instances.

setup_layout()
setup/prepare tour overlay/layout after switch of tour page.

setup_texts()
setup texts in tour layout from page_data.

start()
prepare app tour start.

stop()
stop/cancel tour.

teardown_app_flow()

restore app flow and app states before tour finishing or before preparing/switching to prev/next page.

update_page_ids()

update/change page ids on app flow setup (before tour page loading and the tour overlay/layout setup).

override this method to dynamically change the page_ids in a running tour. after adding/removing a page the attribute values of *last_page_idx* and *page_idx* have to be corrected accordingly.

class TourDropdownFromButton(*main_app*)

Bases: *TourBase*

generic tour base class to auto-explain a dropdown menu, starting with the button opening the dropdown.

determine_page_ids = '_v_'

setup_app_flow()

manage the opening state of the dropdown (open dropdown, only close it if opening button get explained).

setup_layout()

prepare layout for all tour pages - first page explains opening dropdown button.

_saved_app_states: Dict[str, Any]

auto_switch_pages: Union[bool, int]

enable/disable automatic switch of tour pages.

set to *True*, *1* or *-1* to automatically switch tour pages; *True* and *1* will switch to the next page until the last page is reached, while *-1* will switch back to the previous pages until the first page is reached; *-1* and *1* automatically toggles at the first/last page the to other value (endless ping-pong until back/next button gets pressed by the user).

the seconds to display each page before switching to the next one can be specified via the item value of the the dict *page_data* dict with the key *'next_page_delay'*.

page_data: Dict[str, Any]

additional/optional help variables (in *help_vars* key), tour and page text/layout/timing settings.

the class attribute values are default values for all tour pages and get individually overwritten for each tour page by the i18n translations attributes on tour page change via *load_page_data()*.

supported/implemented dict keys:

- *app_flow_delay*: time in seconds to wait until app flow change is completed (def=1.2, >0.9 for auto-width).
- *back_text*: caption of tour previous page button (def=get_text('back')).
- *fade_out_app*: set to 0.0 to prevent the fade out of the app screen (def=1.0).
- *help_vars*: additional help variables, e.g. *help_translation* providing context help translation dict/text.
- *next_text*: caption of tour next page button (def=get_text('next')).
- *next_page_delay*: time in seconds to read the current page before next request_auto_page_switch() (def=9.6).
- *page_update_delay*: time in seconds to wait until tour layout/overlay is completed (def=0.9).
- *tip_text* or '' (empty string): tour page tooltip text fstring message text template. alternatively put as first character a '=' character followed by a tour page flow id to initialize the tip_text to the help translation text of the related flow widget, and the *self* help variable to the related flow widget instance.

- *tour_start_delay*: seconds between tour.start() and on_tour_start main app event (def=TOUR_START_DELAY_DEF).
- *tour_exit_delay*: seconds between tour.stop() and the on_tour_exit main app event (def=TOUR_EXIT_DELAY_DEF).

pages_explained_matchers: Dict[str, Union[Callable[[Any], bool], str, Tuple[Union[Callable[[Any], bool], str], ...]]]

matchers (specified as callable or id-string) to determine the explained widget(s) of each tour page.

each key of this dict is a tour page id (for which the explained widget(s) will be determined).

the value of each dict item is a matcher or a tuple of matchers. each matcher specifies a widget to be explained/targeted/highlighted. for matcher tuples the minimum rectangle enclosing all widgets get highlighted.

the types of matchers, to identify any visible widget, are:

- *find_widget()* matcher callable (scanning framework_win.children)
- evaluation expression resulting in *find_widget()* matcher callable
- widget id string, declared via kv lang, identifying widget in framework_root.ids
- page id string, compiled from widgets app state/flow/focus via *widget_page_id()* to identify widget

page_ids: List[str]

list of tour page ids, either initialized via this class attribute or dynamically.

page_idx: int

index of the current tour page (in *page_ids*)

last_page_idx: Optional[int]

last tour page index (*None* on tour start)

class OnboardingTour(*main_app*)

Bases: *TourBase*

onboarding tour for first app start.

__init__(*main_app*)

overridden to handle onboarding tour starts since app installation.

page_idx: int

index of the current tour page (in *page_ids*)

setup_app_flow()

overridden to open user preferences dropdown in responsible_layout tour page.

teardown_app_flow()

overridden to close the opened user preferences dropdown on leaving layout_font_size tour page.

update_page_ids()

overridden to remove 2nd-/well-done-page (only showing once on next-page-jump from 1st-/welcome-page).

_saved_app_states: Dict[str, Any]

auto_switch_pages: Union[bool, int]

enable/disable automatic switch of tour pages.

set to *True*, *1* or *-1* to automatically switch tour pages; *True* and *1* will switch to the next page until the last page is reached, while *-1* will switch back to the previous pages until the first page is reached; *-1* and *1* automatically toggles at the first/last page the to other value (endless ping-pong until back/next button gets pressed by the user).

the seconds to display each page before switching to the next one can be specified via the item value of the the dict *page_data* dict with the key *'next_page_delay'*.

page_data: Dict[str, Any]

additional/optional help variables (in *help_vars* key), tour and page text/layout/timing settings.

the class attribute values are default values for all tour pages and get individually overwritten for each tour page by the i18n translations attributes on tour page change via *load_page_data()*.

supported/implemented dict keys:

- *app_flow_delay*: time in seconds to wait until app flow change is completed (def=1.2, >0.9 for auto-width).
- *back_text*: caption of tour previous page button (def=get_text('back')).
- *fade_out_app*: set to 0.0 to prevent the fade out of the app screen (def=1.0).
- *help_vars*: additional help variables, e.g. *help_translation* providing context help translation dict/text.
- *next_text*: caption of tour next page button (def=get_text('next')).
- *next_page_delay*: time in seconds to read the current page before next request_auto_page_switch() (def=9.6).
- *page_update_delay*: time in seconds to wait until tour layout/overlay is completed (def=0.9).
- *tip_text* or '' (empty string): tour page tooltip text fstring message text template. alternatively put as first character a *'='* character followed by a tour page flow id to initialize the tip_text to the help translation text of the related flow widget, and the *self* help variable to the related flow widget instance.
- *tour_start_delay*: seconds between tour.start() and on_tour_start main app event (def=TOUR_START_DELAY_DEF).
- *tour_exit_delay*: seconds between tour.stop() and the on_tour_exit main app event (def=TOUR_EXIT_DELAY_DEF).

pages_explained_matchers: Dict[str, Union[Callable[[Any], bool], str, Tuple[Union[Callable[[Any], bool], str], ...]]]

matchers (specified as callable or id-string) to determine the explained widget(s) of each tour page.

each key of this dict is a tour page id (for which the explained widget(s) will be determined).

the value of each dict item is a matcher or a tuple of matchers. each matcher specifies a widget to be explained/targeted/highlighted. for matcher tuples the minimum rectangle enclosing all widgets get highlighted.

the types of matchers, to identify any visible widget, are:

- *find_widget()* matcher callable (scanning framework_win.children)
- evaluation expression resulting in *find_widget()* matcher callable
- widget id string, declared via kv lang, identifying widget in framework_root.ids
- page id string, compiled from widgets app state/flow/focus via *widget_page_id()* to identify widget

page_ids: List[str]

list of tour page ids, either initialized via this class attribute or dynamically.

last_page_idx: Optional[int]

last tour page index (*None* on tour start)

class UserPreferencesTour(*main_app*)

Bases: *TourDropdownFromButton*

user preferences menu tour.

__init__(*main_app*)

auto_switch_pages: Union[bool, int]

enable/disable automatic switch of tour pages.

set to *True*, *1* or *-1* to automatically switch tour pages; *True* and *1* will switch to the next page until the last page is reached, while *-1* will switch back to the previous pages until the first page is reached; *-1* and *1* automatically toggles at the first/last page the to other value (endless ping-pong until back/next button gets pressed by the user).

the seconds to display each page before switching to the next one can be specified via the item value of the the dict *page_data* dict with the key *'next_page_delay'*.

_saved_app_states: Dict[str, Any]

page_data: Dict[str, Any]

additional/optional help variables (in *help_vars* key), tour and page text/layout/timing settings.

the class attribute values are default values for all tour pages and get individually overwritten for each tour page by the i18n translations attributes on tour page change via *load_page_data()*.

supported/implemented dict keys:

- *app_flow_delay*: time in seconds to wait until app flow change is completed (def=1.2, >0.9 for auto-width).
- *back_text*: caption of tour previous page button (def=get_text('back')).
- *fade_out_app*: set to 0.0 to prevent the fade out of the app screen (def=1.0).
- *help_vars*: additional help variables, e.g. *help_translation* providing context help translation dict/text.
- *next_text*: caption of tour next page button (def=get_text('next')).
- *next_page_delay*: time in seconds to read the current page before next request_auto_page_switch() (def=9.6).
- *page_update_delay*: time in seconds to wait until tour layout/overlay is completed (def=0.9).
- *tip_text* or '' (empty string): tour page tooltip text fstring message text template. alternatively put as first character a '=' character followed by a tour page flow id to initialize the tip_text to the help translation text of the related flow widget, and the *self* help variable to the related flow widget instance.
- *tour_start_delay*: seconds between tour.start() and on_tour_start main app event (def=TOUR_START_DELAY_DEF).
- *tour_exit_delay*: seconds between tour.stop() and the on_tour_exit main app event (def=TOUR_EXIT_DELAY_DEF).

pages_explained_matchers: Dict[str, Union[Callable[[Any], bool], str, Tuple[Union[Callable[[Any], bool], str], ...]]]

matchers (specified as callable or id-string) to determine the explained widget(s) of each tour page.

each key of this dict is a tour page id (for which the explained widget(s) will be determined).

the value of each dict item is a matcher or a tuple of matchers. each matcher specifies a widget to be explained/targeted/highlighted. for matcher tuples the minimum rectangle enclosing all widgets get highlighted.

the types of matchers, to identify any visible widget, are:

- *find_widget()* matcher callable (scanning framework_win.children)
- evaluation expression resulting in *find_widget()* matcher callable
- widget id string, declared via kv lang, identifying widget in framework_root.ids
- page id string, compiled from widgets app state/flow/focus via *widget_page_id()* to identify widget

page_ids: List[str]

list of tour page ids, either initialized via this class attribute or dynamically.

page_idx: int

index of the current tour page (in *page_ids*)

last_page_idx: Optional[int]

last tour page index (*None* on tour start)

class HelpAppBase(***console_app_kwargs*)

Bases: *MainAppBase*, ABC

main app help base class.

displayed_help_id: str = ''

message id of currently explained/focused target widget in help mode

help_activator: Any = None

help mode de-/activator button widget

help_layout: Optional[Any] = None

help text container widget in active help mode else None

tour_layout: Optional[Any] = None

tour layout/overlay widget in active tour mode else None

tour_overlay_class: Optional[Type] = None

UI-framework-specific tour overlay class, set by main app subclass

_next_help_id: str = ''

last app-state/flow change to show help text on help mode activation

_closing_popup_open_flow_id: str = ''

flow id of just closed popup

abstract call_method_delayed(*delay*, *callback*, **args*, ***kwargs*)

delayed call of passed callable/method with args/kwargs catching and logging exceptions preventing app exit.

Parameters

- **delay** (float) – delay in seconds before calling the callable/method specified by *callback*.
- **callback** (Union[Callable, str]) – either callable or name of the main app method to call.
- **args** – args passed to the callable/main-app-method to be called.
- **kwargs** – kwargs passed to the callable/main-app-method to be called.

Return type
Any

Returns
delayed call event object instance, providing a *cancel* method to allow the cancellation of the delayed call within the delay time.

abstract ensure_top_most_z_index(*widget*)

ensure visibility of the passed widget to be the top most in the z index/order.

Parameters
widget (Any) – the popup/dropdown/container widget to be moved to the top.

abstract help_activation_toggle()

button tapped event handler to switch help mode between active and inactive (also inactivating tour).

change_app_state(*app_state_name*, *state_value*, *send_event=True*, *old_name=''*)

change app state via *change_app_state()*, show help text in active help mode.

all parameters are documented in the overwritten method *change_app_state()*.

change_flow(*new_flow_id*, ***event_kwargs*)

change/switch flow id - overriding *change_flow()*.

more detailed documentation of the parameters you find in the overwritten method *change_app_state()*.

this method returns True if flow changed and got confirmed by a declared custom event handler (either event method or Popup class) of the app, if the help mode is *not* active or the calling widget is selected in active help mode, else False.

Return type
bool

help_app_state_display(*help_vars*, *changed=False*)

actualize the help layout if active, before and after the change of the app state.

Parameters

- **help_vars** (Dict[str, Any]) – locals (args/kwargs) of overwritten *change_flow()* method.

  items passed to the help text formatter:

  - *count*: optional number used to render a pluralized help text for this app state change.

- **changed** (bool) – False before change of the app state, pass True if app state got just/already changed.

Return type
bool

Returns
True if help mode and layout is active and found target widget is locked, else False.

help_display(*help_id*, *help_vars*, *key_suffix=''*, *must_have=False*)

display help text to the user in activated help mode.

Parameters

- **help_id** (str) – help id to show help text for.
- **help_vars** (Dict[str, Any]) – variables used in the conversion of the f-string expression to a string. optional items passed to the help text formatter: * *count*: optional number used to render a pluralized help text. * *self*: target widget to show help text for.
- **key_suffix** (str) – suffix to the key used if the translation is a dict.
- **must_have** (bool) – pass True to display error help text and console output if no help text exists.

Return type
bool

Returns
True if help text got found and displayed.

help_flow_display(*help_vars*, *changed=False*)

actualize the help layout if active, exclusively called by *change_flow()*.

Parameters

- **help_vars** (Dict[str, Any]) – locals (args/kwargs) of overwritten *change_flow()* method.
- **changed** (bool) – False before change to new flow, pass True if flow got changed already.

Return type
bool

Returns
True if help layout is active and found target widget is locked, else False.

help_is_inactive(*help_id*)

check if help mode is inactive and reserve/note-down current help id for next help mode activation.

Parameters
help_id (str) – help id to be reserved for next help activation with empty help id.

Return type
bool

Returns
True if help mode is inactive, else False.

help_target_and_id(*help_vars*)

find help widget/target and help id on help mode activation.

Parameters
help_vars (Dict[str, Any]) – optional help vars.

Return type
Tuple[Any, str]

Returns
tuple of help target widget and help id.

help_widget(*help_id*, *help_vars*)

ensure/find help target widget via attribute name/value and extend *help_vars*.

Parameters

- **help_id** (str) – widget.help_id attribute value to detect widget and call stack locals.
- **help_vars** (Dict[str, Any]) – help env variables, to be extended with event activation stack frame locals and a 'self' key with the help target widget.

Return type
Any

Returns
found help target widget or self.help_activator if not found.

key_press_from_framework(*modifiers*, *key*)

overwritten ae.gui_app.MainAppBase method to suppress key press events in help or app tour mode.

Parameters

- **modifiers** (str) – modifier keys.
- **key** (str) – key character.

Return type
bool

Returns
True if key got consumed/used else False.

on_app_started()

app initialization event - the last one on app startup.

on_flow_popup_close(*_flow_key*, *_event_kwargs*)

overwritten popup close handler of FlowPopup widget to reset help widget/text.

Parameters

- **_flow_key** (str) – (unused)
- **_event_kwargs** (Dict[str, Any]) – (unused)

Return type
bool

Returns
always True.

save_app_states()

override MainAppBase method to not overwrite app states if app tour is active.

Return type
str

start_app_tour(*tour_class=None*)

start new app tour, automatically cancelling a currently running app tour.

Parameters
tour_class (Optional[Type[*TourBase*]]) – optional tour (pages) class, default: tour of current help id or *OnboardingTour*.

Return type
bool

Returns
True if UI-framework support tours/has tour_overlay_class set and tour got started.

widget_by_page_id(*page_id*)

determine the first (top-most) widget having the passed tour page id.

Parameters
page_id¶ (str) – widgets tour page id from *tap_flow_id/focus_flow_id/app_state_name* attribute.

Return type
Optional[Any]

Returns
widget that has a *tap_flow_id/focus_flow_id/app_state_name* attribute with the value of the passed page id or None if not found.

widget_tourable_children_page_ids(*parent_widget*)

determine all visible and tourable children widgets of the passed parent and its child container widgets.

Parameters
parent_widget¶ (Any) – parent widget to determine all children that are tourable.

Return type
List

Returns
list of page ids of tourable children of the passed parent widget.

4.29 ae.kivy_glsl

4.29.1 add glsl shaders to your kivy widget

this ae namespace portion provides the mixin class *ShadersMixin* that can be combined with any Kivy widget to display GLSL-/shader-based graphics, gradients and animations.

additionally some *built-in shaders* are integrated into this portion. more shader examples can be found in the glsl sub-folder of the GlslTester demo application.

usage of ShadersMixin class

to add the *ShadersMixin* mixin class to a Kivy widget in your python code file you have to specify it in the declaration of your widget class. the following example is extending Kivy's BoxLayout layout with a shader:

```
from kivy.uix.boxlayout import BoxLayout
from ae.kivy_glsl import ShadersMixin

class MyBoxLayoutWithShader(ShadersMixin, BoxLayout):
```

alternatively you can declare a shader for your widget as a new kv rule within a kv file:

```
<MyBoxLayoutWithShader@ShadersMixin+BoxLayout>
```

to register a shader, call the *ShadersMixin.add_shader()* method:

```
shader_id = widget_instance.add_shader()
```

by default *add_shader()* is using the built-in `plasma hearts shader`, provided by this portion. the next example is instead using the built-in `plunge waves shader`:

```
from ae.kivy_glsl import BUILT_IN_SHADERS

widget_instance.add_shader(shader_code=BUILT_IN_SHADERS['plunge_waves'])
```

alternatively you can use your own shader code by specifying it on call of the method *add_shader()* either as code block string to the *paramref: `~ShadersMixin.add_shader.shader_code* argument or as file name to the *paramref: `~ShadersMixin.add_shader.shader_file* argument.

animation shaders like the built-in plunge waves and plasma hearts shaders need to be refreshed by a timer. the refreshing frequency can be specified via the *update_freq* parameter. to disable the automatic creation of a timer event pass a zero value to this argument.

Hint: the demo apps ComPartY and GlslTester are disabling the automatic timer event for each shader and using instead a Kivy clock timer to update the frames of all active shaders.

---

store the return value of *add_shader()* to stop, pause or to delete the shader later. the following examples demonstrates the deletion of a shader by calling the *del_shader()* method:

```
widget_instance.del_shader(shader_id)
```

Note: you can activate multiple shaders for the same widget. the visibility and intensity of each shader depends then on the implementation of the shader codes and the values of the input arguments (especially *alpha* and *tex_col_mix*) for each shader (see parameter *glsl_dyn_args*).

---

shader compilation errors and renderer crashes

on some devices (mostly on Android) the shader script does not compile. the success property of Kivy's shader class is then set to False and an error message like the following gets printed on to the console output:

```
[ERROR   ] [Shader      ] <fragment> failed to compile (gl:0)
[INFO    ] [Shader      ] fragment shader: <b"0:27(6): error: ....
```

some common failure reasons are:

- missing declaration of used uniform input variables.
- non-input/output variables declared on module level (they should be moved into main or any other function).

if a compile error occurred then the *run_state* value of the shader id dict will be set to *'error'* - check the value of the *error_message* key of the shader id dict for more details on the error.

in other cases the shader code compiles fine but then the renderer is crashing in the vbo.so library and w/o printing any Python traceback to the console - see also this Kivy issues).

if the crash only happens on some Adreno GPUs then see issues p4a #2723 and kivy #8080 and a possible fix in kivy #8098. sometimes these type of crashes can be prevented if the texture of the widget (or of the last shader) gets fetched (w/ the function texture2D(texture0, tex_coord0)) - even if it is not used for the final gl_FragColor output variable. in

some cases additional to fetch the texture, the return value of the *texture2D* call has to be accessed at least once at the first render cycle.

built-in shaders

the `circled alpha shader` is a simple gradient pixel shader without any time-based animations.

the `plunge waves shader` is animated and inspired by the kivy pulse shader example (Danguafer/Silexars, 2010) https://github.com/kivy/kivy/blob/master/examples/shader/shadertree.py.

the animated `plasma hearts shader` is inspired by the kivy plasma shader example https://github.com/kivy/kivy/blob/master/examples/shader/plasma.py.

Hint: the GlslTester and ComPartY applications are demonstrating the usage of this portion.

the literals of the built-in shaders got converted into constants, following the recommendations given in the accepted answer of this SO question.

Module Attributes

DEFAULT_FPS	default frames-per-second
ShaderIdType	shader internal data and id
BUILT_IN_SHADERS	dict of built-in shader code blocks - key specifies shader in *shader_code*.
RENDER_SHAPES	supported render shapes, specified in *render_shape*.
SHADER_PARAMETER_MATCHER	match uniform var declarations
HIDDEN_SHADER_PARAMETERS	uniform vars that are not editable by the user

Functions

shader_parameter_alias(shader_code, arg_name)	check for alias for the passed shader argument name in the specified shader code.
shader_parameters(shader_code)	shader arg names of the shader code in the specified shader code.

Classes

ShadersMixin()	shader mixin base class

DEFAULT_FPS = 30.0

default frames-per-second

ShaderIdType

shader internal data and id

alias of `Dict[str, Any]`

```
BUILT_IN_SHADERS = {'circled_alpha':  'uniform float alpha;\nuniform float
tex_col_mix;\nuniform vec2 center_pos;\nuniform vec2 win_pos;\nuniform vec2
resolution;\nuniform vec4 tint_ink;\n\nvoid main(void)\n{\n vec2 pix_pos =
(frag_modelview_mat * gl_FragCoord).xy;\n float len = length(pix_pos - center_pos);\n
pix_pos -= win_pos;\n float dis = len / max(pix_pos.x, max(pix_pos.y, max(resolution.x -
pix_pos.x, resolution.y - pix_pos.y)));\n vec3 col = tint_ink.rgb;\n if (tex_col_mix !=
0.0) {\n vec4 tex = texture2D(texture0, tex_coord0);\n col = mix(tex.rgb, col,
tex_col_mix);\n }\n gl_FragColor = vec4(col, dis * alpha);\n}\n', 'colored_smoke':
'uniform float alpha;\nuniform float tex_col_mix;\nuniform float time;\nuniform vec2
center_pos;\nuniform vec2 mouse; // density, speed\nuniform vec2 resolution;\nuniform
vec4 tint_ink;\n\nconst float ONE = 0.99999999999999;\n\nfloat rand(vec2 n) {\n //This is
just a compounded expression to simulate a random number based on a seed given as n\n
return fract(cos(dot(n, vec2(12.98982, 4.14141))) * 43758.54531);\n}\n\nfloat noise(vec2
n) {\n //Uses the rand function to generate noise\n const vec2 d = vec2(0.0, ONE);\n vec2
b = floor(n), f = smoothstep(vec2(0.0), vec2(ONE), fract(n));\n return mix(mix(rand(b),
rand(b + d.yx), f.x), mix(rand(b + d.xy), rand(b + d.yy), f.x), f.y);\n}\n\nfloat
fbm(vec2 n) {\n //fbm stands for "Fractal Brownian Motion"
https://en.wikipedia.org/wiki/Fractional_Brownian_motion\n float total = 0.0;\n float
amplitude = 1.62;\n for (int i = 0; i < 3; i++) {\n total += noise(n) * amplitude;\n n +=
n;\n amplitude *= 0.51;\n }\n return total;\n}\n\nvoid main() {\n //This is where our
shader comes together\n const vec3 c1 = vec3(126.0/255.0, 0.0/255.0, 96.9/255.0);\n
//const vec3 c2 = vec3(173.0/255.0, 0.0/255.0, 161.4/255.0);\n vec3 c2 = tint_ink.rgb;\n
const vec3 c3 = vec3(0.21, 0.0, 0.0);\n const vec3 c4 = vec3(165.0/255.0, 129.0/255.0,
214.4/255.0);\n const vec3 c5 = vec3(0.12);\n const vec3 c6 = vec3(0.9);\n vec2 pix_pos =
(gl_FragCoord.xy - center_pos) / resolution.xy - vec2(0.0, 0.51);\n //this is how
"packed" the smoke is in our area. try changing 15.0 to 2.1, or something else\n vec2 p =
pix_pos * (ONE + mouse.x / resolution.x * 15.0);\n //the fbm function takes p as its seed
(so each pixel looks different) and time (so it shifts over time)\n float q = fbm(p -
time * 0.12);\n float speed = 3.9 * time * mouse.y / resolution.y;\n vec2 r = vec2(fbm(p
+ q + speed - p.x - p.y), fbm(p + q - speed));\n vec3 col = (mix(c1, c2, fbm(p + r)) +
mix(c3, c4, r.y) - mix(c5, c6, r.x)) * cos(pix_pos.y);\n col *= ONE - pix_pos.y;\n if
(tex_col_mix != 0.0) {\n vec4 tex = texture2D(texture0, tex_coord0);\n col = mix(tex.rgb,
col, tex_col_mix);\n }\n gl_FragColor = vec4(col, (alpha + tint_ink.a) / 2.01);\n}\n',
'fire_storm':  'uniform float alpha;\nuniform float contrast; // speed\nuniform float
tex_col_mix;\nuniform float time;\nuniform vec2 center_pos;\nuniform vec2 mouse; //
intensity, granularity\nuniform vec2 resolution;\nuniform vec4 tint_ink;\n\n#define TAU
6.283185307182\n#define MAX_ITER 15\n\nvoid main( void ) {\n float t = time*contrast +
23.01;\n // uv should be the 0-1 uv of texture...\n vec2 xy = (gl_FragCoord.xy -
center_pos) / resolution.yy; // - vec2(0.9);\n vec2 uv = vec2(atan(xy.y, xy.x) * 6.99999
/ TAU, log(length(xy)) * (0.21 + mouse.y / resolution.y) - time * 0.21);\n vec2 p =
mod(uv*TAU, TAU)-250.02;\n vec2 i = vec2(p);\n float c = 8.52;\n float intensity = 0.003
+ mouse.x / resolution.x / 333.3; // = .005;\n\n for (int n = 0; n < MAX_ITER; n++) {\n
float t = t * (1.02 - (3.498 / float(n+1)));\n i = p + vec2(cos(t - i.x) + sin(t + i.y),
sin(t - i.y) + cos(t + i.x));\n c += 1.0/length(vec2(p.x / (sin(i.x+t)/intensity),p.y /
(cos(i.y+t)/intensity)));\n }\n c /= float(MAX_ITER);\n c = 1.272 - pow(c, 6.42);\n vec3
colour = vec3(pow(abs(c), 8.01));\n colour = clamp(colour + tint_ink.rgb, 0.0,
0.999999);\n if (tex_col_mix != 0.0) {\n vec4 tex = texture2D(texture0, tex_coord0);\n
colour = mix(tex.rgb, colour, tex_col_mix);\n }\n gl_FragColor = vec4(colour, (alpha +
tint_ink.a) / 2.00001);\n }\n', 'plasma_hearts':  'uniform float alpha;\nuniform float
contrast;\nuniform float tex_col_mix;\nuniform float time;\nuniform vec2
center_pos;\nuniform vec2 win_pos;\nuniform vec2 resolution;\nuniform vec4
tint_ink;\n\nconst float THOUSAND = 963.9;\nconst float HUNDRED = 69.3;\nconst float TEN
= 9.9;\nconst float TWO = 1.83;\nconst float ONE = 0.99;\n\nvoid main(void)\n{\n vec2
pix_pos = (frag_modelview_mat * gl_FragCoord).xy - win_pos;\n vec2 rel_center =
center_pos - win_pos;\n float x = abs(pix_pos.x - rel_center.x);\n float y =
abs(pix_pos.y - rel_center.y - resolution.y);\n\n float m1 = x + y + cos(sin(time) * TWO)
* HUNDRED + sin(x / HUNDRED) * THOUSAND;\n float m2 = y / resolution.y;\n float m3 = x /
resolution.x + time * TWO;\n\n float c1 = abs(sin(m2 + time) / TWO + cos(m3 / TWO - m2 -
m3 + time));\n float c2 = abs(sin(c1 + sin(m1 / THOUSAND + time) + sin(y / HUNDRED +
time) + sin((x + y) / HUNDRED) * TWO));\n float c3 = abs(sin(c2 + cos(m2 + m3 + c2) +
```

dict of built-in shader code blocks - key specifies shader in *shader_code*.

RENDER_SHAPES = (<class 'kivy.graphics.vertex_instructions.Ellipse'>, <class 'kivy.graphics.vertex_instructions.Rectangle'>, <class 'kivy.graphics.vertex_instructions.RoundedRectangle'>)

supported render shapes, specified in *render_shape*.

SHADER_PARAMETER_MATCHER = re.compile('^uniform (float|vec2|vec4) ([a-z_]+);', re.MULTILINE)

match uniform var declarations

HIDDEN_SHADER_PARAMETERS = ('resolution', 'time', 'win_pos')

uniform vars that are not editable by the user

shader_parameter_alias(*shader_code*, *arg_name*)

check for alias for the passed shader argument name in the specified shader code.

Parameters

- **shader_code** (str) – shader code to determine the parameter alias names (from comment of uniform declaration).
- **arg_name** (str) – shader arg name.

Return type
str

Returns
alias (if found) or shader arg name (if not).

shader_parameters(*shader_code*)

shader arg names of the shader code in the specified shader code.

Parameters
shader_code (str) – shader code from which to determine the declared uniform parameter names.

Return type
Tuple[str, ...]

Returns
tuple of shader arg names of the shader code passed into the argument *shader_code*, plus the *start_time* argument which controls, independent of to be included in the shader code, how the *time* argument get prepared in each render frame (see also *start_time*).

class ShadersMixin

Bases: object

shader mixin base class

canvas: Any

center: Tuple[float, float]

fbind: Callable

parent: Any

pos: list

size: list

to_window: Callable

unbind_uid: Callable

added_shaders

list of shader-ids/kwarg-dicts for each shader

running_shaders: List[Dict[str, Any]] = []

list/pool of active/running shaders/render-contexts

_pos_fbind_uid: int = 0

_size_fbind_uid: int = 0

add_shader(*add_to=''*, *shader_code='uniform float alpha;\\nuniform float contrast;\\nuniform float tex_col_mix;\\nuniform float time;\\nuniform vec2 center_pos;\\nuniform vec2 win_pos;\\nuniform vec2 resolution;\\nuniform vec4 tint_ink;\\n\\nconst float THOUSAND = 963.9;\\nconst float HUNDRED = 69.3;\\nconst float TEN = 9.9;\\nconst float TWO = 1.83;\\nconst float ONE = 0.99;\\n\\nvoid main(void)\\n{\\n vec2 pix_pos = (frag_modelview_mat * gl_FragCoord).xy - win_pos;\\n vec2 rel_center = center_pos - win_pos;\\n float x = abs(pix_pos.x - rel_center.x);\\n float y = abs(pix_pos.y - rel_center.y - resolution.y);\\n\\n float m1 = x + y + cos(sin(time) * TWO) * HUNDRED + sin(x / HUNDRED) * THOUSAND;\\n float m2 = y / resolution.y;\\n float m3 = x / resolution.x + time * TWO;\\n\\n float c1 = abs(sin(m2 + time) / TWO + cos(m3 / TWO - m2 - m3 + time));\\n float c2 = abs(sin(c1 + sin(m1 / THOUSAND + time) + sin(y / HUNDRED + time) + sin((x + y) / HUNDRED) * TWO));\\n float c3 = abs(sin(c2 + cos(m2 + m3 + c2) + cos(m3) + sin(x / THOUSAND)));\\n\\n vec4 tex = texture2D(texture0, tex_coord0);\\n float dis = TWO * distance(pix_pos, rel_center) / max(resolution.x, resolution.y);\\n vec4 col = vec4(c1, c2, c3, contrast * (ONE - dis)) * tint_ink * TWO;\\n col = mix(tex, col, tex_col_mix);\\n gl_FragColor = vec4(col.rgb, col.a * sqrt(alpha));\\n}\\n'*, *shader_file=''*, *start_time=0.0*, *update_freq=30.0*, *run_state='running'*, *render_shape=<class 'kivy.graphics.vertex_instructions.Rectangle'>*, ***glsl_dyn_args*)

add/register a new shader for the mixing-in widget.

Parameters

- **add_to** (str) – '' to add to current canvas, 'before' and 'after' to the before/after canvas of the widget instance mixed-into. if the canvas does not exist then the shaders render context will be set as a current canvas.
- **shader_code** (str) – fragment shader code block or key of BUILT_IN_SHADERS dict if first character is '='. this argument will be ignored if *shader_file* is not empty.
- **shader_file** (str) – filename with the glsl shader code (with "–VERTEX" or "–FRAGMENT" sections) to load.
- **start_time** (Optional[float]) – base/start time. passing the default value zero is syncing the *time* glsl parameter of this shader with `kivy.clock.Clock.get_boottime()`. pass `None` to initialize this argument to the current Clock boot time, to start the *time* glsl argument at zero.
- **update_freq** (float) – shader render update frequency. pass 0.0 to disable creation of an update timer.
- **run_state** (str) – optional shader run state (default='running'), pass 'paused'/'error' to not run it.
- **render_shape** (Union[Any, str]) – pass one of the supported shapes (*RENDER_SHAPES*) either as shape class or str.

- **glsl_dyn_args** – extra/user dynamic shader parameters, depending on the used shader code. the keys of this dict are the names of the corresponding glsl input variables in your shader code. the built-in shaders (provided by this module) providing the following glsl input variables:
  - *'alpha'*: opacity (float, 0.0 - 1.0).
  - *'center_pos'*: center position in Window coordinates (tuple(float, float)).
  - *'contrast'*: color contrast (float, 0.0 - 1.0).
  - *'mouse'*: mouse pointer position in Window coordinates (tuple(float, float)).
  - *'resolution'*: width and height in Window coordinates (tuple(float, float)).
  - ***'tex_col_mix'*: factor (float, 0.0 - 1.0) to mix the kivy input texture** and the calculated color. a value of 1.0 will only show the shader color, whereas 0.0 will result in the color of the input texture (uniform texture0).
  - *'tint_ink'*: tint color with color parts in the range 0.0 till 1.0.
  - ***'time'*: animation time (offset to *start_time*) in seconds. if** specified as constant (non-dynamic) value then you have to call the *next_tick()* method to increment the timer for this shader.

  pass a callable to provide a dynamic/current value, which will be called on each rendering frame without arguments and the return value will be passed into the glsl shader.

  Note: don't pass *int* values because some renderer will interpret them as *0.0*.

Return type
Dict[str, Any]

Returns
index (id) of the created/added render context.

_compile_shader(*shader_id*)

try to compile glsl shader file/code - raise ValueError if compilation failed.

Parameters
shader_id (Dict[str, Any]) – shader id (internal data dict with either *shader_file* (preference) or *shader_code* key representing the shader code to use).

Return type
RenderContext

Returns
kivy/glsl render context with the compiled shader attached to it.

del_shader(*shader_id*)

remove shader_id added via add_shader.

Parameters
shader_id (Dict[str, Any]) – id of the shader to remove (returned by *add_shader()*). ignoring if the passed shader got already removed.

next_tick(*increment=0.03333333333333333*)

increment glsl *time* input argument if running_shaders get updated manually/explicitly by the app.

Parameters
increment¶ (float) – delta in seconds for the next refresh of all running_shaders with a *time* constant.

on_added_shaders(**_args*)

added_shaders list property changed event handler.

on_parent(**_args*)

parent changed event handler.

play_shader(*shader_id*)

create new render context canvas and add it to the widget canvas to display shader output.

Parameters
shader_id¶ (Dict[str, Any]) – shader id and internal data dict with either *shader_file* (preference) or *shader_code* key representing the shader code to use. a shader dict with the *shader_code* key is a fragment shader (w/o a vertex shader), that will be automatically prefixed with the Kivy fragment shader header file template, if the $HEADER$ placeholder is not included in the shader code (even if it is commented out, like in the following glsl code line: `#ifdef GL_ES //$HEADER$`).

Return type
str

Returns
"" if shader is running/playing or error message string.

_pos_changed(**_args*)

pos changed event handler.

_refresh_glsl(*shader_id*, *_dt*)

timer/clock event handler to animate and sync one canvas shader.

refresh_running_shaders()

manually update all running_shaders.

refresh_shader(*shader_id*)

update the shader arguments for the current animation frame.

Parameters
shader_id¶ (Dict[str, Any]) – dict with render context, rectangle and glsl input arguments.

Return type
str

Returns
empty string if arguments got passed to the shader without errors, else error message.

_size_changed(**_args*)

size changed event handler.

stop_shader(*shader_id*, *set_run_state=True*)

stop shader by removing it from started shaders.

Parameters

- **shader_id**¶ (Dict[str, Any]) – id of the shader to stop. ignoring if the passed shader got already stopped.

- **set_run_state**¶ (bool) – pass False to prevent that the *run_state* of the *shader_id* gets changed to *paused* (for internal use to refresh all running shaders).

update_shaders()

stop/unbind all shaders, then restart/bind the shaders having their *run_state* as *'running'*.

4.30 ae.kivy_dyn_chi

4.30.1 dynamic children mix-in for kivy container widgets

this ae portion is providing the mixin class *DynamicChildrenBehavior* to add children widgets dynamically and data-driven to your kivy popup widget (like DropDowns, Popups, Menus, Selectors).

Module Attributes

AttrMapType	child attribute (value of the 'kwargs' and 'attributes' keys)
DataItemValueType	item dict value (str for 'cls' key, dict for 'kwargs'/'attributes')
DataItemType	*DynamicChildrenBehavior.child_data_maps* item type
ChildrenDataType	*DynamicChildrenBehavior.child_data_maps* type

Classes

DynamicChildrenBehavior(**kwargs)	mixin class for the dynamic creation/refresh of child widgets from a data map.

AttrMapType

child attribute (value of the 'kwargs' and 'attributes' keys)

alias of Dict[str, Any]

DataItemValueType

item dict value (str for 'cls' key, dict for 'kwargs'/'attributes')

alias of Union[str, Dict[str, Any]]

DataItemType

DynamicChildrenBehavior.child_data_maps item type

alias of Dict[str, Union[str, Dict[str, Any]]]

ChildrenDataType

DynamicChildrenBehavior.child_data_maps type

alias of List[Dict[str, Union[str, Dict[str, Any]]]]

_child_data_dict(*child_data*, *cls*, *key*, *defaults*)

determine child data dict values and put children default values for the keys not specified in child data.

Return type
Dict[str, Any]

class DynamicChildrenBehavior(**kwargs*)

Bases: object

mixin class for the dynamic creation/refresh of child widgets from a data map.

at least one of the classes that is mixing in this class has to inherit from Widget (or EventDispatcher) to get the *child_data_maps* attribute correctly initialized and firing on change.

child_data_defaults: List[Dict[str, Union[str, Dict[str, Any]]]]

child data default values for all the children with the same *cls* key that are specified via the *ae.kivy_dyn_chi.DynamicChildrenBehavior.child_data_maps* property. if the *cls* key is missing or its item value is empty then the defaults in the other item values will be used for all children.

child_data_defaults is a ListProperty and defaults to an empty list.

child_data_maps: List[Dict[str, Union[str, Dict[str, Any]]]]

list of child data dicts to instantiate the children of the inheriting layout/widget.

each child data dict is defining its own widget with the following keys:

- *cls*: either the class name or the class/type object of the widget to be created dynamically.
- *kwargs*: dict of keyword arguments that will be passed to the constructor method of the widget. all values in this dict with the magic string *'replace_with_data_map_popup'* will be replaced with the instance of the container before it gets passed to the *__init__* method of the child (see *deep_replace()*).
- *attributes*: dict of attributes where the key is specifying the attribute name/path and the value the finally assigned attribute value. the attribute name (the key of this dict) can be a deep/combined attribute/index path which allows to update deeper objects within the child object (via *key_path_object()*). all values in this dict with the magic string *'replace_with_data_map_popup'* will be replaced with the instance of the container before the child attributes get updated. all values in this dict with the magic string *'replace_with_data_map_child'* will be replaced with the instance of the child before they get applied to it (via *deep_replace()*).

child_data_maps is a ListProperty and defaults to an empty list.

bind: Callable

container: Widget

_container: Widget

__init__(**kwargs*)

add dynamic creation and refresh of children to this layout (Popup/Dropdown/...) widget.

refresh_child_data_widgets(*_args*, ***init_kwargs*)

recreate dynamic children of the passed widget.

Parameters

- **_args** – not needed extra args (only passed if this function get called as event handler).
- **init_kwargs** – container kwargs (passed from *__init__()* method).

4.31 ae.kivy_relief_canvas

4.31.1 inner/outer elliptic/square reliefs for any kivy widget

the *ReliefCanvas* mixin class of this ae namespace portion can be added to any square or elliptic Kivy widget to draw an inner and/or outer relief, in order to convert your widget to have an outstanding or sunken 3D-appearance.

Module Attributes

ReliefColors	tuple of (top, bottom) relief colors or empty tuple
ReliefBrightness	top and bottom brightness/darken factor

Functions

relief_colors([color_or_ink, darken_factors])	calculate the (top and bottom) colors used for the relief lines/drawings.

Classes

ReliefCanvas(**kwargs)	relief canvas mixin class.

ReliefColors

tuple of (top, bottom) relief colors or empty tuple

alias of Union[Tuple[Union[Tuple[float, float, float], List[float]], Union[Tuple[float, float, float], List[float]]], Tuple]

ReliefBrightness

top and bottom brightness/darken factor

alias of Tuple[float, float]

relief_colors(*color_or_ink=(0, 0, 0)*, *darken_factors=(0.6, 0.3)*)

calculate the (top and bottom) colors used for the relief lines/drawings.

Parameters

- **color_or_ink** (Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]]) – color used to calculate the relief colors from, which will first be lightened until one of the color parts (R, G or B) reach the value 1.0; then darken factors will be applied to the color parts. If not passed then grey colors will be returned.

  Note: if the alpha value of paramref:~*relief_colors.color_or_ink* is zero then no relief colors will be calculated and an empty tuple will be returned (disabling relief).

- **darken_factors** (Tuple[float, float]) – two factors to darken (1) the top and (2) the bottom relief color parts.

Return type
Union[Tuple[Union[Tuple[float, float, float], List[float]], Union[Tuple[float, float, float], List[float]]], Tuple]

Returns
tuple with darkened colors calculated from `color_or_ink` and `darken_factors` or an empty tuple if the alpha value of `color_or_ink` has a zero value.

class ReliefCanvas(***kwargs*)

Bases: object

relief canvas mixin class.

to activate the drawing of a relief you have to specify two colors, one for the top part and another one for the bottom part of the relief, which are both stored in a single kivy property. the function *relief_colors()* can be used to calculate lightened and darkened values of the surface color of the widget:

```
MySquareRaisedWidgetWithColoredSurface:
    surface_color: 0.9, 0.6, 0.3, 1.0
    relief_square_outer_colors: relief_colors(color_or_ink=self.surface_color)
```

this will result in a raised widget with a square outer relief where the top/left relief color get a lightened value and the bottom/right relief a darkened value of the color specified by *surface_color*.

using the default values will result in raised widgets with the inner part sunken, simulating the light source in the top left window border/corner. to make a sunken widget for the same light source position you simply have to flip the items of the *darken_factors* argument of the *relief_colors()* function.

the following example shows this for a round/elliptic button widget:

```
MyRoundSunkenButton:
    relief_ellipse_outer_colors: relief_colors(darken_factors=(0.3, 0.6))
```

the other color attributes of this mixin class control the relief colors for the inner part of a square shaped widget (*relief_square_inner_colors*) and for the inner part of an elliptic shape widget (*relief_ellipse_inner_colors*).

the depth of the outer raise/sunk effect can be controlled with the *relief_square_outer_lines* property/attribute. *relief_square_inner_lines* controls the raise/sunk depth of the inner surface of a square widget. *relief_ellipse_inner_lines* and *relief_ellipse_outer_lines* are doing the same for widgets with a round/elliptic shape.

the properties *relief_square_inner_offset* and *relief_ellipse_inner_offset* are specifying the width of the widget border (the part between the outer and the inner relief) in pixels.

Note: at least one of the classes that is mixing in this class has to inherit from Widget (or EventDispatcher) to get the widgets *pos*, *size*, *canvas* properties and the *bind* method.

---

relief_pos_size

list/tuple of optional relief position and size (x, y, width, height) in pixels.

if not specified or empty list/tuple, than the pos/size values of the mixing-in widget will be used instead.

relief_pos_size is a ListProperty and defaults to an empty list.

relief_ellipse_inner_colors: Union[Tuple[Union[Tuple[float, float, float], List[float]], Union[Tuple[float, float, float], List[float]]], Tuple]

list/tuple of ellipse inner (top, bottom) rgb colors.

relief_ellipse_inner_colors is a ObjectProperty and defaults to an empty tuple.

relief_ellipse_inner_lines

number of ellipse inner lines/pixels to be drawn.

relief_ellipse_inner_lines is a NumericProperty and defaults to '3sp'.

relief_ellipse_inner_offset

number of pixels left unchanged at the border of the inner elliptic surface before the inner relief starts.

relief_ellipse_inner_offset is a NumericProperty and defaults to '1sp'.

relief_ellipse_outer_colors: Union[Tuple[Union[Tuple[float, float, float], List[float]], Union[Tuple[float, float, float], List[float]]], Tuple]

list/tuple of ellipse outer (top, bottom) rgb colors.

relief_ellipse_outer_colors is a ObjectProperty and defaults to an empty tuple.

relief_ellipse_outer_lines

number of ellipse outer lines/pixels to be drawn.

relief_ellipse_outer_lines is a NumericProperty and defaults to '3sp'.

relief_square_inner_colors

list/tuple of square inner (top, bottom) rgb colors.

relief_square_inner_colors is a ObjectProperty and defaults to an empty tuple.

relief_square_inner_lines

number of square inner lines/pixels to be drawn.

relief_square_inner_lines is a NumericProperty and defaults to '3sp'.

relief_square_inner_offset

number of pixels left unchanged at the border of the square inner surface before the inner relief starts.

relief_square_inner_offset is a NumericProperty and defaults to '1sp'.

relief_square_outer_colors: Union[Tuple[Union[Tuple[float, float, float], List[float]], Union[Tuple[float, float, float], List[float]]], Tuple]

list/tuple of square outer (top, bottom) rgb colors.

relief_square_outer_colors is a ObjectProperty and defaults to an empty tuple.

relief_square_outer_lines: NumericProperty

number of square outer lines/pixels to be drawn.

relief_square_outer_lines is a NumericProperty and defaults to '3sp'.

bind: Any

canvas: Any

pos: list

size: list

__init__(***kwargs*)

_relief_refresh(**_args*)

pos/size or color changed event handler.

_relief_ellipse_inner_refresh(*add_instruction*, *top_color*, *bottom_color*, *wid_x*, *wid_y*, *wid_width*, *wid_height*)

ellipse pos/size or color changed event handler.

_relief_ellipse_outer_refresh(*add_instruction*, *top_color*, *bottom_color*, *wid_x*, *wid_y*, *wid_width*, *wid_height*)

ellipse pos/size or color changed event handler.

_relief_square_inner_refresh(*add_instruction*, *top_color*, *bottom_color*, *wid_x*, *wid_y*, *wid_width*, *wid_height*)

square pos/size or color changed event handler.

_relief_square_outer_refresh(*add_instruction*, *top_color*, *bottom_color*, *wid_x*, *wid_y*, *wid_width*, *wid_height*)

square pos/size or color changed event handler.

4.32 ae.kivy

4.32.1 core application classes and widgets for GUIApp-conform Kivy apps

this ae portion is implementing the Kivy-framework-specific parts for apps with multilingual context-sensitive help, user onboarding, tours, walkthroughs and tutorials.

by extending and joining the app classes *MainAppBase*, *HelpAppBase* and App, it is providing additional *config variables*, some useful constants, behaviors and widgets for your multi-platform apps.

this portion is composed of the following modules:

- *i18n*: internationalization (i18n) function *get_txt()* for python and kv code
- *behaviors*: widget behavior classes
- *widgets*: generic widget classes and some useful constants
- *tours*: app tour widget classes
- *apps*: providing the two application classes (*FrameworkApp* and *KivyMainApp*)

unit tests

unit tests are currently still incomplete and need at least V 2.0 of OpenGL and the Kivy framework installed.

Note: unit tests are currently not passing at the gitlab CI because is failing to set up a properly running OpenGL graphics/window system on the python image that all ae portions are using.

4.33 ae.kivy.i18n

4.33.1 ae.kivy.i18n module

this module is adding translatable f-strings to the python and kv code of your app, via the helper function *get_txt()* and the *_GetTextBinder* class.

Module Attributes

get_txt(text[, count, language, loc_vars])	instantiate global i18n translation callable and language switcher helper

class _GetTextBinder

Bases: Observable

redirect *ae.i18n.get_f_string()* to an instance of this class.

kivy currently only support a single one automatic binding in kv files for all function names ending with _ (see *watched_keys* extension in kivy/lang/parser.py line 201; e.g. *f_* would get recognized by the lang_tr re pattern, but kivy will only add the _ symbol to watched_keys and therefore *f_* not gets bound.) to allow both - f-strings and simple get_text messages - this module binds *ae.i18n.get_f_string()* to the *get_txt* symbol (instead of *ae.i18n.get_text()*).

get_txt can be used as translation callable, but also to switch the current default language. additionally *get_txt* is implemented as an observer that automatically updates any translations messages of all active/visible kv rules on switch of the language at app run-time.

inspired by (see also discussion at https://github.com/kivy/kivy/issues/1664):

- https://github.com/tito/kivy-gettext-example
- https://github.com/Kovak/kivy_i18n_test
- https://git.bluedynamics.net/phil/woodmaster-trainer/-/blob/master/src/ui/kivy/i18n.py

observers: List[Tuple[Callable, tuple, dict]] = []

list of bound observer tuples (func, args, kwargs)

_bound_uid = -1

fbind(*name*, *func*, **args*, ***kwargs*)

override fbind (fast bind) from Observable to collect and separate _ bindings.

Parameters

- **name** (str) – attribute name to be bound.
- **func** (Callable) – observer notification function (to be called if attribute changes).
- **args** – args to be passed to the observer.
- **kwargs** – kwargs to be passed to the observer.

Return type
int

Returns
unique id of this binding.

funbind(*name*, *func*, **args*, ***kwargs*)

override fast unbind.

Parameters

- **name** (str) – bound attribute name.
- **func** (Callable) – observer notification function (called if attribute changed).
- **args** – args to be passed to the observer.
- **kwargs** – kwargs to be passed to the observer.

switch_lang(*lang_code*)

change language and update kv rules properties.

Parameters

lang_code (str) – language code to switch this app to.

__call__(*text*, *count=None*, *language=''*, *loc_vars=None*, ***kwargs*)

translate text into the current-default or the passed language.

Parameters

- **text** (str) – text to translate.
- **count** (Optional[int]) – optional count for pluralization.
- **language** (str) – language code to translate the passed text to (def=current default language).
- **loc_vars** (Optional[Dict[str, Any]]) – local variables used in the conversion of the f-string expression to a string. the *count* item of this dict will be overwritten by the value of the *count* parameter (if this argument got specified).
- **kwargs** – extra kwargs (e.g. *glo_vars* or *key_suffix* - see *get_f_string()*).

Return type

str

Returns

translated text.

get_txt(*text*, *count=None*, *language=''*, *loc_vars=None*, ***kwargs*) **= <ae.kivy.i18n._GetTextBinder object>**

instantiate global i18n translation callable and language switcher helper

4.34 ae.kivy.widgets

4.34.1 ae.kivy.widgets module

this module provides constants and widgets for your multi-platform apps.

the generic constants for animations and vibration patterns (mostly used on mobile platforms).

most of the widgets provided by this module are based on the widgets of the Kivy framework, extended to work with *app state variables*, e.g. to support app styles and theming (dark or light) and user definable font sizes. some of them also change the *application flow*.

by importing this module the following generic widgets will be registered in the kivy widget class factory maps, to be available in the kv language for your app:

- *AppStateSlider*: extended version of Slider, changing the value of *app state variables*.
- *FlowButton*: button to change the application flow.
- *FlowDropDown*: attachable menu-like popup, based on DropDown.
- *FlowInput*: dynamic kivy widget based on TextInput with application flow support.
- *FlowPopup*: dynamic auto-content-sizing popup to query user input or to show messages.
- *FlowSelector*: attachable popup used for dynamic elliptic auto-spreading menus and toolbars.
- *FlowToggler*: toggle button based on *ImageLabel* and ToggleButtonBehavior to change the application flow or any flag or application state.
- *HelpToggler* is a toggle button widget that switches the app's help and tour mode on and off.
- *ImageLabel*: dynamic kivy widget extending the Kivy Label widget with an image.
- *MessageShowPopup*: simple message box widget based on *FlowPopup*.
- OptionalButton: dynamic kivy widget based on *FlowButton* which can be dynamically hidden.
- ShortenedButton: dynamic kivy widget based on *FlowButton* shortening the button text.
- *Tooltip* displays text blocks that are automatically positioned next to any widget to providing e.g. i18n context help texts or app tour/onboarding info.
- UserNameEditorPopup: popup window used e.g. to enter new user, finally registered in the app config files.

tooltip popup to display context-sensitive help and app tour texts

the tooltip popup widget class *Tooltip* allows you to target any widget by pointing with an arrow to it. the position and size of this widget gets automatically calculated from the targeted widget position and size and the tooltip text size. and if the screen/window size is not big enough then the tooltip texts get scrollable.

Hint: use cases of the class *Tooltip* are e.g. the help texts prepared and displayed by the method *help_display()* as well as the "explaining widget" tooltips in an app tour.

---

help activation and de-activation

use the widget class *HelpToggler* provided by this module to toggle the active state of the help mode.

Hint: the *HelpToggler* class is using the low-level touch events to prevent the dispatch of the Kivy events *on_press*, *on_release* and *on_dismiss*, to allow to show help texts for opened dropdowns and popups, without closing/dismissing them.

---

to attach help texts to your widget instances add the behavior class *HelpBehavior*.

Module Attributes

ANI_SINE_DEEPER_REPEAT3	sine 3 x deeper repeating animation, used e.g.
CRITICAL_VIBRATE_PATTERN	very long/~2.4s vibrate pattern for critical error notification (sending SOS to the mobile world;)
ERROR_VIBRATE_PATTERN	long/~2s vibrate pattern for error notification.
LOVE_VIBRATE_PATTERN	short/~1.2s vibrate pattern for fun/love notification.
MAIN_KV_FILE_NAME	default file name of the main kv file of your app

Classes

AbsolutePosSizeBinder(*widgets[, ...])	propagate changes of *pos*/*size* properties of one or more widgets plus their parents to attributes/callbacks.
AppStateSlider(**kwargs)	slider widget with help text to change app state value.
ExtTextInputCutCopyPaste(**kwargs)	overwrite/extend `kivy.uix.textinput.TextInputCutCopyPaste` w/ translatable and autocomplete options.
FlowButton(**kwargs)	button to change the application flow.
FlowDropDown(**kwargs)	flow based widget class to implement dynamic menu-like user selections and toolbars.
FlowInput(**kwargs)	text input/edit widget with optional autocompletion.
FlowPopup(**kwargs)	popup for dynamic and auto-sizing dialogs and other top-most or modal windows.
FlowSelector(**kwargs)	attachable popup used for dynamic elliptic auto-spreading menus and toolbars.
FlowToggler(**kwargs)	toggle button changing flow id.
HelpToggler(**kwargs)	widget to activate and deactivate the help mode.
ImageLabel(**kwargs)	base label used for all labels and buttons - declared in widgets.kv and also in this module to inherit from.
MessageShowPopup(**kwargs)	flow popup to display info or error messages.
Tooltip(**kwargs)	semi-transparent and optional click-through container to display help and tour page texts.

ANI_SINE_DEEPER_REPEAT3 = <kivy.animation.Sequence object>

sine 3 x deeper repeating animation, used e.g. to animate help layout (see *Tooltip* widget)

CRITICAL_VIBRATE_PATTERN = (0.0, 0.12, 0.12, 0.12, 0.12, 0.12, 0.12, 0.24, 0.12, 0.24, 0.12, 0.24, 0.12, 0.12, 0.12, 0.12, 0.12, 0.12)

very long/~2.4s vibrate pattern for critical error notification (sending SOS to the mobile world;)

ERROR_VIBRATE_PATTERN = (0.0, 0.09, 0.09, 0.18, 0.18, 0.27, 0.18, 0.36, 0.27, 0.45)

long/~2s vibrate pattern for error notification.

LOVE_VIBRATE_PATTERN = (0.0, 0.12, 0.12, 0.21, 0.03, 0.12, 0.12, 0.12)

short/~1.2s vibrate pattern for fun/love notification.

MAIN_KV_FILE_NAME = 'main.kv'

default file name of the main kv file of your app

class AbsolutePosSizeBinder(*widgets, bind_window_size=False)

Bases: object

propagate changes of *pos*/*size* properties of one or more widgets plus their parents to attributes/callbacks.

create an instance of this class passing the widget(s) to observe on change of their pos/size. then call the methods *pos_to_attribute()*, *pos_to_callback()*, *size_to_attribute()* and *size_to_callback()* to specify the propagation of the changed *pos* and/or *size*. to remove the change propagation call the method *unbind()*.

Note: the *pos* attribute/callback propagations are providing absolute window coordinates.

__init__(*widgets, bind_window_size=False)

instantiate binder specifying the monitored widget(s).

Parameters

- **widgets** (Widget) – widget(s) to observe changes of their *pos* and *size* properties. if specified more than one widget then the pos/size coordinates of the rectangle that is enclosing all specified widgets are propagated.
- **bind_window_size** (bool) – pass True to propagate pos and size changes if window size changes.

_bind()

_propagate(*wid*, *value*, *attributes*, *callbacks*)

_wid_pos_changed(*wid*, *new_pos*)

propagate *pos* property change to target attributes and subscribed observers.

Parameters

- **wid** (Widget) – bound widget or a ScrollView that is embedding the bound widget, which pos changed.
- **new_pos** (List[float]) – new position of the bound widget/ScrollView (unused).

_wid_size_changed(*wid*, *new_size*)

propagate *size* property change to target attributes and subscribed observers.

Parameters

- **wid** (Widget) – bound widget or a ScrollView that is embedding the bound widget, which pos changed.
- **new_size** (List[float]) – new position of the bound widget/ScrollView (unused).

_rel_pos_changed(*_rel*, *_new_pos*)

propagate *pos* property change of relative/scrollable layout/container.

Parameters

- **_rel** (Widget) – relative layout or a scroll view, embedding bound widget(s), which pos changed.
- **_new_pos** (list) – new position of the RelativeLayout/ScrollView (unused).

_rel_size_changed(*_rel*, *_new_size*)

propagate size change of relative/scrollable layout/container.

Parameters

- **_rel** (Widget) – relative layout or a scroll view, embedding bound widget(s), which size changed.
- **_new_size** (list) – new size of the RelativeLayout/ScrollView (unused).

pos_to_attribute(*target*, *attribute*, *converter=None*)

request the propagation of the changed (absolute) widget(s) position to an object attribute.

Parameters

- **target** (Any) – the object which attribute will be changed on change of *pos*.
- **attribute** (str) – the name of the attribute to assign the new/changed absolute position.
- **converter** (Optional[Callable[[Widget, List[float]], Any]]) – optional pos value converter, returning the final value assigned to the attribute.

pos_to_callback(*callback*)

bind callable to *pos* change event.

Parameters

callback (Callable[[Widget, List[float]], Any]) – callable to be called when pos changed with the changed widget and pos as arguments.

size_to_attribute(*target*, *attribute*, *converter=None*)

request the propagation of the changed widget(s) size to an object attribute.

Parameters

- **target** (Any) – the object which attribute will be changed on change of *size*.
- **attribute** (str) – the name of the attribute to assign the new/changed size.
- **converter** (Optional[Callable[[Widget, List[float]], Any]]) – optional pos value converter, returning the final value assigned to the attribute.

size_to_callback(*callback*)

bind callable to *size* change event.

Parameters

callback (Callable[[Widget, List[float]], Any]) – callable to be called when size changed with the changed widget and size as arguments.

unbind()

unbind the widget(s) of this binder instance.

Note: this instance can be destroyed after the call of this method. for new bindings create a new instance.

class AppStateSlider(***kwargs*)

Bases: *HelpBehavior*, *ShadersMixin*, Slider

slider widget with help text to change app state value.

app_state_name

name of the app state to be changed by this slider value

__str__()

added for easier debugging.

on_value(*args)

value changed event handler.

Parameters
args¶ – tuple of instance and new value.

class HelpToggler(**kwargs)

Bases: ReliefCanvas, Image

widget to activate and deactivate the help mode.

To prevent dismiss of opened popups and dropdowns at help mode activation, this singleton instance has to:

- be registered in its __init__ to the *help_activator* attribute and
- have a *on_touch_down()* method that is eating the activation touch event (returning True) and
- a *on_touch_down()* method not passing an activation touch in all DropDown/Popup widgets.

ani_value

float value (range: 0.0 - 1.0) to animate this button in help/tour mode

__init__(**kwargs)

initialize an instance of this class and also *help_activator*.

ani_start()

start animation of this button.

ani_stop()

stop animation of this button.

on_touch_down(*touch*)

touch down event handler to toggle help mode while preventing dismiss of open dropdowns/popups.

Parameters
touch¶ (MotionEvent) – touch event.

Return type
bool

Returns
True if touch happened on this button (and will get no further processed => eaten).

class ImageLabel(**kwargs)

Bases: ReliefCanvas, ShadersMixin, Label

base label used for all labels and buttons - declared in widgets.kv and also in this module to inherit from.

Note: hide-able label needs extra handling, because even setting width/height to zero the text can still be visible, especially in dark mode and even with having the text-color-alpha==0. to fully hide the texture in all cases, set either the text to an empty string or the opacity to zero.

__repr__()

added for easier debugging of *FlowButton* and *FlowToggler* widgets.

class FlowButton(**kwargs)

Bases: HelpBehavior, SlideSelectBehavior, TouchableBehavior, ButtonBehavior, *ImageLabel*

button to change the application flow.

long_tap_flow_id

flow id that will be set when this button gets long tap event

tap_flow_id

the new flow id that will be set when this button get tapped

tap_kwargs

kwargs dict passed to event handler (change_flow) when button get tapped

__init__(***kwargs*)

on_long_tap(*touch*)

long tap/click default handler.

Parameters

touch¶ (MotionEvent) – motion/touch event data with the touched widget in *touch.grab_current*.

on_release()

overridable touch release event handler.

class FlowDropDown(***kwargs*)

Bases: *ContainerChildrenAutoWidthBehavior*, *DynamicChildrenBehavior*, *SlideSelectBehavior*, *ReliefCanvas*, DropDown

flow based widget class to implement dynamic menu-like user selections and toolbars.

close_kwargs

kwargs passed to all close action flow change event handlers

content

layout container

menu_items

container/content children, like buttons, text inputs or sliders

parent_popup_to_close

parent popup widget instance to be closed if this dropdown closes

__init__(***kwargs*)

__repr__()

added for easier debugging.

_real_dismiss(**_args*)

overridden to ensure that return value of on_dismiss-dispatch get recognized.

dismiss(**args*)

override DropDown method to prevent dismiss of any dropdown/popup while clicking on activator widget.

Parameters

args¶ – args to be passed to DropDown.dismiss().

close(**args*)

override DropDown method to prevent dismiss of any dropdown/popup while clicking on activator widget.

Parameters

args¶ – args to be passed to DropDown.dismiss().

on_container(*instance*, *value*)

sync *content* widget and *menu_items* list with container widget.

Parameters

- **instance** (Widget) – self.
- **value** (Widget) – new/changed container widget.

on_dismiss()

default dismiss/close default event handler.

Return type
Optional[bool]

Returns
True to prevent/cancel the dismiss/close.

on_touch_down(*touch*)

prevent the processing of a touch on the help activator widget by this dropdown.

Parameters
touch (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

_reposition(**args*)

fixing Dropdown bug - see issue #7382 and PR #7383. TODO: remove if PR gets merged and distributed.

class ExtTextInputCutCopyPaste(***kwargs*)

Bases: TextInputCutCopyPaste

overwrite/extend kivy.uix.textinput.TextInputCutCopyPaste w/ translatable and autocomplete options.

__init__(***kwargs*)

create Bubble instance to display the cut/copy/paste options.

the monkey patch of TextInputCutCopyPaste which was done in *FlowInput._show_cut_copy_paste()* has to be temporarily reset before the super() call below, to prevent endless recursion because else the other super(cls, instance) call (in python2 style within TextInputCutCopyPaste.__init__()) results in the same instance (instead of the overwritten instance).

on_parent(*instance*, *value*)

overwritten to translate BubbleButton texts and to add extra menus to add/delete ac texts.

Parameters

- **instance** (Widget) – self.
- **value** (Widget) – kivy main window.

class FlowInput(***kwargs*)

Bases: *HelpBehavior*, *ShadersMixin*, TextInput

text input/edit widget with optional autocompletion.

until version 0.1.43 of this portion the background and text color of *FlowInput* did automatically get switched by a change of the light_theme app state. now all colors left unchanged (before only the ones with <unchanged>):

```
* background_color: Window.clearcolor              # default: 1, 1, 1, 1
* cursor_color: app.font_color                     # default: 1, 0, 0, 1
* disabled_foreground_color: <unchanged>           # default: 0, 0, 0, .5
* foreground_color: app.font_color                 # default: 0, 0, 0, 1
* hint_text_color: <unchanged>                     # default: 0.5, 0.5, 0.5,␣
↪1.0
* selection_color: <unchanged>                     # default: 0.1843, 0.6549,
↪ 0.8313, .5
```

to implement a dark background for the dark theme we would need also to change the images in the properties:
background_active, background_disabled_normal and self.background_normal.

the images/colors of the bubble that is showing e.g. on long press of the TextInput widget (cut/copy/paste/...) kept unchanged - only the font_size get adapted and the bubble button texts get translated. for that the class *ExtTextInputCutCopyPaste* provided by this portion inherits from the original bubble class `TextInputCutCopyPaste`.

the original bubble class is getting monkey patched shortly/temporarily in the moment of the instantiation to translate the bubble menu options, change the font sizes and add additional menu options to memorize/forget auto-completion texts.

focus_flow_id

flow id that will be set when this widget get focus

unfocus_flow_id

flow id that will be set when this widget lost focus

auto_complete_texts: List[str]

list of autocompletion texts

auto_complete_selector_index_ink: Union[Tuple[float, float, float], List[float], Tuple[float, float, float, float]]

color and alpha used to highlight the currently selected text of all matching autocompletion texts

_ac_dropdown: Any = None

singleton FlowDropDown instance for all TextInput instances

_matching_ac_texts: List[str] = []

one list instance for all TextInput instances is enough

_matching_ac_index: int = 0

index of selected text in the dropdown matching texts list

__init__(***kwargs*)

__repr__()

added for easier debugging.

_change_selector_index(*delta*)

change/update/set the index of the matching texts in the opened autocompletion dropdown.

Parameters
delta¶ (int) – index delta value between old and new index (e.g. pass +1 to increment index). set index to zero if the old/last index was on the last item in the matching list.

_delete_ac_text(*ac_text=''*)

delete_text_from_ac(**_args*)
check if current text is in autocompletion list and if yes then remove it.

called by FlowInput kbd event handler and from menu button added by ExtTextInputCutCopyPaste.on_parent().

Parameters
_args¶ – unused event args.

extend_ac_with_text(**_args*)
add non-empty text to autocompletion texts.

Parameters
_args¶ – unused event args.

keyboard_on_key_down(*window*, *keycode*, *text*, *modifiers*)
overwritten TextInput/FocusBehavior kbd event handler.

Parameters
- **window**¶ (Any) – keyboard window.
- **keycode**¶ (Tuple[int, str]) – pressed key as tuple of (numeric key code, key name string).
- **text**¶ (str) – pressed key value string.
- **modifiers**¶ (List[str]) – list of modifier keys (pressed or locked).

Return type
bool

Returns
True if key event get processed/used by this method.

keyboard_on_textinput(*window*, *text*)
overridden to suppress any user input if tour is running/active.

on_focus(*_self*, *focus*)
change flow on text input change of focus.

Parameters
- **_self**¶ (Widget) – unused dup ref to self.
- **focus**¶ (bool) – True if this text input got focus, False on unfocus.

on_text(*_self*, *text*)
TextInput.text change event handler.

Parameters
- **_self**¶ (Widget) – unneeded duplicate reference to TextInput/self.
- **text**¶ (str) – new/current text property value.

_select_ac_text(*selector*)
put selected autocompletion text into text input and close _ac_dropdown

_show_cut_copy_paste(**args*, ***kwargs*)

Show a bubble with cut copy and paste buttons

class FlowPopup(***kwargs*)

Bases: *ModalBehavior*, *DynamicChildrenBehavior*, *SlideSelectBehavior*, *ReliefCanvas*, BoxLayout

popup for dynamic and auto-sizing dialogs and other top-most or modal windows.

the scrollable *container* (a ScrollView instance) can only have one child, referenced by the *content* attribute, which can be any widget (e.g. a label). use a layout for *content* to display multiple widgets. set *optimal_content_width* and/or *optimal_content_height* to make the popup size as small as possible, using e.g. *minimum_width* respectively *minimum_height* if *content* is a layout that is providing and updating this property, or `text_size_guess()` if it is a label or button widget.

Hint: texture_size could provide a more accurate size than *text_size_guess()*, but should be used with care to prevent recursive property change loops.

---

this class is very simular to Popup and can be used as replacement, incompatible are the following attributes of Popup and ModalView:

- background: FlowPopup has no `BorderImage`.
- border: FlowPopup is using a `RoundedRectangle`.
- title_align: is 'center' and could be changed via the *title_bar* id.
- title_color is the *app.font_color*.
- title_font is the default font.
- title_size is the default button height (*button_height*).

Events

***on_pre_open*:**
fired before the FlowPopup is opened and got added to the main window.

***on_open*:**
fired when the FlowPopup is opened.

***on_pre_dismiss*:**
fired before the FlowPopup is closed.

***on_dismiss*:**
fired when the FlowPopup is closed. if the callback returns True, the popup will stay opened.

close_kwargs

kwargs passed to all close action flow change event handlers.

close_kwargs is a DictProperty. the default depends the action of the penultimate flow id in the `ae.gui_app.flow_path`: is empty or 'enter' dict then it defaults to an empty flow, else to an empty dict.

container: Widget

popup scrollable layout underneath the title bar and the parent of the *content* container.

container is an ObjectProperty and is read-only.

content

popup main content container, displayed as a child of the scrollable layout *container*.

content is an ObjectProperty and has to be specified either in the kv language as children or via the *content* kwarg.

menu_items

sequence of the content widgets and close button.

menu_items is an ObjectProperty and includes by default the content widgets as well as the close button of this popup.

optimal_content_width

width of the content to be fully displayed/visible.

optimal_content_width is a NumericProperty. if *0* or *None* or not explicitly set then it defaults to the main window width and - in landscape orientation - minus the *side_spacing* and the width needed by the *query_data_maps* widgets.

optimal_content_height

height of the content to be fully displayed/visible.

optimal_content_height is a NumericProperty. if *0* or *None* or not explicitly set then it defaults to the main window height minus the height of *title* and - in portrait orientation - minus the *side_spacing* and the height needed by the *query_data_maps* widgets.

parent_popup_to_close

parent popup widget instance to be closed if this popup closes.

parent_popup_to_close is a ObjectProperty and defaults to None.

query_data_maps: List[Dict[str, Any]]

list of child data dicts to instantiate the query widgets (most likely *FlowButton*) of this popup.

query_data_maps is a ListProperty and defaults to an empty list.

separator_height

height of the separator.

separator_height is a NumericProperty and defaults to 3sp.

side_spacing

padding in pixels from Window.width in landscape-orientation, and from Window.height in portrait-orientation.

side_spacing is a NumericProperty and defaults to 192sp.

title

title string of the popup.

title is a StringProperty and defaults to an empty string.

_anim_alpha

internal opacity/alpha for fade-in/-out animations

_anim_duration

internal time in seconds for fade-in/-out animations

_max_height

popup max height (calculated from Window/side_spacing)

_max_width

popup max width (calculated from Window/side_spacing)

__events__ = ('on_pre_open', 'on_open', 'on_pre_dismiss', 'on_dismiss')

__init__(***kwargs*)

background_color

background ink tuple in the format (red, green, blue, alpha).

the *background_color* is a ColorProperty and defaults to `clearcolor`.

overlay_color

ink (color + alpha) tuple in the format (red, green, blue, alpha) used for dimming of the main window.

overlay_color is a ColorProperty and defaults to the current color value `clearcolor` with an alpha of 0.6 (set in *__init__()*).

separator_color

color used by the separator between title and the content-/container-layout.

separator_color is a ColorProperty and defaults to the current value of the `font_color` property.

__repr__()

added for easier debugging.

add_widget(*widget*, *index=0*, *canvas=None*)

add container and content widgets.

first call set container from kv rule, 2nd the content, 3rd raise error.

Parameters

- **widget** (Widget) – widget instance to be added.
- **index** (int) – index kwarg of `kivy.uix.widget.Widget()`.
- **canvas** (Optional[str]) – canvas kwarg of `kivy.uix.widget.Widget()`.

close(**_args*, ***kwargs*)

close/dismiss container/layout (ae.gui_app popup handling compatibility for all GUI frameworks).

Note: prevents close/dismiss of any dropdown/popup while clicking on help activator widget.

Parameters

- **_args** – arguments (to have compatible signature for DropDown/Popup/ModalView widgets).
- **kwargs** – keyword arguments (compatible signature for DropDown/Popup/ModalView widgets).

dismiss(**_args*, ***kwargs*)

alias method of *close()*

on__anim_alpha(*_instance*, *value*)

_anim_alpha changed event handler.

on_content(*_instance*, *value*)

optional single widget (to be added to the container layout) set directly or via FlowPopup kwargs.

on_dismiss()

default dismiss/close event handler.

Return type
Optional[bool]

Returns
return True to prevent/cancel the dismiss/close.

on_open()

open default event handler.

on_pre_dismiss()

pre close/dismiss event handler.

on_pre_open()

pre open default event handler.

open(**_args*, ***kwargs*)

start optional open animation after calling open method if exists in inheriting container/layout widget.

Parameters

- **_args** – unused argument (to have compatible signature for Popup/ModalView and DropDown widgets passing the parent widget).
- **kwargs** – extra arguments that are removed before to be passed to the inheriting open method:
  - 'animation': *False* will disable the fade-in-animation (default=True).

class FlowSelector(***kwargs*)

Bases: *ModalBehavior*, *DynamicChildrenBehavior*, *FlowButton*

attachable popup used for dynamic elliptic auto-spreading menus and toolbars.

this app flow based menu-like popup consists of a central button and animated elliptic-auto-spreading menu items.

any widget class can be used for the menu items of this class, although ShortenedButton instances are best-prepared to auto-shorten the *text* property.

Events

***on_pre_open*:**
fired before the FlowSelector is opened and got added to the main window.

***on_open*:**
fired when the FlowSelector is opened.

***on_pre_dismiss*:**
fired before the FlowSelector is closed.

***on_dismiss*:**
fired when the FlowSelector is closed. if the callback returns True, the menu will stay opened.

inspired by https://github.com/kivy-garden/garden.modernmenu

attached_widget

widget from which this instance got opened.

The *open()* method will automatically set this property whilst *close()* will set it back to None.

close_kwargs

kwargs passed to all close action flow change event handlers.

close_kwargs is a DictProperty. the default depends the action of the penultimate flow id in the ae.gui_app.flow_path: is empty or 'enter' dict then it defaults to an empty flow, else to an empty dict.

is_open

True if the *open()* method of this instance got called. *close()* sets this value to *False*.

is_open is a BooleanProperty and defaults to False.

parent_popup_to_close

parent menu/popup widget instance to be closed if this menu closes.

parent_popup_to_close is a ListProperty and defaults to an empty list.

radian_offset

start/end angle offset (in radians) for the elliptically positioned items of an elliptic menu.

radian_offset is a NumericProperty and defaults to 9 degrees (as radian, respectively `tau * 9.0 / 360.0`).

separator_height

line width of the border of the menu-back-button and of the connectors between the menu and its items.

separator_height is a NumericProperty and defaults to 1sp.

scale_x

spread/widen factor of the menu item ellipse in x direction.

scale_x is a NumericProperty and defaults to 1.0.

scale_y

spread/widen factor of the menu item ellipse in y direction.

scale_y is a NumericProperty and defaults to 1.0.

_anim_alpha

internal opacity/alpha for fade-in/-out animations

_anim_duration

internal time in seconds for fade-in/-out animations

_creation_direction = 1.0

creation-ellipse-direction of the menu buttons/items

_start_radian = 0.0

angle of the first menu item

_item_radian = 0.0

angle of each menu item

__events__ = ('on_pre_open', 'on_open', 'on_pre_dismiss', 'on_dismiss')

__init__(***kwargs*)

overlay_color

ink (color + alpha) tuple in the format (red, green, blue, alpha) used for dimming of the main window.

overlay_color is a ColorProperty and defaults to the current color value `clearcolor` with an alpha of 0.6 (set in *__init__()*).

separator_color

color used to draw the border of the menu-back-button.

separator_color is a ColorProperty and defaults to the current value of the `font_color` property.

container: Widget

parent widget of the menu items (for compatibility with other popups/dropdowns).

container is an ObjectProperty and is read-only.

menu_items

sequence of the menu items widgets (for compatibility with *SlideSelectBehavior*).

menu_items is an ListProperty and defaults to the items specified via the *child_data_maps* property and the kv language.

_attached_pos(*widget*, *_pos*)

_attached_size(*widget*, *_size*)

_finalize_close(**_args*)

final real dismiss after animations are finished.

_item_moved(*_anim*, *item*, *progress*)

draw line from menu-center to animated item-center.

_layout_items(**_args*)

calculate the radians/angles and positions of the menu-items.

given an offset angle of 9 degrees the (start-angle direction menu-item-angle) for the 9 block regions from top-left … center/middle … bottom-right would be:

Table 4.4: possibly blocked window regions

	left	center	right
top	0 -72	180+180	180 +72
middle	81-162	90+360	99+162
bottom	72 -72	180-180	108 +72

add_widget(*widget*, *index=0*, *canvas=None*)

manage children added via kv, python and *ae.kivy_dyn_chi.DynamicChildrenBehavior*.

close(**_args*, ***kwargs*)

close/dismiss menu (ae.gui_app popup handling compatibility for all GUI frameworks).

Note: prevents close/dismiss of any dropdown/popup while clicking on help activator widget.

Parameters

- **_args**¶ – arguments (to have compatible signature for DropDown/Popup/ModalView widgets).

- **kwargs¶ – keyword arguments (compatible signature for DropDown/Popup/ModalView widgets): *force*: pass *True* to force closing, ignoring return value of *dispatch('on_dismiss') animation*: pass False to close this menu without fade-out animation

dismiss(**_args*, ***kwargs*)

alias method of *close()*

on_dismiss()

dismiss/close default event handler.

Return type
Optional[bool]

Returns
return True to prevent/cancel the dismiss/close.

on_open()

open default event handler.

on_pre_dismiss()

pre close/dismiss event handler.

on_pre_open()

pre open default event handler.

on_release()

touch release default event handler.

open(*attach_to*, ***kwargs*)

display flow selector menu items, with animation and as a popup in modal mode.

Parameters

- **attach_to¶** (Widget) – the widget to which this menu gets attached to.
- **kwargs¶** (Dict[str, Any]) – extra arguments that are removed before to be passed to the inheriting open method:
  - 'animation': *False* will disable the fade-in-animation (default=True).

remove_widget(*widget*)

sync self.menu_items with self.children.

touch_pos_is_inside(*pos*)

is touch inside of this widget or a group of sub-widgets. overwritten to also include the menu items.

Parameters
pos¶ (List[float]) – touch position (x, y) in window coordinates.

Return type
bool

Returns
True if this menu and its items would process a touch event at pos.

class FlowToggler(***kwargs*)

Bases: *HelpBehavior*, *SlideSelectBehavior*, *TouchableBehavior*, ToggleButtonBehavior, *ImageLabel*

toggle button changing flow id.

long_tap_flow_id

flow id that will be set when this button gets long tap event

tap_flow_id

the new flow id that will be set when this toggle button get released

tap_kwargs

kwargs dict passed to event handler (change_flow) when button get tapped

__init__(***kwargs*)

on_long_tap(*touch*)

long tap/click default handler.

Parameters

touch¶ (MotionEvent) – motion/touch event data with the touched widget in *touch.grab_current*.

on_release()

overridable touch release event handler.

class MessageShowPopup(***kwargs*)

Bases: *FlowPopup*

flow popup to display info or error messages.

_container: Widget

attach_to: Optional[Widget]

_layout_finished: bool

_opened_item: Optional[Widget]

_touch_moved_outside: bool

message

popup window message text to display

title

popup window title text to display

class Tooltip(***kwargs*)

Bases: ScrollView

semi-transparent and optional click-through container to display help and tour page texts.

tip_text

tooltip text string to display.

tip_text is a StringProperty and defaults to an empty string.

anchor_spe

anchor pos and direction, see *AnchorSpecType* (read-only)

has_tour

True if a tour exists for the current app flow/help context (read-only)

tap_thru

True if user can tap widgets behind/covered by this tooltip win (read-only)

tour_start_pos

screen position of the optionally displayed tour start button (read-only)

tour_start_size

size of the optionally displayed tour start button (read-only)

__init__(***kwargs*)

targeted_widget

target widget to display tooltip text for (mostly a button, but could any, e.g. a layout widget).

targeted_widget is a ObjectProperty and defaults to the main app help_activator.

_actual_pos(**_args*)

Return type
Tuple[float, float]

collide_tap_thru_toggler(*touch_x*, *touch_y*)

check if touch is on the tap through toggler pseudo button.

Parameters

- **touch_x** (float) – window x position of touch.
- **touch_y** (float) – window y position of touch.

Return type
bool

Returns
True if user touched the tap through toggler.

collide_tour_start_button(*touch_x*, *touch_y*)

check if touch is on the tap through toggler pseudo button.

Parameters

- **touch_x** (float) – window x position of touch.
- **touch_y** (float) – window y position of touch.

Return type
bool

Returns
True if user touched the tap through toggler.

on_size(**_args*)

(re-)position help_activator tooltip correctly after help text loading and layout resizing.

on_targeted_widget(**_args*)

targeted widget changed event handler.

Parameters
_args – change event args (unused).

on_touch_down(*touch*)

check for additional events added by this class.

Parameters
touch (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

4.35 ae.kivy.apps

4.35.1 ae.kivy.apps module

this module is providing two application classes, one of them extending the Kivy App class. the other app class is used as main app class, extending *HelpAppBase* with additional attributes and helper methods.

application classes

the class *KivyMainApp* is implementing a main app class, reducing the amount of code needed to create a Python application based on the Kivy framework.

KivyMainApp is based on the following classes:

- the abstract base class *HelpAppBase* which adds context-sensitive help.
- the abstract base class *MainAppBase* which adds *application status*, *app state variables*, *app state constants*, *application flow* and *application events*.
- *ConsoleApp* is adding *config files*, *config variables* and *config options*.
- *AppBase* is adding *application logging* and *application debugging*.

this namespace portion is also encapsulating the Kivy App class via the *FrameworkApp* class. this Kivy app class instance can be directly accessed from the main app class instance via the *framework_app* attribute.

kivy app config variables

all the *config variables* and app constants inherited from the base app classes are available.

Hint: please see the documentation of the namespace portions/modules *ae.console*, *ae.gui_app* and *ae.gui_help* for more detailed information on all the inherited *config variables*, *config options*, *config files* and *app state constants*.

---

the additional *config variables* *win_min_width* and *win_min_height*, added by this portion, you can optionally restrict the minimum size of the kivy main window of your app. their default values are set on app startup in the method *on_app_run()*.

more constants provided by this portion are declared in the *widgets* module.

kivy application events

the main app class is firing *application events*, additional to the ones provided by *MainAppBase*, by redirecting events of Kivy's App class. these framework app events get fired after the event *on_app_run()*, in the following order (the Kivy event/callback-method name is given in brackets):

- on_app_build (kivy.app.App.build, after the main kv file get loaded).
- on_app_built (kivy.app.App.build, after the root widget get build).
- on_app_start (kivy.app.App.on_start)
- on_app_started (one clock tick after on_app_start/kivy.app.App.on_start)
- on_app_pause (kivy.app.App.on_pause)
- on_app_resume (kivy.app.App.on_resume)
- on_app_stop (kivy.app.App.on_stop)
- on_app_stopped (one clock tick after on_app_stop)

Classes

FrameworkApp(main_app, **kwargs)	Kivy framework app class proxy redirecting events and callbacks to the main app class instance.
KivyMainApp(**console_app_kwargs)	Kivy application

class FrameworkApp(*main_app*, ***kwargs*)

Bases: App

Kivy framework app class proxy redirecting events and callbacks to the main app class instance.

app_states

duplicate of MainAppBase app state for events/binds

button_height

default button height, dynamically calculated from font size

displayed_help_id

help id of the currently explained/help-target widget

font_color

rgba color of the font used for labels/buttons/…

help_layout

layout widget if help mode is active else None

landscape

True if app win width is bigger than the app win height

max_font_size

maximum font size in pixels bound to window size

min_font_size

minimum - " -

mixed_back_ink

background color mixed from available back inks

tour_layout

overlay layout widget if tour is active else None

__init__(*main_app*, ***kwargs*)

init kivy app

main_app

set reference to KivyMainApp instance

build()

kivy build app callback.

Return type
Widget

Returns
root widget (Main instance) of this app.

key_press_from_kivy(*keyboard*, *key_code*, *_scan_code*, *key_text*, *modifiers*)

convert and redistribute key down/press events coming from Window.on_key_down.

Parameters

- **keyboard** (Any) – used keyboard.
- **key_code** (int) – key code of pressed key.
- **_scan_code** (int) – key scan code of pressed key.
- **key_text** (Optional[str]) – key text of pressed key.
- **modifiers** (List[str]) – list of modifier keys (including e.g. 'capslock', 'numlock', ...)

Return type
bool

Returns
True if key event got processed used by the app, else False.

key_release_from_kivy(*keyboard*, *key_code*, *_scan_code*)

key release/up event.

Return type
bool

Returns
return value of call to *on_key_release* (True if ke got processed/used).

on_pause()

app pause event automatically saving the app states.

emits the *on_app_pause* event.

Return type
bool

Returns
True.

on_resume()

app resume event automatically loading the app states.

emits the *on_app_resume* event.

Return type
bool

Returns
True.

on_start()

kivy app start event.

called after *run_app()* method, after Kivy created the main layout (by calling its build() method) and has attached it to the main window.

emits the events: *on_app_start* and *on_app_started*.

on_stop()

quit app event automatically saving the app states.

emits the *on_app_stopped* event whereas the method *stop_app()* emits the *on_app_stop* event.

win_pos_size_change(*_)

resize handler updates: *win_rectangle*, *landscape*.

class KivyMainApp(**console_app_kwargs*)

Bases: *HelpAppBase*

Kivy application

documents_root_path: str = '.'

root file path for app documents, e.g. for import/export

get_txt_(*text*, *count=None*, *language=''*, *loc_vars=None*, ***kwargs*): Any = **<ae.kivy.i18n._GetTextBinder object>**

make i18n translations available via main app instance

kbd_input_mode: str = 'scale'

optional app state to set Window[Base].softinput_mode

tour_overlay_class

Kivy main app tour overlay class

alias of *TourOverlay*

_debug_enable_clicks: int = 0

init_app(*framework_app_class=<class 'ae.kivy.apps.FrameworkApp'>*)

initialize framework app instance and prepare app startup.

Parameters
framework_app_class¶ (Type[*FrameworkApp*]) – class to create app instance (optionally extended by app project).

Return type
Tuple[Optional[Callable], Optional[Callable]]

Returns
callable to start and stop/exit the GUI event loop.

app_env_dict()

collect run-time app environment data and settings.

Return type
Dict[str, Any]

Returns
dict with app environment data/settings.

call_method_delayed(*delay*, *callback*, **args*, ***kwargs*)

delayed call of passed callable/method with args/kwargs catching and logging exceptions preventing app exit.

Parameters

- **delay** (float) – delay in seconds before calling the callable/method specified by *callback*.
- **callback** (Union[Callable, str]) – either callable or name of the main app method to call.
- **args** – args passed to the callable/main-app-method to be called.
- **kwargs** – kwargs passed to the callable/main-app-method to be called.

Return type
Any

Returns
delayed call event (in Kivy of Type[ClockEvent]) providing a *cancel* method to allow the cancellation of the delayed call within the delay time.

change_light_theme(*light_theme*)

change font and window clear/background colors to match 'light'/'black' themes.

Parameters
light_theme (bool) – pass True for light theme, False for black theme.

static class_by_name(*class_name*)

resolve kv widgets

Return type
Optional[Type]

static dpi_factor()

dpi scaling factor - overridden to use Kivy's dpi scaling.

Return type
float

ensure_top_most_z_index(*widget*)

ensure visibility of the passed widget to be the foremost in the z index/order.

Parameters
widget (Widget) – widget to check and possibly correct to be the foremost one.

if other dropdown/popup opened after the passed widget/layout, then only correct z index/order to show this widget/layout as popup (in front, as foremost widget). if the passed widget has a method named *activate_modal* (like e.g. *ae.kivy.behaviors.ModalBehavior.activate_modal()*) then it will be called.

global_variables(***patches*)

overridden to add Kivy-specific globals.

Return type
Dict[str, Any]

help_activation_toggle()

button tapped event handler to switch help mode between active and inactive (also inactivating tour).

load_sounds()

override to preload audio sounds from app folder snd into sound file cache.

on_app_build()

kivy App build event handler called at the beginning of kivy.app.App.build().

on_app_built()

kivy App build event handler called at the end of kivy.app.App.build().

on_app_pause()

kivy on_pause() event handler.

on_app_resume()

kivy on_resume() event handler.

on_app_run()

run app event handler - used to set the user preference app states and initial window pos and size.

on_app_start()

app start event handler - triggered by FrameworkApp.on_start().

on_app_started()

kivy on_start() event handler (called after on_app_build/on_app_built).

on_app_stopped()

kivy on_stop() event handler (called after on_app_stop).

on_flow_widget_focused()

set focus to the widget referenced by the current flow id.

on_kbd_input_mode_change(*mode*, *_event_kwargs*)

language app state change event handler.

Parameters

- **mode** (str) – the new softinput_mode string (passed as flow key).
- **_event_kwargs** (Dict[str, Any]) – unused event kwargs.

Return type
bool

Returns
True to confirm the language change.

on_lang_code()

language code app-state-change-event-handler to refresh kv rules.

on_light_theme()

theme app-state-change-event-handler.

on_user_preferences_open(*_flow_id*, *_event_kwargs*)

enable debug mode after clicking 3 times within 6 seconds.

Parameters

- **_flow_id** (str) – (unused).
- **_event_kwargs** (Dict[str, Any]) – (unused).

Return type
bool

Returns
False for *on_flow_change()* get called, opening user preferences popup.

play_beep()

make a short beep sound.

play_sound(*sound_name*)

play audio/sound file.

play_vibrate(*pattern=(0.0, 0.09, 0.09, 0.18, 0.18, 0.27, 0.18, 0.36, 0.27, 0.45)*)

play vibrate pattern.

open_popup(*popup_class*, ***popup_kwargs*)

open Popup or DropDown using the *open* method. overwriting the main app class method.

Parameters

- **popup_class** (Type[Union[*FlowPopup*, Popup, DropDown]]) – class of the Popup or DropDown widget.
- **popup_kwargs** – args to be set as attributes of the popup class instance plus an optional *opener* kwarg that will pass the popup opener widget to the popup.open() method; if *opener* gets not specified then the framework window will be used.

Return type
Widget

Returns
created and displayed/opened popup class instance.

text_size_guess(*text*, *font_size=0.0*, *padding=(0.0, 0.0)*)

quickly roughly pre-calculate texture size of a multi-line string without rendering.

Parameters

- **text** (str) – text string which can contain line feed characters.
- **font_size** (float) – the font size to pseudo-render the passed text; using the value of *font_size* as default if not passed.
- **padding** (Tuple[float, float]) – optional padding in pixels for x and y coordinate (totals for left+right/top+bottom).

Return type
Tuple[float, float]

Returns
roughly the size (width, height) to display the string passed into *text*. more exactly size would need to use internal render methods of Kivy, like e.g. `_get_text_width()` and `get_extents()`.

static widget_pos(*wid*)

return widget's window x/y position (overridden for absolute coordinates relative/scrollable layouts).

Parameters
wid¶ – widget to determine the position of.

Return type
Tuple[float, float]

Returns
tuple of x and y screen coordinate.

4.36 ae.kivy.behaviors

4.36.1 ae.kivy.behaviors module

this module provides the following behavior classes:

- *HelpBehavior* extends and prepares any Kivy widget to show an individual help text for it.
- *ModalBehavior* is a generic mix-in class that provides modal behavior to any container widget.
- *SlideSelectBehavior*: quickly navigate in elliptically-shaped sub-/menus, alternatively starting with a long touch, then slide to the menu item to select and release.
- *TouchableBehavior*: extends toggle-/touch-behavior of ButtonBehavior.

help behaviour mixin

to show a i18n translatable help text for a Kivy widget create a subclass of the widget and add the mixin-/behavior-class HelpBehavior. the following example is attaching a help text to the Kivy Button widget:

```
from kivy.uix.button import Button
from ae.kivy.widgets import HelpBehavior

class ButtonWithHelpText(HelpBehavior, Button):
    ...
```

alternatively you can archive this via the definition of a new kv-lang rule, like shown underneath:

```
<ButtonWithHelpText@HelpBehavior+Button>
```

Note: to automatically lock and mark the widget you want to add help texts for, this mixin class has to be specified as the first inheriting class in the class or rule declaration.

modal behavior mixin

to convert a container widget into a modal dialog, add the *ModalBehavior* mix-in class, provided by this ae namespace portion.

the following code snippet demonstrates a typical implementation:

```
class MyContainer(ModalBehavior, BoxLayout):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def open(self):
        self.activate_esc_key_close()
        self.activate_modal()

    def close(self):
        self.deactivate_esc_key_close()
        self.deactivate_modal()
```

calling the method *activate_esc_key_close()* in the *open* method of a container class allows the user to close the popup by pressing the Escape key (or Back on Android). this optional feature can be reverted by calling the *deactivate_esc_key_close()* method in your *close* method.

to additionally activate the modal mode call the method *activate_modal()*. the modal mode can be deactivated by calling the *deactivate_modal()* method.

all touch, mouse and keyboard user interactions will be consumed or filtered after activating the modal mode. therefore it is recommended to also visually change the GUI while in the modal mode, which has to be implemented by the mixing-in container widget.

Hint: usage examples of the *ModalBehavior* mix-in are e.g. the classes *TourOverlay* and *FlowPopup*.

Module Attributes

TOUCH_VIBRATE_PATTERN	very short/~0.3s vibrate pattern for button and toggler touch.

Classes

HelpBehavior()	behaviour mixin class for widgets providing help texts.
ModalBehavior()	mix-in to allow close on Escape/Back key and to optionally provide a modal mode to a container widget.
SlideSelectBehavior(**kwargs)	quickly navigate in sub-/menus, starting with a long touch, then slide to the menu item to select and release.
TouchableBehavior(**kwargs)	touch-/toggle-button mix-in class with shaders, animations and additional events for double/triple/long touches.

TOUCH_VIBRATE_PATTERN = (0.0, 0.09, 0.09, 0.06, 0.03, 0.03)

very short/~0.3s vibrate pattern for button and toggler touch.

class HelpBehavior

Bases: object

behaviour mixin class for widgets providing help texts.

help_id

unique help id of the widget.

The correct identification of each help-aware widget presuppose that the attribute *help_id* has a unique value for each widget instance. This is done automatically for the widgets provided by the module *ae.kivy.widgets* by converting the app flow or app state of these widgets into a help id (see e.g. the implementation of the class *FlowButton*).

help_id is a StringProperty and defaults to an empty string.

help_lock

this property is True if the help mode is active and this widget is not the help target.

help_lock is a BooleanProperty and defaults to the value *False*.

help_vars

dict of extra data to displayed/render the help text of this widget.

The *help_vars* is a dict which can be used to provide extra context data to dynamically generate, translate and display individual help texts.

help_vars is a DictProperty and defaults to an empty dict.

_shader_args

shader internal data / id

collide_point: Callable

on_touch_down(*touch*)

prevent any processing if touch is done on the help activator widget or in active help mode.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

class ModalBehavior

Bases: object

mix-in to allow close on Escape/Back key and to optionally provide a modal mode to a container widget.

to make the container widget's modal state more obvious, add in your container widget an overlay color with an alpha between 0.3 and 0.9, together with the following canvas instructions:

canvas:

Color:
rgba: root.my_overlay_color[:3] + [root.my_overlay_color[-1] if self.is_modal else 0]

Rectangle:
size: Window.size if self.is_modal else (0, 0)

two rectangles will be needed to not overlay/fade-out the help activator button:

canvas:

Color:
rgba: self.my_overlay_color[:3] + [self.my_overlay_color[-1] if self.is_modal else 0]

Rectangle:

size:
Window.width if self.is_modal else 0, Window.height - app.main_app.help_activator.height if self.is_modal else 0

Rectangle:
pos: app.main_app.help_activator.right, app.main_app.help_activator.y width: Window.width - app.main_app.help_activator.width if self.is_modal else 0 height: app.main_app.help_activator.height

center: List

center position of Widget

close: Callable

method to dismiss the container widget (provided by self/container-widget)

collide_point: Callable

method to detect collisions with other widgets of Widget

disabled: bool

disabled property of Widget

fbind: Callable

fast binding method of Widget

funbind: Callable

fast unbinding method of Widget

unbind_uid: Callable

even faster unbinding method of Widget

auto_dismiss

determines if the container is automatically dismissed when the user hits the Esc/Back key or clicks outside it.

auto_dismiss is a BooleanProperty and defaults to True.

is_modal

flag if modal mode is active. use *activate_modal()* and *deactivate_modal()* to change this value.

is_modal is a BooleanProperty and defaults to False.

_center_aligned: bool = False

True if self will be repositioned to Window center

_fast_bound_center_uid: int = 0

if of center (pos and size) fbind/unbind_uid

_touch_started_inside: Optional[bool] = None

flag if touch started inside this widget or -group

_align_center(*_args*)

reposition container to Window center.

Parameters
_args¶ – unused (passed only on bound window resize events)

_on_key_down(*_window*, *key*, *_scancode*, *_codepoint*, *_modifiers*)

close/dismiss this popup if back/Esc key get pressed - allowing stacking with DropDown/FlowDropDown.

activate_esc_key_close()

activate key press handler, calling self.close() if Escape/Back key get pressed.

activate_modal(*align_center=True*)

activate or renew modal mode for the mixing-in container widget.

Parameters
align_center¶ (bool) – pass False to prevent the automatic alignment of center to center on reposition or resize of self or on resize of Window.

deactivate_esc_key_close()

deactivate keyboard event handler, activated via *activate_esc_key_close()*.

deactivate_modal()

de-activate modal mode for the mixing-in container.

on_touch_down(*touch*)

touch down event handler, prevents the processing of a touch on the help activator widget by this popup.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

on_touch_move(*touch*)

touch move event handler.

Return type
bool

on_touch_up(*touch*)

touch up event handler.

Return type
bool

touch_pos_is_inside(*pos*)

check if the touch pos is inside of this widget or a group of sub-widgets.

Parameters
pos¶ (List[float]) – touch position (x, y) in window coordinates.

Return type
bool

Returns
True if this widget or group would process a touch event at pos.

class SlideSelectBehavior(**kwargs)

Bases: object

quickly navigate in sub-/menus, starting with a long touch, then slide to the menu item to select and release.

the slide-select feature of this class allows a quicker select of any menu item, by opening any popup via the *on_long_tap()* event, then move the pointer/finger onto the menu item to select to finally release the touch. to enable this feature specify the touch event in the *touch_event* key of the *popup_kwargs* dict in the *change_flow()* call, e.g. by adding the following lines in your kv code onto the *FlowButton*/*FlowToggler* that is opening the popup:

```
on_long_tap:
    app.main_app.change_flow(id_of_flow('open', 'my_menu'),
    **update_tap_kwargs(self, popup_kwargs=dict(touch_event=args[1])))
```

Note: has to be inherited (to be in the MRO) before ButtonBehavior, respectively ToggleButtonBehavior, for the touch event get grabbed properly.

attach_to: Optional[Widget]

close: Callable

collide_point: Callable

dispatch: Callable

to_widget: Callable

__init__(**kwargs)

set normal pressed state shader on widget initialization.

static _cancel_slide_select_closer(*touch*)

static _cancel_slide_select_opener(*touch*)

_grab_and_open(*touch*, *item*, *first_close*, *_args)

static _ungrab_and_close(*touch*, *popup*, *_args)

on_touch_move(*touch*)

disable long touch on mouse/finger moves.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

on_touch_up(*touch*)

disable long touch on mouse/finger up.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

class TouchableBehavior(**kwargs)

Bases: object

touch-/toggle-button mix-in class with shaders, animations and additional events for double/triple/long touches.

Events

***on_double_tap*:**
fired with the touch-down MotionEvent instance arg when a button get tapped twice within short time.

***on_triple_tap*:**
fired with the touch-down MotionEvent instance arg when a button get tapped three times within short time.

***on_long_tap*:**
fired with the touch-down MotionEvent instance arg when a button get tapped more than 2.4 seconds.

***on_alt_tap*:**
fired with the touch-down MotionEvent instance arg when a button get either double, triple or long tapped.

Note: has to be inherited (to be in the MRO) before ButtonBehavior, respectively ToggleButtonBehavior, for the touch event get grabbed properly.

---

add_shader: Callable

center_x: float

center_y: float

collide_point: Callable

del_shader: Callable

disabled: bool

dispatch: Callable

main_app: Any

state: str

_touch_anim
widget-got-touched-animation

_touch_x
x pos moving in touch animation from initial touch to center pos

_touch_y
y pos moving in touch animation from initial touch to center pos

__events__ = ('on_alt_tap', 'on_double_tap', 'on_long_tap', 'on_triple_tap')

__init__(**kwargs)

set normal pressed state shader on widget initialization.

down_shader

shader running if button is in pressed state *'down'*.

down_shader is a DictProperty and defaults to the `firestorm shader`. set to *None* to not render the default shader on button press/down.

normal_shader

shader running if button is in pressed state *'normal'*.

normal_shader is a DictProperty and defaults to the `plunge wave shader`. set to *None* to not render a shader on button release/up.

static _cancel_long_touch_clock(*touch*)

Return type
bool

on_alt_tap(*touch*)

default handler for alternative tap (double, triple or long tap/click).

Parameters
touch¶ (MotionEvent) – motion/touch event data with the touched widget in *touch.grab_current*.

on_double_tap(*touch*)

double tap/click default handler.

Parameters
touch¶ (MotionEvent) – motion/touch event data with the touched widget in *touch.grab_current*.

on_down_shader(*_args)

button down state shader changed event handler.

on_long_tap(*touch*)

long tap/click default handler.

Parameters
touch¶ (MotionEvent) – motion/touch event data with the touched widget in *touch.grab_current*.

on_normal_shader(*_args)

button normal state shader changed event handler.

on_state(*_widget*, *_value*)

button pressed state changed event handler, switching between *'normal'* and *'down'* state shader.

Parameters

- **_widget**¶ (Any) – button widget (is self).
- **_value**¶ (str) – new state value (either 'normal' or 'down').

on_touch_down(*touch*)

add sound, vibration and animation, check if tour is running and additional double/triple/alt touch events.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

`on_touch_move`(*touch*)

disable long touch on mouse/finger moves.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

`on_touch_up`(*touch*)

disable long touch on mouse/finger up.

Parameters
touch¶ (MotionEvent) – motion/touch event data.

Return type
bool

Returns
True if event got processed/used.

`on_triple_tap`(*touch*)

triple tap/click default handler.

Parameters
touch¶ (MotionEvent) – motion/touch event data with the touched widget in *touch.grab_current*.

`_update_shader`()

update shader on changed shader or button state.

4.37 ae.kivy.tours

4.37.1 ae.kivy.tours module

this module provides the following classes to augment the user interface of your apps with animated product tours, tutorials, walkthroughs and user onboarding/welcome features:

- *AnimatedTourMixin*
- *AnimatedOnboardingTour*
- *TourOverlay*

the class *TourOverlay* is implementing an overlay layout widget to display the animations, shaders, tour page texts, tooltip text and the navigation buttons of an active/running app tour.

the *AnimatedTourMixin* can be mixed-into a tour class that inherits from *TourBase* to extend it with animation and glsl shader features.

the class *AnimatedOnboardingTour* is based on *OnboardingTour* and *AnimatedTourMixin* to extend the generic app onboarding tour class with animations. it provides a generic app onboarding tour that covers the core features, that can be easily extended with app-specific tour pages.

to integrate a more app-specific onboarding tour into your app, simply declare a class with a name composed by the name of your app (`app_name`) in camel-case, followed by the suffix *'OnboardingTour'*.

Module Attributes

PageAnimationType	tuple of a widget id string and an Animation instance/evaluation-expression.
PageAnimationsType	tuple of *PageAnimationType* items
WidgetValues	a key of this dict specifies the name, the dict value the value of a widget property/attribute.
DEF_FADE_OUT_APP	default of tour layout fade out app screen factor

Functions

ani_start_check(ani, wid)	start animation if needed else skip animation start.
animated_widget_values(wid, ani)	determine from a widget the attribute/property values animated/changed by an animation.
restore_widget_values(wid, values)	restore property values of a widget.

Classes

AnimatedOnboardingTour(main_app)	onboarding tour, extended with animations and glsl shaders.
AnimatedTourMixin(main_app)	tour class mixin to add individual shaders to the tour layout and their children widgets.
TourOverlay(main_app[, tour_class])	tour layout/view overlay singleton class to display an active/running modal app tour with optional glsl shaders.

PageAnimationType

tuple of a widget id string and an Animation instance/evaluation-expression.

if the first character of the widget id is a `@` then the `repeat` attribute of the Animation instance will be set to True. the rest of the widget id string specifies the widget to be animated which is either:

- one of the widgets of the *TourOverlay* layout class, identified by the on of the following strings: *'next_but'*, *'page_lbl'*, *'tap_pointer'*, *'prev_but'*, *'title_lbl'*, *'tooltip'*, *'tour_page_texts'*.
- the explained widget if an empty string is given.
- the *TourOverlay* layout class instance for any other string (e.g. *'layout'* or *'overlay'*).

alternative to an animation instance, a evaluation string can be specified. these evaluations allow to use the following globals: Animation (also abbreviated as *A*), `Clock`, *layout*, `sp`, `Window` and a reference to the instance of this app tour via *tour*.

alias of `Tuple`[`str`, `Union`[Animation, `str`]]

PageAnimationsType

tuple of *PageAnimationType* items

alias of Tuple[Tuple[str, Union[Animation, str]], ...]

WidgetValues

a key of this dict specifies the name, the dict value the value of a widget property/attribute.

alias of Dict[str, Union[list, tuple, dict, float]]

DEF_FADE_OUT_APP = 0.39

default of tour layout fade out app screen factor

ani_start_check(*ani*, *wid*)

start animation if needed else skip animation start.

Parameters

- **ani**¶ (Animation) – Animation instance.
- **wid**¶ (Widget) – widget to start/skip the animation for.

animated_widget_values(*wid*, *ani*)

determine from a widget the attribute/property values animated/changed by an animation.

Parameters

- **wid**¶ (Widget) – widget of which the animation property values will get retrieved.
- **ani**¶ (Union[Animation, CompoundAnimation]) – Animation/kivy.animation.CompoundAnimation instance.

Return type
Dict[str, Union[list, tuple, dict, float]]

Returns
dict with widget property names and values.

restore_widget_values(*wid*, *values*)

restore property values of a widget.

Parameters

- **wid**¶ (Widget) – widget of which the animation property values will get restored.
- **values**¶ (Dict[str, Union[list, tuple, dict, float]]) – attribute/property values to restore on the widget.

class AnimatedTourMixin(*main_app*)

Bases: object

tour class mixin to add individual shaders to the tour layout and their children widgets.

layout: Widget

main_app: Any

page_ids: List[str]

page_idx: int

setup_texts: Callable

__init__(*main_app*)

pages_animations: Dict[Optional[str], Tuple[Tuple[str, Union[Animation, str]], ...]]

dict of compound animation instances of the pages of this tour.

the key of this dict is the page id or None (for animations available in all pages of this tour). each value of this dict is of the type *PageAnimationsType*.

pages_shaders: Dict[Optional[str], Tuple[Tuple[str, Dict[str, Any]], ...]]

dict of widget shaders for the pages of this tour.

the key of this dict is the page id or None (for shaders available in all pages of this tour). each value of this dict is a tuple of tuples of widget id and add_shader()-kwargs.

the widget id string specifies the widget to which a shader will be added, which is either:

- one of the widgets of the *TourOverlay* layout class, identified by the on of the following strings: *'next_but'*, *'page_lbl'*, *'tap_pointer'*, *'prev_but'*, *'title_lbl'*, *'tooltip'*, *'tour_page_texts'*.
- the explained widget if an empty string is given.
- the *TourOverlay* layout class instance for any other string (e.g. *'layout'* or *'overlay'*).

before the add_shader()-kwargs dict will be passed to the *add_shader()* method, all their non-string values, specifying as strings, will be evaluated/converted automatically. the evaluation provides the following globals: *layout*, sp, Clock, Window and the *tour* instance.

switch_next_animations: Dict[Optional[str], Tuple[Tuple[str, Union[Animation, str]], ...]]

dict of compound animation instances for the next page switch transition of the pages of this tour.

the key of this dict is the page id or None (for animations available in all pages of this tour). each value of this dict is of the type *PageAnimationsType*.

_add_animations(*animations*)

add animations to the tour page currently displayed in the tour layout/overlay.

Parameters
animations (Tuple[Tuple[str, Union[Animation, str]], ...]) – tuple of tuples of widget id and animation instance/evaluation-string.

Returns
length of the longest animation added (in seconds).

next_page()

overridden to add demo animations before/on switch to the next tour page.

setup_explained_widget()

overridden to bind pos/size of explained widget(s) to the tour layout/overlay placeholder.

Return type
list

Returns
list of explained widget instances.

setup_page_shaders_and_animations()

setup shaders and animations of the current page.

specified in *pages_shaders* and *pages_animations*.

setup_layout()

overridden to set up animations and shaders of the current tour page.

simulate_text_input(*text_input*, *text_to_delay*, *text_to_insert=''*, *deltas=(1.8, 0.6, 0.3)*)

simulate the typing of texts by a user entered into an explained TextInput widget of a tour page.

Parameters

- **text_input** (TextInput) – text input widget, either of type TextInput or *FlowInput*.
- **text_to_delay** (str) – text string to be inserted delayed by the seconds specified in deltas[0].
- **text_to_insert** (str) – text string to be inserted directly into the passed text input widget.
- **deltas** (Tuple[float, ...]) – delay deltas in seconds between each character to simulate text inputted by a user. first delta default is a bit higher to finish navigation button y-pos-animation.

tap_animation(*wid_id=''*, *pos_delay=2.34*, *press_delay=0.69*, *release_delay=0.39*)

create a compound animation instance simulating a user touch/tap on the specified widget.

Parameters

- **wid_id** (str) – specifies the widget to be tap simulated: either a widget id string (first item of the *PageAnimationType* tuple), or (if prefixed with a column character) tap/focus/ state id of a widget, or an empty string (specifies the currently explained widget).
- **pos_delay** (float) – time in seconds to position/move the pointer from the next button to the widget.
- **press_delay** (float) – time in seconds of the button press simulation animation.
- **release_delay** (float) – time in seconds of the button release simulation animation.

Return type
Tuple[str, Union[Animation, str]]

Returns
compound animation instance simulating a tap.

Note: use as animation evaluation expression, to get the widget values on setup-time of the page (not tour).

---

teardown_shaders_and_animations()

teardown all added shaders and animations of current tour page (including switch next page animations).

teardown_app_flow()

overridden to teardown the animations of the current/last-shown tour page.

class AnimatedOnboardingTour(*main_app*)

Bases: *AnimatedTourMixin*, *OnboardingTour*

onboarding tour, extended with animations and glsl shaders.

__init__(*main_app*)

overridden to handle onboarding tour starts since app installation.

next_page()

overriding to remove next button size animation only visible in the first tour after app re/start.

setup_layout()

overridden to update layout texts if app window/screen orientation (app.landscape) changes.

teardown_shaders_and_animations()

overridden to unbind setup_texts() on leaving the responsible_layout tour page.

layout: Widget

tour overlay layout instance

main_app: Any

shortcut to main app instance

page_ids: List[str]

list of tour page ids, either initialized via this class attribute or dynamically.

page_idx: int

index of the current tour page (in *page_ids*)

_added_animations: List[Tuple[Widget, Animation, WidgetValues]]

_added_shaders: List[Tuple[Widget, ShaderIdType]]

pages_animations: Dict[Optional[str], PageAnimationsType]

dict of compound animation instances of the pages of this tour.

the key of this dict is the page id or None (for animations available in all pages of this tour). each value of this dict is of the type *PageAnimationsType*.

pages_shaders: Dict[Optional[str], Tuple[Tuple[str, ShaderIdType], ...]]

dict of widget shaders for the pages of this tour.

the key of this dict is the page id or None (for shaders available in all pages of this tour). each value of this dict is a tuple of tuples of widget id and add_shader()-kwargs.

the widget id string specifies the widget to which a shader will be added, which is either:

- one of the widgets of the *TourOverlay* layout class, identified by the on of the following strings: *'next_but'*, *'page_lbl'*, *'tap_pointer'*, *'prev_but'*, *'title_lbl'*, *'tooltip'*, *'tour_page_texts'*.
- the explained widget if an empty string is given.
- the *TourOverlay* layout class instance for any other string (e.g. *'layout'* or *'overlay'*).

before the add_shader()-kwargs dict will be passed to the *add_shader()* method, all their non-string values, specifying as strings, will be evaluated/converted automatically. the evaluation provides the following globals: *layout*, sp, Clock, Window and the *tour* instance.

switch_next_animations: Dict[Optional[str], PageAnimationsType]

dict of compound animation instances for the next page switch transition of the pages of this tour.

the key of this dict is the page id or None (for animations available in all pages of this tour). each value of this dict is of the type *PageAnimationsType*.

class TourOverlay(*main_app*, *tour_class=None*, ***kwargs*)

Bases: *ModalBehavior*, *ShadersMixin*, FloatLayout

tour layout/view overlay singleton class to display an active/running modal app tour with optional glsl shaders.

ani_value

animated float value between 0.0 and 1.0, used e.g. by *AnimatedTourMixin.pages_animations*.

ani_value is a NumericProperty and is read-only.

explained_pos

window position (absolute x, y window coordinates) of the targeted/explained/highlighted widget.

explained_pos is a ListProperty and is read-only.

explained_size

widget size (width, height) of the targeted/explained/highlighted widget.

explained_size is a ListProperty and is read-only.

fade_out_app

fade out app screen factor: 0.0 prevents fade out of the areas around TourPageTexts and the explained widget.

1.0 results in maximum app screen fade out. configurable for individual tour page via *page_data['fade_out_app']*.

fade_out_app is a NumericProperty and defaults to 0.39.

label_height

height in pixels of the page text labels and text lines.

label_height is a NumericProperty and is read-only.

navigation_disabled

if this flag is True then the back/next buttons in the tour layout/overlay are disabled.

navigation_disabled is a BooleanProperty and is read-only.

close: Callable

method to dismiss the container widget (provided by self/container-widget)

tour_instance

holding the *TourBase* instance of the current tour, initialized by *start_tour()*.

tour_instance is a ObjectProperty and is read-only.

__init__(*main_app*, *tour_class=None*, ***kwargs*)

prepare app and tour overlay (singleton instance of this class) to start tour.

Parameters

- **main_app**¶ (*HelpAppBase*) – main app instance.
- **tour_class**¶ (Optional[Type[*TourBase*]]) – optional tour (pages) class, default: tour class of current help id or OnboardingTour.

explained_widget

explained widget instance on actual tour (page).

explained_widget is a ObjectProperty and is read-only.

next_page()

switch to next tour page.

on_navigation_disabled(**_args*)

navigation button disabled change event, used to hide page texts (blend-in-anim in page_updated()).

page_updated()

callback from `setup_layout()` for UI-specific patches, after tour layout/overlay setup.

prev_page()

switch to previous tour page.

start_tour(*tour_cls=None*)

reset app state and prepare tour to start.

Parameters
tour_cls¶ (Optional[Type[*TourBase*]]) – optional tour (pages) class, default: tour of currently shown help id or OnboardingTour.

Return type
bool

Returns
True if tour exists and got started.

stop_tour()

stop tour and restore the initially backed-up app state.

4.38 ae.kivy_auto_width

4.38.1 automatic width mix-in classes for kivy widgets

this ae portion is providing classes to mix them into any kivy widget with a *texture* property (like e.g. `Label` or `Button`), to automatically size and resize widgets and/or to display a long text as scrolling ticker within a tall widget.

automatic font size iteration with animation

mix-in the *AutoFontSizeBehavior* into any kivy widget with a *texture* property, to automatically grow or shrink the font size to fully fill the width/height of the widget with the size of their texture.

more details see in the documentation of the *AutoFontSizeBehavior* class.

automatic container width with opening animation

the class *ContainerChildrenAutoWidthBehavior* determines the optimal width of a container widget, so that the text of any children widget is fully visible/displayed.

the optimal container width is determined by increasing width of the container in iterations, which are implemented through a kivy `Animation`. as soon as the texts of all children are fully displayed (or the maximum width is reached) the animation stops.

more details see in the documentation of the *ContainerChildrenAutoWidthBehavior* class.

automatic ticker animation

mix-in the *SimpleAutoTickerBehavior* class to automatic slide the texture of a widget if it is too big to be completely/fully displayed, like in a news-ping-pong-ticker.

for more details check the documentation of the *SimpleAutoTickerBehavior* class.

Classes

AutoFontSizeBehavior(**kwargs)	mix-in to interpolate the optimal font size so that the texture is filling the full width of the widget.
ContainerChildrenAutoWidthBehavior()	detect minimum width for the complete display of the textures of all children at opening with animation.
SimpleAutoTickerBehavior(**kwargs)	mix-in class to slide texture in a ping-pong-like-ticker animation, if too long to be displayed completely.

class AutoFontSizeBehavior(***kwargs*)

Bases: object

mix-in to interpolate the optimal font size so that the texture is filling the full width of the widget.

the desired spacing (left plus right) between the texture border and the widget borders can be set via the *auto_font_text_spacing* property. additional padding can be added via the *padding_x* property of the mixing-in widget.

to disable the animation set the length of the *auto_font_anim_duration* to zero or very short value.

the minimum and maximum of the texture font size can be restricted by setting the attributes *auto_font_min_size* and *auto_font_max_size*.

bind: Callable

font_size: float

texture_size: tuple

texture_update: Callable

width: float

height: float

auto_font_anim_duration: float

duration in seconds of the font size grow/shrink animation.

auto_font_anim_duration is a NumericProperty and defaults to 0.9 seconds.

auto_font_text_spacing: float

horizontal padding in pixels between widget and texture width (including the additional horizontal *padding_x*).

auto_font_text_spacing is a NumericProperty and defaults to 18sp.

_font_size_anim: Optional[Animation] = None

_font_anim_mode: int = 0

_last_font_size: float = 0.0

__init__(**kwargs)

auto_font_max_size: float

maximum font size.

auto_font_max_size is a NumericProperty and defaults to min(*MAX_FONT_SIZE*, *max_font_size*).

auto_font_min_size: float

minimum font size.

auto_font_min_size is a NumericProperty and defaults to max(*MIN_FONT_SIZE*, *min_font_size*).

_font_size_adjustable()

check if font size need/has to be adjustable.

_start_font_anim(*_args)

delayed anim check

_stop_font_anim()

_font_size_progress(*_anim*, *_self*, *_progress*)

animation on_progress event handler.

class ContainerChildrenAutoWidthBehavior

Bases: object

detect minimum width for the complete display of the textures of all children at opening with animation.

this mix-in class can be added to any type of container or layout widget to provide a consistent API with *open()* and *close()* methods, a *on_complete_opened()* event and a *container* attribute.

Note: a *container* attribute will be automatically created for container classes without it.

---

the animation starts when the *open()* method get called. this call will be forwarded via *super()* to the container if it has an *open* method.

at animation start the width of this container will be set to value of the *auto_width_start* attribute. then the container width increases via the running animation until, either:

- the container width is greater than the value of the *auto_width_minimum* attribute and the textures of all children are fully visible or
- the container width reaches the app window width minus the window padding specified in the *auto_width_window_padding* attribute.

the window width gets bound to the container width to ensure proper displaying if the window width changes.

Events

***on_complete_opened*:**
fired when the container width animation is finished or stopped because all children are fully visible.

container: Widget

widget to add the dynamic children to (provided by the widget to be mixed into)

dismiss: Callable

optional method provided by the widget to be mixed into

dispatch: Callable

event dispatch method, provided by the widget to be mixed into

opacity: float

opacity of the widget to be mixed into

parent: Widget

parent of this widget/container.

width: float

width of the widget to be mixed into (mostly a parent of self.container)

auto_width_anim_duration: float

duration in seconds of the auto-width-animation.

auto_width_anim_duration is a NumericProperty and defaults to 0.9 seconds.

auto_width_window_padding: float

horizontal padding in pixels between the window and the container.

auto_width_window_padding is a NumericProperty and defaults to 96sp.

auto_width_minimum: float

minimum container width in pixels (before the width animation will be stopped).

auto_width_minimum is a NumericProperty and defaults to 369sp.

auto_width_child_padding: float

horizontal padding in pixels between child widget and child texture.

auto_width_child_padding is a NumericProperty and defaults to 87sp.

auto_width_start: float

container width in pixels at the start of the width animation.

auto_width_start is a NumericProperty and defaults to 3sp.

_width_anim: Animation = None

_complete_width: float = 0.0

__events__ = ('on_complete_opened',)

close(**_args*, ***_kwargs*)

close/dismiss container/layout (ae.gui_app popup handling compatibility for all GUI frameworks).

Parameters

- **_args** – unused argument (to have compatible signature for DropDown/Popup/ModalView widgets).
- **_kwargs** – unused argument (to have compatible signature for DropDown/Popup/ModalView widgets).

on_complete_opened()

dispatch event default handler, called on opening when the final width got determined.

open(**_args*, ***kwargs*)

open container, optionally starting auto-width-animation.

Parameters

- **_args** – unused argument (to have compatible signature for Popup/ModalView and DropDown widgets passing the parent widget).
- **kwargs** – optional extra arguments:
  - 'animation': *False* will disable the *width* and open()-animations (default=True).

reset_width_detection()

call to reset the last detected minimum container width (e.g. if the children text got changed).

_detect_complete_width()

check clients textures until widest child texture got detected.

Return type
float

Returns
0.0 until complete width got detected, then the last detected minimum container width.

_on_complete_opened(*_args*)

open animation completion callback/event.

_on_win_width(*_args*)

Window.width event handler.

_open_width_progress(*_anim*, *_self*, *_progress*)

animation on_progress event handler.

_win_width_bind()

bind Window width property to container width.

class SimpleAutoTickerBehavior(**kwargs*)

Bases: object

mix-in class to slide texture in a ping-pong-like-ticker animation, if too long to be displayed completely.

if the *text* or *size* of the widget where this class get mixed in changes then this instance is first determining the number of characters that can be displayed completely in this widget. this is done with a kivy animation. the duration of this animation can be set via the property *auto_ticker_length_anim_duration*.

to adjust the padding space between the widget border and their texture width, the property *auto_ticker_text_spacing* can be set accordingly.

after determining the maximum number of characters that can be displayed and storing this value into the private attribute *_ticker_text_length* a second animation - the offset animation - gets started to slide/scroll the text. the speed of the offset animation can be set via the property *auto_ticker_offset_anim_speed*.

Note: while the ticker animations are running the *text* property of the widget is only containing the visible part of the full initial text string. use the private attribute *_ori_text* to determine the full text string.

---

bind: Callable

get_root_window: Callable

text: str

texture_size: tuple

texture_update: Callable

unbind: Callable

width: float

auto_ticker_length_anim_duration: float

duration in seconds of the iteration animation to determine the maximum text length.

auto_ticker_length_anim_duration is a NumericProperty and defaults to 0.9 seconds.

auto_ticker_offset_anim_speed: float

speed of the ticker text offset animation in characters per second.

auto_ticker_length_anim_duration is a NumericProperty and defaults to 9.6.

auto_ticker_text_spacing: float

horizontal padding between widget and texture width in pixels.

auto_ticker_text_spacing is a NumericProperty and defaults to 18sp.

_bound_properties: Dict[str, Callable]

properties of the mixing in widget to bind

_length_anim: Optional[Animation] = None

shorten length animation

_min_text_len: int = 6

minimal length of shortened text

_offset_anim: Optional[Animation] = None

ticker text offset animation

_ori_text: str = ''

original/full text string

_ticker_text_offset: int = 0

current animated offset in the ticker text

_ticker_text_length: int = 6

number of characters that are completely visible in widget

_ticker_text_updating: bool = False

flag to block restart of ticker on internal update of *text* property

__init__(***kwargs*)

_bind_properties()

_start_length_anim(**_args*)

_start_offset_anim(*offset_on_complete=0*)

_stop_length_anim(*reset=True*)

_stop_offset_anim(*reset=True*)

_text_changed(**_args*)

called on change of label text. assert _args[1] == self.text

_ticker_length_progress(*_anim*, *_self*, *progress*)

_ticker_max_offset()

Return type
int

_ticker_offset_progress(*_anim*, *_self*, *progress*)

_ticker_text_update(*text*)

_unbind_properties()

4.39 ae.kivy_file_chooser

4.39.1 extended kivy file chooser widget

This ae namespace portion provides the *FileChooserPopup* widget (*chooser_popup*) which is embedding Kivy's FileChooser class in a dropdown window (*FlowDropDown*), and extending it with a path selector and a button to switch between list and icon view.

file chooser dropdown usage

The *FileChooserPopup* widget can be used like any Kivy DropDown widget - see the python and kv lang examples in the doc strings of the dropdown module. Additionally all the features of the *FlowDropDown* like e.g. the *child_data_maps* are available.

Alternatively (and without the need to explicitly instantiate the file chooser dropdown widget) you simply have to change the application flow to *id_of_flow('open', 'file_chooser')* to open this file chooser (see also *application flow*):

```
main_app.change_flow(id_of_flow('open', 'file_chooser'),
                     **update_tap_kwargs(open_button))
```

The variable *open_button* in this example represents a button widget instance that opens the file chooser dropdown (and to which the file chooser gets attached to).

Use the *submit_to* property to distinguish multiple usages of the file chooser in a single app:

```
main_app.change_flow(id_of_flow('open', 'file_chooser'),
                     **update_tap_kwargs(open_button,
                                         popup_kwargs=dict(submit_to=submit_to_str_or_
↪callable))
```

The variable *submit_to_str_or_callable* of the above example can be either a string or a callable. If you pass a callable, *FileChooserPopup* will call it if the user has selected a file (by touching or double clicking on a file entry). This callback receives two arguments: the file path of just selected file and the FileChooser dropdown widget instance an and can be declared like:

```
def submit_to_callable(file_path: str, chooser_popup: Widget):
```

Passing a string to *submit_to* (or if it get not specified at all) the hard-coded *on_file_chooser_submit* event handler callback method of your main app instance will be executed with the same two arguments:

```
def on_file_chooser_submit(self, file_path: str, chooser_popup: Widget):
    if chooser_popup.submit_to == 'usage1':
        usage1_object_or_process.file_path = file_path
        chooser_popup.dismiss()
    elif chooser_popup.submit_to == 'usage2':
        ...
    elif chooser_popup.submit_to == '':     # w/o specifying `submit_to`
        ...
```

Use the key of the *tap_flow_id* property of the *FlowButton* to provide a separate i18n help text for each individual button.

The *filters* property of Kivy's kivy.uix.filechooser.FileChooser can be used to filter the files displayed in this file chooser widget.

The path selector dropdown (*FileChooserPathSelectPopup*) situated at the top of this file chooser dropdown is providing all common OS and app specific paths that are registered in the *PATH_PLACEHOLDERS* dict. The keys of this dict will be displayed as shortcut path names instead of the full path strings. Additionally translation texts can be provided for the shortcut path names to display them in the language selected by the app user.

To extend the path selector dropdown with additional paths you can either register them within *PATH_PLACEHOLDERS*, or you add them to the optional app state variable *file_chooser_paths* by calling the method *register_file_path()*.

By adding the list *file_chooser_paths* to the *app state variables* of your app, the paths provided by the path selector widget will automatically maintain and keep the OS and user paths persistent between app runs.

to record and remember the last selected path add also the app state *file_chooser_initial_path* to the *:ref:`app state variables* of your app.

Override the method *_init_default_user_cfg_vars()* within the main app instance of your app to make these two persistent *app state variables* user-specific:

```
def _init_default_user_cfg_vars(self):
    super()._init_default_user_cfg_vars()
    self.user_specific_cfg_vars |= {
        (APP_STATE_SECTION_NAME, 'file_chooser_initial_path'),
        (APP_STATE_SECTION_NAME, 'file_chooser_paths'),
    }
```

Hint: you don't need to override *_init_default_user_cfg_vars()* if your app is embedding the ae portion *ae.kivy_sideloading*.

Classes

FileChooserPathSelectPopup(**kwargs)	file chooser path selector dropdown.
FileChooserPopup(**kwargs)	file chooser drop down container.

class FileChooserPopup(***kwargs*)

Bases: *FlowDropDown*

file chooser drop down container.

initial_path

initial file path displayed on opening

submit_to

callable or string to identify which action/part requested the selected file

filters

see `kivy.uix.filechooser.FileChooser.filters`.

static on_file_chooser_entry_added(*view_entries*)

on_entry_added/on_subentry_to_entry event handler to patch theme-related properties of Kivy FileChooser.

Parameters

view_entries¶ (`List`[`Widget`]) – list of view entries for a node (icon or label) of the file chooser.

Note: This method get called for each node in the moment when a file entry widget (FileListEntry or FileIconEntry) gets added to an instance of Kivy's `FileChooser` widget class.

Therefore the patches done here are not affected if the user preferences (e.g. the font size or light/dark theme) get changed while a file chooser instance is displayed. In this case the user has to simply close and reopen/re-instantiate the file chooser to display the nodes with the just changed user preferences.

Theme adaption is still missing for the file chooser progress: all font sizes and colors of the currently used `FileChooserProgressBase` are hard-coded, so a theme-aware progress class has to be implemented (and assigned to the `progress_cls` property).

---

static register_file_path(*file_path*, *main_app*)

set folder of the passed file path as new initial path and add it to path history.

Parameters

- **file_path**¶ (`str`) – file path (mostly the last just selected file) of which the folder will be registered.
- **main_app**¶ (`Any`) – main app instance.

_container: Widget

_layout_finished: bool

_opened_item: Optional[Widget]

_touch_moved_outside: bool

class FileChooserPathSelectPopup(***kwargs*)

Bases: *FlowDropDown*

file chooser path selector dropdown.

paths

list of file paths in the path selection dropdown

_container: Widget

_layout_finished: bool

_opened_item: Optional[Widget]

_touch_moved_outside: bool

4.40 ae.kivy_iterable_displayer

4.40.1 iterable displayer widget

The popup widget provided by this ae namespace portion displays items and sub-items of any type of iterables, like dicts, lists, sets and tuples.

iterable displayer usage

To open a popup displaying the keys/indexes and values of an iterable simple instantiate *IterableDisplayerPopup*. You can specify a popup window title string via the *title* kwarg and pass the iterable to the *data* kwarg (or property):

```
dict_displayer = IterableDisplayerPopup(title="popup window title", data=iterable_data)
```

A widget will be automatically instantiated for each sub-item of *iterable_data* to display the item key and value. The used widget class is depending on the type of the sub-item. For non-iterable sub-items the *IterableDisplayerLabel* widget will be used. If instead a sub-item is containing another iterable then *IterableDisplayerPopup* will use the *IterableDisplayerButton* class, which when tapped displays another instance of *IterableDisplayerPopup* with the sub-sub-items.

Note: The string in the `title` property may be shortened automatically by *FlowPopup*, depending on the width of the popup layout and the *font_size* app state.

Classes

IterableDisplayerPopup(**kwargs)	FlowPopup displaying iterable data - useful for quick prototyping and debugging.

class IterableDisplayerPopup(***kwargs*)

Bases: *FlowPopup*

FlowPopup displaying iterable data - useful for quick prototyping and debugging.

data

the iterable (dict, list, set, tuple) from which the items will be shown

static compile_data_maps(*data*)

re-create data maps if the *data* attribute changes.

Parameters

data¶ (Union[dict, list, set, tuple]) – dict/list/set/tuple data to display (==self.data binding).

Returns

list of dicts to be assigned to self.child_data_maps.

_container: Widget

attach_to: Optional[Widget]

_layout_finished: bool

_opened_item: Optional[Widget]

_touch_moved_outside: bool

4.41 ae.kivy_qr_displayer

4.41.1 qr code displayer widget

the popup widget *QrDisplayerPopup* provided by this ae namespace portion is displaying QR codes.

the *QrDisplayerPopup* is inherited from *ae.kivy.widgets.FlowPopup* and is embedding the Kivy Garden `kivy_garden.qrcode` module.

qr displayer popup usage

to display a QR code instantiate *QrDisplayerPopup* specifying in the *title* property of the popup the string to encode to a QR image and in the *qr_content* property a short string describing the content of the string to encode. after that call the *open* method:

```
qr_displayer = QrDisplayerPopup(title="string to encode", qr_content="what to encode")
qr_displayer.open()
```

alternatively you can simply `change the application flow` to *id_of_flow('open', 'qr_displayer')* (see also *application flow*):

```
main_app.change_flow(id_of_flow('open', 'qr_displayer'),
                     popup_kwargs=dict(title="string to encode",
                                       qr_content="what to encode"))
```

the label texts used by this popup widget are automatically translated into the german and spanish language via the translation texts provided in the resources of this ae namespace portion.

Note: if your app is providing i18n translations then the *qr_content* string has to be translated (e.g. by using *get_txt()* or *get_text()*) before it gets passed to the popup kwargs.

---

to support additional languages simply add the translations texts to your app's translation texts resources or submit a PR to add them to this ae namespace portion. alternatively you could put different wording by specifying also the english translation text.

Hint: apart from *root.qr_content* you can also use *root.title* in the translation texts to repeat/mention the string to encode in the text content.

Classes

QrDisplayerPopup(**kwargs)	qr code displayer.

class QrDisplayerPopup(***kwargs*)

Bases: *FlowPopup*

qr code displayer.

qr_content

string to name the content that get displayed as QR code

_container: Widget

attach_to: Optional[Widget]

_layout_finished: bool

_opened_item: Optional[Widget]

_touch_moved_outside: bool

4.42 ae.kivy_sideloading

4.42.1 kivy mixin and widgets to integrate a sideloading server in your app

this namespace portion provides widgets and a mixin class for you main app instance to easily integrate and control the *ae sideloading server* <*ae.sideloading_server*> into your *main app*.

kivy sideloading integration into your main app class

add the *SideloadingMainAppMixin* mixin provided by this ae namespace portion to your main app class:

```
class MyMainAppClass(SideloadingMainAppMixin, KivyMainApp):
```

the sub app of the sideloading server will then automatically be instantiated when your app starts and will initialize the *sideloading_app* attribute with this sub app instance.

Hint: if you prefer to instantiate the sideloading server sub app manually then specify *SideloadingMainAppMixin* after *KivyMainApp* in the declaration of your main app class.

adding *sideloading_active* to the *:ref: `app state variables* of your app's *config files* will ensure that the running status of the sideloading server gets automatically stored persistent on pause or stop of the app for the next app start.

the running status of the sideloading server will be restored in the app start event handler method (*on_app_run()*).

to manually start it offering the APK of the embedding app call the *on_sideloading_server_start()* method passing an empty string and dict:

```
self.on_sideloading_server_start("", {})
```

Hint: when you pass the dict with a number in a 'port' key then this number will be used as server listening port.

if no 'port' gets specified then *SideloadingMainAppMixin* will calculate an individual port number from the first character of the *app_name* of the app mixing in this class. this is to prevent the server socket error *[Errno 98] Address already in use* if two different applications with sideloading are running on the same device and want to offer sideloading.

to manually pause the sideloading server call the *on_sideloading_server_stop()* method passing an empty string and dict:

```
self.on_sideloading_server_stop("", {})
```

usage of the sideloading button

this ae namespace portion is additionally providing the *SideloadingButton* flow button widget to integrate it in your Kivy app. This button can be used to:

- start or stop the sideloading server,
- select a file for sideloading via the *FileChooserPopup*.
- display file info like full file path and file length.
- display the URL of your sideloading server as QR code to allow connections from other devices.

to optionally integrate this *SideloadingButton* into your app add it to the root layout in your app's main kv file with the *id sideloading_button*:

```
MyRootLayout:
    ...
    SideloadingButton:
        id: sideloading_button
```

if the sideloading server is not active and the user is clicking the *SideloadingButton* then this portion will first check if the *Downloads* folder of the device is containing an APK file for the running app and if yes then the sideloading server will be started providing the found APK file.

ff the sideloading server is instead already running/active and the user is tapping on the *SideloadingButton* then a dropdown menu will be shown with options to (1) display info of the sideloading file, (2) select a new file, (3) display the sideloading server URL as QR code or (4) stop the sideloading server.

dependencies/requirements in *buildozer.spec*

to build an Android APK with the kivy sideloading server integrated, make sure that the following external packages are specified in the *requirements* setting of the *[app]* section of your *buildozer.spec* file.

- ae.kivy_file_chooser
- ae.kivy_iterable_displayer
- ae.kivy_qr_displayer
- ae.kivy_sideloading
- ae.sideloading_server
- kivy_garden.qrcode

- qrcode

additionally, the following packages and ae namespace portions required by the above packages have to be included:

```
qrcode, kivy_garden.qrcode,
ae.base, ae.files, ae.paths, ae.deep, ae.dynamicod, ae.i18n,
ae.updater, ae.core, ae.literal, ae.console, ae.parse_date, ae.gui_app,
ae.gui_help, ae.kivy_auto_width, ae.kivy_dyn_chi,
ae.kivy_relief_canvas, ae.kivy, ae.kivy_user_prefs, ae.kivy_glsl,
ae.kivy_file_chooser, ae.sideloading_server, ae.kivy_sideloading,
ae.kivy_iterable_displayer, ae.kivy_qr_displayer
```

sideloading server life cycle

to activate the sideloading server to offer a different file, specify the path (or glob file mask) of the file to be offered/available via sideloading in the *sideloading_file_mask* attribute and then call the method *on_sideloading_server_start()*. this method will check if the specified file exists and if yes then it will start the sideloading server. if you specify a file mask instead of a concrete file path then this method will check if exists exactly one file matching the file mask.

after the start of the sideloading server the *sideloading_file_ext* attribute will contain the file extension of the file available via sideloading.

the sideloading server will automatically be shut down on quit/close of the embedding app. you can alternatively stop the sideloading server manually at any time by calling the *on_sideloading_server_stop()* method.

Classes

SideloadingMainAppMixin()	mixin class with default methods for the main app class.
SideloadingMenuPopup(**kwargs)	dropdown menu to control sideloading server.
SideloadingMenuTour(main_app)	user preferences menu tour.

class SideloadingMenuPopup(***kwargs*)

Bases: *FlowDropDown*

dropdown menu to control sideloading server.

__init__(***kwargs*)

_container: Widget

_layout_finished: bool

_opened_item: Optional[Widget]

_touch_moved_outside: bool

class SideloadingMenuTour(*main_app*)

Bases: *TourDropdownFromButton*

user preferences menu tour.

__init__(*main_app*)

page_ids: List[str]

list of tour page ids, either initialized via this class attribute or dynamically.

_saved_app_states: Dict[str, Any]

auto_switch_pages: Union[bool, int]

enable/disable automatic switch of tour pages.

set to *True*, *1* or *-1* to automatically switch tour pages; *True* and *1* will switch to the next page until the last page is reached, while *-1* will switch back to the previous pages until the first page is reached; *-1* and *1* automatically toggles at the first/last page the to other value (endless ping-pong until back/next button gets pressed by the user).

the seconds to display each page before switching to the next one can be specified via the item value of the the dict `page_data` dict with the key *'next_page_delay'*.

page_data: Dict[str, Any]

additional/optional help variables (in *help_vars* key), tour and page text/layout/timing settings.

the class attribute values are default values for all tour pages and get individually overwritten for each tour page by the i18n translations attributes on tour page change via `load_page_data()`.

supported/implemented dict keys:

- *app_flow_delay*: time in seconds to wait until app flow change is completed (def=1.2, >0.9 for auto-width).
- *back_text*: caption of tour previous page button (def=get_text('back')).
- *fade_out_app*: set to 0.0 to prevent the fade out of the app screen (def=1.0).
- *help_vars*: additional help variables, e.g. *help_translation* providing context help translation dict/text.
- *next_text*: caption of tour next page button (def=get_text('next')).
- *next_page_delay*: time in seconds to read the current page before next request_auto_page_switch() (def=9.6).
- *page_update_delay*: time in seconds to wait until tour layout/overlay is completed (def=0.9).
- *tip_text* or '' (empty string): tour page tooltip text fstring message text template. alternatively put as first character a '=' character followed by a tour page flow id to initialize the tip_text to the help translation text of the related flow widget, and the *self* help variable to the related flow widget instance.
- *tour_start_delay*: seconds between tour.start() and on_tour_start main app event (def=TOUR_START_DELAY_DEF).
- *tour_exit_delay*: seconds between tour.stop() and the on_tour_exit main app event (def=TOUR_EXIT_DELAY_DEF).

pages_explained_matchers: Dict[str, Union[ExplainedMatcherType, Tuple[ExplainedMatcherType, ...]]]

matchers (specified as callable or id-string) to determine the explained widget(s) of each tour page.

each key of this dict is a tour page id (for which the explained widget(s) will be determined).

the value of each dict item is a matcher or a tuple of matchers. each matcher specifies a widget to be explained/targeted/highlighted. for matcher tuples the minimum rectangle enclosing all widgets get highlighted.

the types of matchers, to identify any visible widget, are:

- *find_widget()* matcher callable (scanning framework_win.children)
- evaluation expression resulting in *find_widget()* matcher callable
- widget id string, declared via kv lang, identifying widget in framework_root.ids
- page id string, compiled from widgets app state/flow/focus via `widget_page_id()` to identify widget

page_idx: int

index of the current tour page (in *page_ids*)

last_page_idx: Optional[int]

last tour page index (*None* on tour start)

class SideloadingMainAppMixin

Bases: object

mixin class with default methods for the main app class.

app_name: str

change_app_state: Callable

change_flow: Callable

dpo: Callable

framework_root: Widget

get_opt: Callable

show_message: Callable

user_specific_cfg_vars: set

vpo: Callable

file_chooser_initial_path: str = ''

used by `file_chooser` to select side-loaded file

file_chooser_paths: List[str] = []

recently used paths as app state for file chooser

sideloading_active: tuple = ()

app state flag if sideloading server is running

sideloading_app: *SideloadingServerApp*

http sideloading server console app

sideloading_file_ext: str = '.'

extension of selected sideloading file

sideloading_file_mask: str = ''

file mask of sideloading file

_init_default_user_cfg_vars()

on_app_run()

run app event.

on_app_started()

initialize and start shaders after kivy app, window and widget root got initialized.

on_debug_level_change(*level_name*, *_event_kwargs*)

debug level app state change flow change confirmation event handler.

Parameters

- **level_name** (str) – the new debug level name to be set (passed as flow key).
- **_event_kwargs** (Dict[str, Any]) – unused event kwargs.

Return type
bool

Returns
True to confirm the debug level change.

on_file_chooser_submit(*file_path*, *chooser_popup*)

event callback from FileChooserPopup.on_submit() on selection of file.

Parameters

- **file_path** (str) – path string of selected file.
- **chooser_popup** (Widget) – file chooser popup/container widget.

on_sideloading_server_start(*_flow_key*, *event_kwargs*)

start the sideloading server.

Parameters

- **_flow_key** (str) – unused/empty flow key.
- **event_kwargs** (Dict[str, Any]) – event kwargs: * 'port': TCP/IP server listening port. * 'tap_widget': button instance that initiated the start of the server.

Return type
bool

Returns
always True to confirm change of flow id.

on_sideloading_server_stop(*_flow_key*, *_event_kwargs*)

stop a running sideloading http server.

Return type
bool

Parameters

- **_flow_key** (str) – unused/empty flow key.
- **_event_kwargs** (Dict[str, Any]) – unused event kwargs.

:re change of flow id.

4.43 ae.kivy_user_prefs

4.43.1 user preferences widgets for your kivy app

This namespace portion is providing a set of widgets to allow the users of your app to change their personal app settings/preferences, like the theme, the font size, the language and the used colors.

To use it in your app import this module, which can be done either in one of the modules of your app via:

```
import ae.kivy_user_prefs
```

Alternatively and when you use the Kivy framework for your app, you can import it within your main KV file, like this:

```
#: import _any_dummy_name ae.kivy_user_prefs
```

Note: The i18n translation texts of the namespace portion get registered on importing. When you import this portion from the main KV file and your app is overwriting a translation text of this portion, then you have to make sure that the translation texts of your main app get registered after the import of this portion. For that reason `MainAppBase` is using the *on_app_build* event to load the application resources, which gets fired after Kivy has imported the main KV file.

The user preferences are implemented as a `FlowDropDown` via the widget *UserPreferencesPopup*.

To integrate it in your app you simply add the *UserPreferencesButton* widget to the main KV file of your app.

user preferences debug mode

The user preferences are activating a debug mode when you click/touch the *UserPreferencesButton* button more than three times within 6 seconds.

This debug mode activation is implemented in the `on_user_preferences_open()` event handler method declared in the `ae.kivy.apps` module. It can be disabled for your app by simply overriding this method with an empty method in your main app class.

4.44 ae.lisz_app_data

4.44.1 lisz demo app data handling

this module provides common constants, functions and methods for the showcase/demo application *Lisz*, which is demonstrating the usage of the GUI framework packages provided by the ae namespace.

usage demonstration of ae namespace portions

the usage of the following ae namespace portions get demonstrated by this application:

- `ae.base`: basic constants and helper functions
- `ae.files`: file collection, grouping and caching
- `ae.deep`: deep data structure search and replace
- `ae.i18n`: internationalization / localization helpers

- ae.paths: generic file path helpers
- ae.dynamicod: evaluation and execution helper functions
- ae.updater: application environment updater
- ae.core: application core constants, helper functions and base classes
- ae.literal: literal type detection and evaluation
- ae.console: console application environment
- ae.parse_date: parse date strings more flexible and less strict
- ae.gui_app: base class for python applications with a graphical user interface
- ae.gui_help: main app base class with context help for flow and app state changes

Hint: the Kivy variant of this demo app uses additionally the following ae namespace portions: ae.kivy_auto_width, ae.kivy_dyn_chi, ae.kivy_relief_canvas, ae.kivy and ae.kivy_user_prefs.

features of the lisz demo app

- internationalization of texts, user messages, help texts, button/label texts (ae.i18n)
- easy mapping of files in complex folder structures (ae.files, ae.paths)
- providing help screens (ae.gui_help, ae.kivy.widgets)
- colors changeable by user (ae.kivy_user_prefs, ae.enaml_app)
- font and button sizes are changeable by user (ae.gui_app)
- dark and light theme switchable by user
- sound output support with sound volume configurable by user
- recursive item data tree manipulation: add, edit and delete item
- each item can be selected/check marked
- filtering of selected/checked and unselected/unchecked items
- an item represents either a sub-node (sub-list) or a leaf of the data tree
- item order changeable via drag & drop
- item can be moved to the parent or a sub-node
- easy navigation within the item tree (up/down navigation in tree and quick jump)

lisz application data model

fhe lisz demo app is managing a recursive item tree - a list of lists - that can be used e.g. as to-do or shopping list.

to keep this demo app simple, the data managed by the lisz application is a minimalistic tree structure that gets stored as an *application status*, without the need of any database. the root node and with that the whole recursive data structure gets stored in the app state variable *root_node*.

the root of the tree structure is a list of the type *LiszNode* containing list items of type *LiszItem*. a *LiszItem* element represents a dict of the type *Dict[str, Any]*.

each *LiszItem* element of the tree structure is either a leaf or a node. and each node is a sub-list with a recursive structure identical to the root node and of the type *LiszNode*.

the following graph is showing an example data tree:

the above example tree structure is containing the root node items *A* (which is a leaf) and *C* (which is a sub-node).

the node *C* consists of the items *CA* and *CB* where *CA* is a leaf and *CB* is a node.

the first item of the node *CB* is another sub-node with the leaf item *CBA*.

4.44.2 GUI framework demo implementations

the plan is to integrate the following GUI frameworks on top of the `abstract base class` (implemented in the *ae.gui_app* portion of the ae namespace):

- *Kivy* based on the Kivy framework: kivy lisz demo app
- *Enaml* based on the enaml framework: enaml lisz demo app
- `Beeware Toga` based on the beeware framework: beeware toga lisz demo app
- `Dabo` based on the dabo framework: dabo lisz demo app
- `pyglet`
- `pygobject`
- `AppJar`

the main app base mixin class *LiszDataMixin* provided by this module is used to manage the common data structures, functions and methods for the various demo applications variants based on *ae.gui_app* and the related GUI framework implementation portions (like e.g. *ae.kivy.apps* and *ae.enaml_app*) of the ae namespace.

4.44.3 gui framework implementation variants

kivy

the kivy lisz app is based on the Kivy framework, a pypi package documented here.

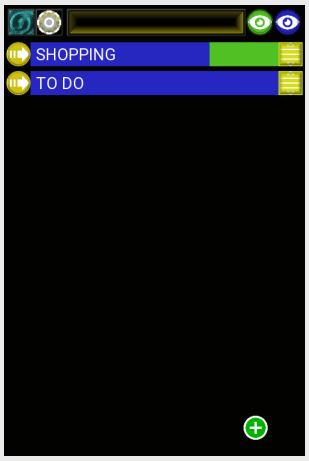

Fig. 4.1: kivy lisz app root list

Fig. 4.2: fruits sub-list

Fig. 4.3: using light theme

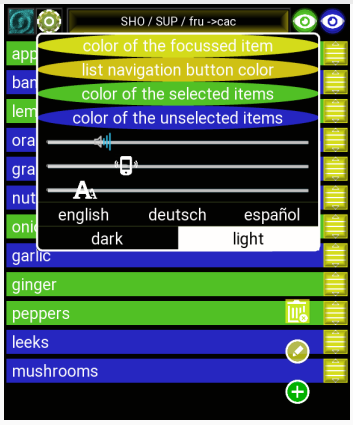

Fig. 4.4: lisz user preferences

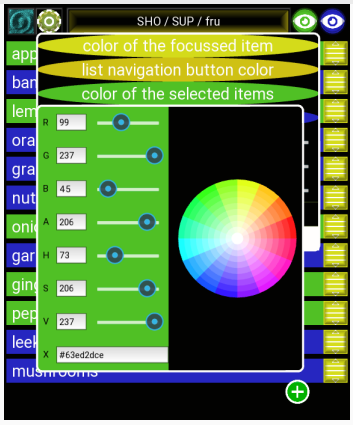

Fig. 4.5: kivy lisz color editor

Fig. 4.6: bigger font size

kivy wish list

- kv language looper pseudo widget (like enaml is providing) to easily generate sets of similar widgets.

enaml

the enaml lisz demo app is based on the enaml framework, a pypi package documented here at ReadTheDocs.

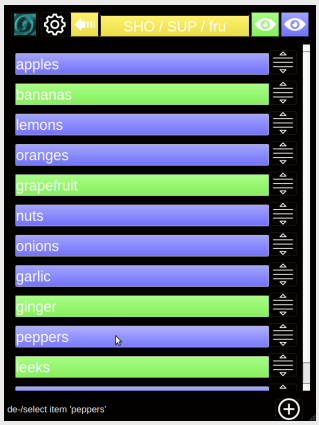

Fig. 4.7: enaml/qt lisz app

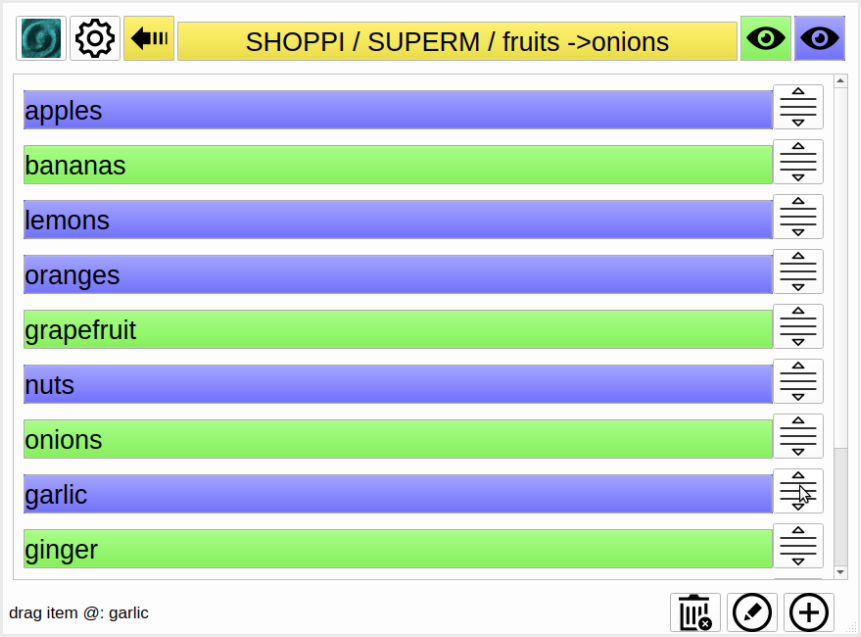

Fig. 4.8: light themed in landscape

automatic update of widget attributes

dependencies have to be executed/read_from, so e.g. the icon attribute will not be updated if app.app_state_light_theme gets changed:

```
icon << main_app.cached_icon('font_size') or app.app_state_light_theme
```

in contrary the icon will be updated by the following two statements:

```
icon << main_app.cached_icon('font_size') if app.app_state_light_theme else main_app.
↪cached_icon('font_size')
icon << app.app_state_light_theme == None or main_app.cached_icon('font_size')
```

KeyEvent implementation based on this SO answer posted by the enamlx author frmdstryr/Jairus Martin: https://stackoverflow.com/questions/20380940/how-to-get-key-events-when-using-enaml. alternative and more complete implementation can be found in the enamlx package (https://github.com/frmdstryr/enamlx).

enaml wish list

- type and syntax checking, code highlighting and debugging of enaml files within PyCharm.
- fix freezing of linux/Ubuntu system in debugging of opening/opened PopupViews in PyCharm.

Module Attributes

FLOW_PATH_ROOT_ID	pseudo item id needed for flow path jumper and for drop onto leave item button
FLOW_PATH_TEXT_SEP	flow path separator for *flow_path_text()*
FOCUS_FLOW_PREFIX	prefix shown in front of flow key of focused item
INVALID_ITEM_ID_PREFIX_CHARS	invalid initial chars in item id (to detect id | literal in flow key)
NODE_FILE_PREFIX	file name prefix for node imports/exports
NODE_FILE_EXT	file extension for node imports/exports
IMPORT_NODE_MAX_FILE_LEN	maximum file length of importable <file.NODE_FILE_EXT> file
IMPORT_NODE_MAX_ITEMS	maximum number of items to import or paste from clipboard
LiszItem	node item data (nid) type
LiszNode	node/list type
NodeFileInfo	tuple of (node-parent-name, node-list, file_path, error-message)
NodeFilesInfo	list of node file info tuples

Functions

check_item_id(item_id)	check if the passed item id string is valid.
correct_item_id(item_id)	strip and replace extra/invalid characters from the passed item id string.
flow_path_items_from_text(text)	parse and interpret text block for (optional) flow path text and node data (in pprint.pformat/repr format).
item_sel_filter(item)	callable to filter selected LiszItems.
item_unsel_filter(item)	callable to filter unselected LiszItems.

Classes

LiszDataMixin()	lisz data model - independent from used GUI framework.

FLOW_PATH_ROOT_ID = '^^^^'

pseudo item id needed for flow path jumper and for drop onto leave item button

FLOW_PATH_TEXT_SEP = ' / '

flow path separator for *flow_path_text()*

FOCUS_FLOW_PREFIX = '->'

prefix shown in front of flow key of focused item

INVALID_ITEM_ID_PREFIX_CHARS = '[{'

invalid initial chars in item id (to detect id | literal in flow key)

NODE_FILE_PREFIX = 'node_'

file name prefix for node imports/exports

NODE_FILE_EXT = '.txt'

file extension for node imports/exports

IMPORT_NODE_MAX_FILE_LEN = 8192

maximum file length of importable <file.NODE_FILE_EXT> file

IMPORT_NODE_MAX_ITEMS = 12

maximum number of items to import or paste from clipboard

LiszItem

node item data (nid) type

alias of `Dict[str, Any]`

LiszNode

node/list type

alias of `List[Dict[str, Any]]`

NodeFileInfo

tuple of (node-parent-name, node-list, file_path, error-message)

alias of `Tuple[str, List[Dict[str, Any]], str, str]`

NodeFilesInfo

list of node file info tuples

alias of `List[Tuple[str, List[Dict[str, Any]], str, str]]`

check_item_id(*item_id*)

check if the passed item id string is valid.

Parameters
item_id (str) – item id to check.

Return type
str

Returns
"" if all chars in the specified `item_id` argument are valid, else one of the translated message strings.

correct_item_id(*item_id*)

strip and replace extra/invalid characters from the passed item id string.

Parameters
item_id (str) – item id string to correct.

Return type
str

Returns
corrected item id (can result in an empty string).

flow_path_items_from_text(*text*)

parse and interpret text block for (optional) flow path text and node data (in pprint.pformat/repr format).

Parameters
text (str) – text block to be parsed. the text block can optionally be prefixed with an extra line (separated by a new line 'n' character) containing the destination flow path in text format (using FLOW_PATH_TEXT_SEP to separate the flow path items).

the (rest of the) text block represents the node/item data in one of the following formats:

- single text line (interpreted as single leaf item).
- multiple text lines (interpreted as multiple leaf items).
- dict repr string, starting with '{' character.
- list repr string, starting with '[' character.

Return type
Tuple[str, str, List[Dict[str, Any]]]

Returns
tuple of error message (empty string if no error occurred), flow path (empty string if root or not given) and node list.

item_sel_filter(*item*)

callable to filter selected LiszItems.

Parameters
item (Dict[str, Any]) – item data structure to check.

Return type
bool

Returns
True if item is a selected leaf or if item is a node with only selected sub-leaves, else False.

item_unsel_filter(*item*)

callable to filter unselected LiszItems.

Parameters
item (Dict[str, Any]) – item data structure to check.

Return type
bool

Returns
True if item is an unselected leaf or if item is a node with only unselected leaves, else False.

class LiszDataMixin

Bases: object

lisz data model - independent from used GUI framework.

root_node: List[Dict[str, Any]] = []

root of lisz data structure

current_node_items: List[Dict[str, Any]]

node item data of the current node / sub list (stored as app state via root_node)

filtered_indexes: List[int]

indexes of the filtered/displayed items in the current node

filter_selected: bool = False

True to hide/filter selected node items

filter_unselected: bool = False

True to hide/filter unselected node items

debug_level: int

debug_level

flow_id: str

current attr:*flow id <ae.gui_app.MainAppBase.flow_id>*

flow_path: List[str]

flow path ref. current node

_refreshing_data: bool = False

DEBUG True while running *refresh_all()* method

call_method: Callable

call_method_delayed: Callable

change_app_state: Callable

change_flow: Callable

flow_path_action: Callable

play_sound: Callable

refresh_node_widgets: Callable

add_item(*nid*, *node_to_add_to=None*, *new_item_index=0*)

add item (leaf or node) to currently displayed node.

Parameters

- **nid** (Dict[str, Any]) – LiszItem to add (has to have a non-empty item id).
- **node_to_add_to** (Optional[List[Dict[str, Any]]]) – node where the passed item will be added to (def=current node).
- **new_item_index** (int) – index where the new item will be inserted (default=0, ignored if item already exists).

Return type
str

Returns
error message if any error happened, else empty string.

add_items(*items*, *node_to_add_to=None*)

add item to currently displayed node.

Parameters

- **items** (List[Dict[str, Any]]) – LiszNode list to add (each item has to have a non-empty item id).
- **node_to_add_to** (Optional[List[Dict[str, Any]]]) – node where the passed item will be added to (def=current node).

Return type
str

Returns
error message if any error happened (multiple error messages are separated by \n), else empty string.

`change_sub_node_sel`(*node*, *set_sel_to*)

change the selection of all the sub-leaves of the passed node to the specified value.

Parameters

- **node** (List[Dict[str, Any]]) – node of which to change the selection of the sub-item leaves.
- **set_sel_to** (bool) – True will only toggle the unselected sub-item leaves, False only the selected ones.

`current_item_or_node_literal`()

return the currently focused/displayed item or node as repr string.

Return type
str

Returns
pformat repr string of the currently focused item id/node or of the displayed node.

`delete_items`(**item_ids*, *parent_node=None*, *node_only=False*)

delete either complete items or sub node of the items (identified by the passed item ids).

Parameters

- **item_ids** (str) – tuple of item ids to identify the items/sub-nodes to be deleted.
- **parent_node** (Optional[List[Dict[str, Any]]]) – node from where the item has to be removed from (default=current node).
- **node_only** (bool) – True if only delete the sub-node of the identified item, False to delete the item.

Return type
List[Dict[str, Any]]

`edit_validate`(*old_item_index*, *new_id=None*, *want_node=None*, *parent_node=None*, *new_item_index=0*)

validate the user changes after adding a new item or editing an existing item.

Parameters

- **old_item_index** (int) – index in the current node of the edited item or -1 if a new item (to be added).
- **new_id** (Optional[str]) – new/edited id string.
- **want_node** (Optional[bool]) – True if the new/edited item will have a sub-node, False if not.
- **parent_node** (Optional[List[Dict[str, Any]]]) – node where the edited/added item as to be updated/inserted (default=current list).
- **new_item_index** (int) – index where the new item have to be inserted (default=0, ignored in edit item mode).

Return type
str

Returns
empty string on successful edit validation or on cancellation of new item (with empty id string), else error string or *'request_delete_confirmation_for_item'* if the user has to confirm the deletion after the user wiped the item id string or *'request_delete_confirmation_for_node'* if the user has to confirm the removal of the sub-node.

export_node(*flow_path*, *file_path='.'*, *node=None*)

export node specified by the passed *flow_path* argument.

Parameters

- **flow_path** (List[str]) – flow path of the node to export.
- **file_path** (str) – folder to store the node data into (def=current working directory).
- **node** (Optional[List[Dict[str, Any]]]) – explicit/filtered node items (if not passed then all items will be exported).

Return type
str

Returns
empty string if node got exported without errors, else the error message/raised exception.

find_item_index(*item_id*, *searched_node=None*)

determine list index of the passed item id in the searched node or in the current node.

Parameters

- **item_id** (str) – item id to find.
- **searched_node** (Optional[List[Dict[str, Any]]]) – searched node. if not passed then the current node will be searched instead.

Return type
int

Returns
item list index in the searched node or -1 if item id was not found.

flow_key_text(*flow_id*, *landscape*)

determine the shortest possible text fragment of the passed flow key that is unique in the current node.

used to display unique part of the key of the focused item/node.

Parameters

- **flow_id** (str) – flow id to get key to check from (pass the observed value to update GUI automatically, either self.app_state_flow_id or self.app_states['flow_id']).
- **landscape** (bool) – True if window has landscape shape (resulting in larger abbreviation). pass the observed attribute, mostly situated in the framework_win (e.g. self.framework_win.landscape).

Return type
str

Returns
display text containing flow key.

flow_path_from_text(*text*, *skip_check=False*)

restore the full complete flow path from the shortened flow keys generated by *flow_path_text()*.

Parameters

- **text** (str) – flow path text - like returned by *flow_path_text()*.
- **skip_check** (bool) – pass True to skip the check if the flow path exists in the current self.root_node.

Return type
List[str]

Returns
flow path list.

flow_path_node(*flow_path=None*, *create=False*)

determine the node specified by the passed flow path, optionally create missing nodes of the flow path.

Parameters

- **flow_path** (Optional[List[str]]) – flow path list.
- **create** (bool) – pass True to create missing nodes. only on False this method will return empty list on invalid/broken flow_path.

Return type
List[Dict[str, Any]]

Returns
node list at flow_path (if found -or- created is True and no-creation-errors) or empty list (if flow_path not found and created is False or on creation error).

flow_path_quick_jump_nodes()

determine the current flow paths of all nodes excluding the current, to quick-jump from current node.

Return type
List[str]

Returns
list of flow path texts of all nodes apart from the current one.

flow_path_text(*flow_path*, *min_len=3*, *display_root=False*, *separator='/'*)

generate shortened display text from the passed flow path.

Parameters

- **flow_path** (List[str]) – flow path list.
- **min_len** (int) – minimum length of node ids (flow id keys). pass zero value to not shorten ids.
- **display_root** (bool) – pass True to return FLOW_PATH_ROOT_ID on empty/root path.
- **separator** (str) – path item separator (default=FLOW_PATH_TEXT_SEP).

Return type
str

Returns
shortened display text string of the passed flow path (which can be converted back to a flow path list with the method *flow_path_from_text()*).

focus_neighbour_item(*delta*)

move flow id to previous/next displayed/filtered node item.

Parameters
delta¶ (int) – moving step (if greater 0 then forward, else backward).

global_variables(***patches*)

overridden to add app-specific globals.

Return type
Dict[str, Any]

importable_node_files(*folder_path='.'*)

load and check all nodes found in the app folder documents of this app.

Parameters
folder_path¶ (str) – path to the folder where the node files are situated (def=current working directory).

Return type
List[Tuple[str, List[Dict[str, Any]], str, str]]

Returns
list of node file tuples of (node_name, node, file_path, error-message).

static import_file_info(*file_path*, *node_name=''*)

load node file and determine node content.

Parameters

- **file_path**¶ (str) – path to the node file to import.
- **node_name**¶ (str) – optional name/id of the parent node name. if not passed then it will be determined from the file name (removing *NODE_FILE_PREFIX* and file extension).

Return type
Tuple[str, List[Dict[str, Any]], str, str]

Returns
node file info tuple as: (node name, node-data or empty list, file path, error message).

import_items(*node*, *parent=None*, *item_index=0*)

import passed node items into the passed parent/destination node at the given index.

Parameters

- **node**¶ (List[Dict[str, Any]]) – node with items to import/add.
- **parent**¶ (Optional[List[Dict[str, Any]]]) – destination node to add the node items to (def=current node list).
- **item_index**¶ (int) – list index in the destination node where the items have to be inserted (default=0).

Return type
str

Returns
empty string if all items of node got imported correctly, else error message string.

import_node(*node_id*, *node*, *parent=None*, *item_index=0*)

import passed node as new node into the passed parent node at the given index.

Parameters

- **node_id** (str) – id of the new node to import/add.
- **node** (List[Dict[str, Any]]) – node with items to import/add.
- **parent** (Optional[List[Dict[str, Any]]]) – destination node to add the new node to (def=current node list).
- **item_index** (int) – list index in the parent node where the items have to be inserted (default=0).

Return type
str

Returns
empty string if node got added/imported correctly, else error message string.

item_by_id(*item_id*, *searched_node=None*)

search item in either the passed or the current node.

Parameters

- **item_id** (str) – item id to find.
- **searched_node** (Optional[List[Dict[str, Any]]]) – searched node. if not passed then the current node will be searched instead.

Return type
Dict[str, Any]

Returns
found item or if not found a new dict with the single key=value 'id'=item_id.

move_item(*dragged_node*, *dragged_id*, *dropped_path=None*, *dropped_id=''*)

move item id from passed dragged_node to the node and index specified by dropped_path and dropped_id.

Parameters

- **dragged_node** (List[Dict[str, Any]]) – node where the item got dragged/moved from.
- **dragged_id** (str) – id of the dragged/moved item.
- **dropped_path** (Optional[List[str]]) – optional destination/drop node path, if not passed use dragged_node.
- **dropped_id** (str) – optional destination item where the dragged item will be moved before it. if empty string passed or not passed then the item will be placed at the end of the destination node.

Return type
bool

node_info(*node*, *what=()*, *recursive=True*)

determine statistics info for the node specified by *flow_path*.

Parameters

- **node** (List[Dict[str, Any]]) – node to get info for.

- **what** (Tuple[str, ...]) – pass tuple of statistic info fields to include only these into the returned dict (passing an empty tuple or nothing will include all the following fields):
  - 'count': number of items (nodes and leaves) in this node (including sub-nodes).
  - 'leaf_count': number of sub-leaves.
  - 'node_count': number of sub-nodes.
  - 'selected_leaf_count': number of selected sub-leaves.
  - 'unselected_leaf_count': number of unselected sub-leaves.
  - 'names': list of all sub-item/-node names/ids.
  - 'leaf_names': list of all sub-leaf names.
  - 'selected_leaf_names': list of all selected sub-leaf names.
  - 'unselected_leaf_names': list of all unselected sub-leaf names.
- **recursive** (bool) – pass False if only the passed node has to be investigated.

Return type
Dict[str, Union[int, str, List[str]]]

Returns
dict with the node info specified by the *what* argument.

on_app_state_root_node_save(*root_node*)

shrink root_node app state variable before it get saved to the config file.

Return type
List[Dict[str, Any]]

on_filter_toggle(*toggle_attr*, *_event_kwargs*)

toggle filter on click of either the selected or the unselected filter button.

note that the inverted filter may be toggled to prevent both filters active.

Parameters
- **toggle_attr** (str) – specifying the filter button to toggle, either 'filter_selected' or 'filter_unselected'.
- **_event_kwargs** (Dict[str, Any]) – unused.

Return type
bool

Returns
True to process flow id change.

on_item_enter(*_key*, *event_kwargs*)

entering sub node from current node.

Parameters
- **_key** (str) – flow key (item id).
- **event_kwargs** (dict) – event kwargs.

Return type
bool

Returns
True to process/change flow id.

`on_item_leave`(*_key*, *event_kwargs*)

leaving sub node, setting current node to parent node.

Parameters

- **_key** (`str`) – flow key (item id).
- **event_kwargs** (`dict`) – event kwargs.

Return type
`bool`

Returns
True to process/change flow id.

`on_item_sel_toggle`(*item_id*, *event_kwargs*)

toggle selection of leaf item.

Parameters

- **item_id** (`str`) – item id of the leaf to toggle selection for.
- **event_kwargs** (`dict`) – event kwargs.

Return type
`bool`

Returns
True to process/change flow id.

`on_item_sel_change`(*item_id*, *event_kwargs*)

toggle, set or reset in the current node the selection of a leaf item or of the sub-leaves of a node item.

Parameters

- **item_id** (`str`) – item id of the leaf/node to toggle selection for.
- **event_kwargs** (`dict`) – event kwargs, containing a *set_sel_to* key with a boolean value, where True will select and False deselect the item (or the sub-items if the item is a non-empty node).

Return type
`bool`

Returns
True to process/change flow id.

this flow change event can be used alternatively to *on_item_sel_toggle()* for more sophisticated lisz app implementations, like e.g. the kivy lisz demo app .

`on_item_sel_confirming`(*item_id*, *event_kwargs*)

confirming sub-list item de-/selection from ItemSelConfirmPopup

Return type
`bool`

`on_key_press`(*modifiers*, *key_code*)

check key press event to be handled and processed as command/action.

Parameters

- **modifiers** (`str`) – modifier keys string.

- **key_code** (str) – key code string.

Return type
bool

Returns
True if key event got processed/used, else False.

on_node_extract(*flow_path_text*, *event_kwargs*)

extract the leaves of the node specified by *flow_path_text*.

Parameters

- **flow_path_text** (str) – flow path text or list literal (identifying the start node to extract from).
- **event_kwargs** (dict) – extra arguments specifying extract options (only *extract_type* is mandatory):

  extract_type specifies extract destination and an optional filter on un-/selected items. the first part defines the extract action (copy/cut/delete/export/share) and an optional second part (separated by an underscore) the filter. e.g. the following string values can be passed for a 'copy' extract action:

  – 'copy' is copying all items of the specified node to the clipboard.

  – 'copy_sel' is copying only the selected items of the node to the clipboard.

  – 'copy_unsel' is copying only the unselected items to the clipboard.

  recursive specifies if *False* that the extraction affects only the leaves of the current node specified by *flow_path_text* and if *True* the extraction affects also the leaves of the sub-nodes (default=True).

  export_path specifies the destination folder of the export action (default='.'/CWD).

Return type
bool

Returns
True to process/change flow.

on_node_jump(*flow_path_text*, *event_kwargs*)

FlowButton clicked event handler restoring flow path from the flow key.

Parameters

- **flow_path_text** (str) – flow path text (identifying where to jump to).
- **event_kwargs** (Dict[str, Any]) – event arguments (used to reset flow id).

Return type
bool

Returns
always True to process/change flow.

refresh_all()

changed flow event handler refreshing currently displayed items after changing current node/flow path.

refresh_current_node_items_from_flow_path()

refresh current node including the depending on display node.

shrink_node_size(*node*, *item_filter=None*, *recursive=True*)

shrink node size by removing unneeded items and *sel* keys, e.g. to export or save space in config file.

Parameters

- **node** (List[Dict[str, Any]]) – start or root node to shrink (in-place!).
- **item_filter** (Optional[Callable[[Dict[str, Any]], bool]]) – pass callable to remove items from the passed node and its sub-nodes. the callable is getting each item as argument and has to return True to remove it from its node.
- **recursive** (bool) – pass False if only the passed start node has to be shrunk.

sub_item_ids(*node=None*, *item_ids=()*, *leaves_only=True*, *hide_sel_val=None*, *recursive=True*, *sub_ids=None*)

return item names/ids of the specified items including their sub-node items (if exists and recursive==True).

used to determine the affected item ids if user want to delete or de-/select the sub-items of the item(s) specified by the passed arguments.

Parameters

- **node** (Optional[List[Dict[str, Any]]]) – searched node, if not passed use the current node as default.
- **item_ids** (Tuple[str, ...]) – optional item id filter, if passed only items with an id in this tuple will be returned. this filter will not be used for sub-node filtering (if recursive==True).
- **leaves_only** (bool) – pass False to also include/return node item ids.
- **hide_sel_val** (Optional[bool]) – pass False/True to exclude un-/selected leaf items from the returned list of ids. if None or not passed then all found items will be included.
- **recursive** (bool) – pass False if only the passed start node has to be investigated/included.
- **sub_ids** (Optional[List[str]]) – already found sub item ids (used for the recursive calls of this method).

Return type
List[str]

Returns
list of found item ids.

toggle_item_sel(*node_idx*)

toggle the item selection of the item identified by the list index in the current node.

Parameters
node_idx (int) – list index of the item in the current node to change the selection for.

4.45 ae.enaml_app

4.45.1 enaml application widgets, helper functions and classes

the enaml module *widgets* is providing widgets to write themed applications which can switch their font colors and backgrounds at run-time between dark and light.

another set of widgets provided by this namespace portion allows the automatic change of the application flow with only few lines of code.

to convert colors between the enaml and other formats the functions declared in the `functions` of this package can be used.

main application class for GUIApp-conform Enaml app

the classes *FrameworkApp* and *EnamlMainApp* of this ae portion are bundling and adding useful attributes and methods for your application and are extendable by creating a subclass.

the class *EnamlMainApp* is implementing a main app class that is reducing the amount of code needed to create a Python application based on the enaml framework.

EnamlMainApp is based on the following classes:

- the abstract base class *MainAppBase* which is providing *application status* (including *app state variables* and *app state constants*), *application flow* and *application events*.
- the class *ConsoleApp* is adding *config files*, *config variables* and *config options*.
- the class *AppBase* is adding *application logging* and *application debugging*.

the main app class *EnamlMainApp* is also encapsulating the enaml app class for the Qt widget set (<enaml.QtApplication>) within the *FrameworkApp* class.

an instance of the Enaml app class can be directly accessed from the main app class instance via the *framework_app* attribute.

enaml application events

this portion is firing *application events* additional to the ones provided by *MainAppBase*. these framework app events get fired after *on_app_run()* in the following order:

- on_app_build (fired on start of the application event loop).
- on_app_stopped (fired after the main application window got closed)

Functions

convert_key_event_to_code(event)	converts the Qt key-press/-release event into a modifiers and key code string.

Classes

EnamlMainApp(**console_app_kwargs)	enaml application main base class
FrameworkApp(*args, **kwargs)	enaml framework application class with atom member/attribute support.

convert_key_event_to_code(*event*)

converts the Qt key-press/-release event into a modifiers and key code string.

Return type
Tuple[str, str]

class FrameworkApp(**args*, ***kwargs*)

Bases: QtApplication

enaml framework application class with atom member/attribute support.

app_state_flow_id

A value of type *str*.

By default, bytes will NOT be promoted to strings. Pass strict=False to the constructor to enable loose string checking.

app_state_flow_path

A List member which supports container notifications.

app_state_font_size

A value of type *float*.

By default, ints and longs will be promoted to floats. Pass strict=True to the constructor to enable strict float checking.

app_state_light_theme

A value of type *bool*.

app_state_sound_volume

A value of type *float*.

By default, ints and longs will be promoted to floats. Pass strict=True to the constructor to enable strict float checking.

app_state_win_rectangle

A member which allows tuple values.

If item validation is used, then assignment will create a copy of the original tuple before validating the items, since validation may change the item values.

app_state_flow_id_ink

A member which allows list values.

Assigning to a list creates a copy. The orginal list will remain unmodified. This is similar to the semantics of the assignment operator on the C++ STL container classes.

app_state_flow_path_ink

A member which allows list values.

Assigning to a list creates a copy. The orginal list will remain unmodified. This is similar to the semantics of the assignment operator on the C++ STL container classes.

app_state_selected_item_ink

A member which allows list values.

Assigning to a list creates a copy. The orginal list will remain unmodified. This is similar to the semantics of the assignment operator on the C++ STL container classes.

app_state_unselected_item_ink

A member which allows list values.

Assigning to a list creates a copy. The orginal list will remain unmodified. This is similar to the semantics of the assignment operator on the C++ STL container classes.

landscape

saved via win_rectangle app state

max_font_size

maximum font size in pixels bound to window size

min_font_size

minimum - " -

mixed_back_ink

mixed color - used for user pref dropdown opening button

__atom_specific_members__ = frozenset({'app_state_flow_id', 'app_state_flow_id_ink', 'app_state_flow_path', 'app_state_flow_path_ink', 'app_state_font_size', 'app_state_light_theme', 'app_state_selected_item_ink', 'app_state_sound_volume', 'app_state_unselected_item_ink', 'app_state_win_rectangle', 'landscape', 'max_font_size', 'min_font_size', 'mixed_back_ink'})

__slotnames__ = []

class EnamlMainApp(***console_app_kwargs*)

Bases: *MainAppBase*

enaml application main base class

_original_key_press_handler: Optional[Callable] = None

_original_key_release_handler: Optional[Callable] = None

_original_win_resize_handler: Optional[Callable] = None

init_app(*framework_app_class=<class 'ae.enaml_app.FrameworkApp'>*)

initialize framework app instance and root window/layout, return GUI event loop start/stop methods.

Return type
Tuple[Optional[Callable], Optional[Callable]]

cached_icon(*icon_name*, *size*, *light*)

get cached image/icon object.

Return type
Optional[Icon]

call_method_delayed(*_delay*, *callback*, **args*, ***kwargs*)

delay not implemented - for now redirect to direct call.

Return type
Any

focus_widget(*widget*)

set input/keyboard focus to the passed widget.

Parameters
widget¶ (Widget) – widget/window that will receive the focus.

focused_widget()

enaml/qt focus debug helper method determining tool tip of the current qt widget with focus

Return type
str

Returns
tool tip string of current focus or app window status.

key_press_from_enaml(*event*)

convert/normalize enaml/Qt key press/down event and pass it to MainAppBase key press dispatcher.

key_release_from_enaml(*event*)

convert/normalize enaml/Qt key release/up event and pass it to MainAppBase key release dispatcher.

load_images()

overwrite un-cached image file register to use cached image files instead.

on_app_run()

run/start app event handler.

on_flow_widget_focused()

set focus to the widget referenced by the current flow id.

on_font_size_change(*_flow_key*, *event_kwargs*)

font size app state flow change confirmation event handler.

Parameters

- **_flow_key**¶ (str) – flow key.
- **event_kwargs**¶ (Dict[str, Any]) – event kwargs with key *'font_size'* containing the font_size in pixels.

Return type
bool

Returns
True to confirm change of flow id.

open_popup(*popup_class*, ***popup_kwargs*)

open Popup and set focus to the first widget.

Parameters

- **popup_class**¶ (Type) – class of the Popup widget/window.
- **popup_kwargs**¶ – args to instantiate and show/open the popup.

Return type
Widget

Returns
instance of the popup widget.

play_sound(*sound_name*)

play audio/sound file.

user_preference_color_selected(*color_name*, *dialog*)

ColorDialog callback.

win_activated(*main_window*)

main window activated event handler, called only once on app startup via widgets.enaml/ThemeMainWindow.

Parameters
main_window¶ (Any) –

Returns

win_closed(*changed*)

callback fired on close of Main window to save/restore framework_win.geometry on app exit/start.

Parameters
changed¶ (dict) – qt/enaml changed event dict.

Note: neither self.framework_app.stop() nor self.framework_app._ qapp.exit(exit_code) trigger window closed event.

win_resize_from_enaml(*event*)

convert/normalize enaml/Qt key press/down event and pass it to MainAppBase key press dispatcher.

4.46 ae.enaml_app.functions

enaml application helper functions.

Functions

ae_rgba(color)	convert enaml Color instance into ae color format.
enaml_rgba(red, green, blue, alpha)	convert ae color rgba floats into enaml compatible rgba integers.
style_rgba(red, green, blue, alpha)	convert ae color rgba floats into Qt style compatible rgba color string.

ae_rgba(*color*)

convert enaml Color instance into ae color format.

Parameters
color¶ (Color) – enaml Color instance.

Return type
Tuple[float, float, float, float]

Returns
rgba tuple with 4 float values (0.0 … 1.0).

enaml_rgba(*red*, *green*, *blue*, *alpha*)

convert ae color rgba floats into enaml compatible rgba integers.

Parameters

- **red**¶ (float) – red color value (0.0 … 1.0).
- **green**¶ (float) – green color value (0.0 … 1.0).
- **blue**¶ (float) – blue color value (0.0 … 1.0).
- **alpha**¶ (float) – alpha/opacity value (0.0 … 1.0).

Return type
Color

Returns
rgba enaml Color instance (0 … 255).

style_rgba(*red*, *green*, *blue*, *alpha*)

convert ae color rgba floats into Qt style compatible rgba color string.

Parameters

- **red**¶ (float) – red color value (0.0 … 1.0).
- **green**¶ (float) – green color value (0.0 … 1.0).
- **blue**¶ (float) – blue color value (0.0 … 1.0).
- **alpha**¶ (float) – alpha/opacity value (0.0 … 1.0).

Return type
str

Returns
rgba enaml Color instance (0 … 255).

CHAPTER

FIVE

INDICES AND TABLES

- portion repositories at gitlab.com
- genindex
- modindex
- `ae` namespace projects and documentation
- `aedev` namespace projects and documentation

PYTHON MODULE INDEX

a

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X