IPM e2e

Release 0.0.1

David Cabrero Souto

Mar 08, 2022

CONTENTS:

Release 0.0.1

Date Mar 08, 2022

This library helps in writting of automated *end to end* tests when a graphical user interface is involved.

To achive its goal, it offers a functional api that performs programmatically the usual interactions with the graphical interface, on behalf of a human user.

Right now it only works with applications that implement the AT-SPI.

Maybe you'd like to think about this library as a higher level abstraction of the AT-SPI api.

CHAPTER

ONE

IPM E2E

This library implements the usual functions that we'll need to write *end to end* tests.

It offers a functional api that performs programmatically the usual interactions with the graphical interface, on behalf of a human user.

In order to do its job, this library uses the at-spi api, so the corresponding service must be available and the applications under test must implement this api.

1.1 Features

- High level, interaction-oriented api

1.2 Installation

```
pip install ipm_e2e
```

1.2.1 Dependencies (no python)

This library depends on several services and libraries, mainly c code, that cannot be installed using pip:

- AT-SPI service
- GObject introspection libraries
- Assistive Technology Service Provider Interface - shared library
- Assistive Technology Service Provider (GObject introspection)

You should use your system's package manager to install them. The installation process depends on your system, by example, for a debian distro:

```
$ sudo apt install at-spi2-core gir1.2-atspi-2.0
```

Note that, if you're using Gnome, some of these packages are already installed.

1.2.2 Dependencies (python)

This library depends on the following python library:

- Python 3 bindings for gobject-introspection libraries

That `python3-gi` library itself depends on some libraries like `gir1.2-glib-2.0`, `gir1.2-atspi-2.0`, ... If you've installed them using your system's package manager, the safe bet would be to do the same for this one. By example:

```
$ sudo apt install python3-gi
```

1.2.3 Dependencies (virtual environment)

If you're using a virtual environment, probably you'll prefer not to manually install/compile the non-python libraries, neither use the `system-site-packages` option. Instead of that, it's easier to install `vext`:

```
$ pip install vext vext.gi
```

, or `pygobject`:

```
$ pip install pygobject
```

1.3 Documentation

The documentation is available at readthedocs.

1.4 Support

Please open an issue for support.

1.5 License

The project is licensed under the LGPL license.

CHAPTER

TWO

MOTIVATION

AT-SPI provides an api that we can use to write automated *end to end* tests that interact with the *graphical user interface*. That's great, but this api is a low-level api, that can be really cumbersome. Let's take, as an example, the following user history:

```
GIVEN I started the application
WHEN I click the button "Contar"
THEN I see the label "Has pulsado 1 vez"
```

When we implement that user history as a test using the bare at-spi, we can get something like:

```
# GIVEN I started the application

## Run it as a new OS process
path = "./contador.py"
name = f"{path}-test-{str(random.randint(0, 100000000))}"
process = subprocess.Popen([path, '--name', name])
assert process is not None, f"No pude ejecuar la aplicación {path}"

## Wait until at-spi founds it in the desktop
## Include a timeout
desktop = Atspi.get_desktop(0)
start = time.time()
timeout = 5
app = None
while app is None and (time.time() - start) < timeout:
    gen = filter(lambda child: child and child.get_name() == name,
                 (desktop.get_child_at_index(i) for i in range(desktop.get_child_
↪count())))
    app = next(gen, None)
    if app is None:
        time.sleep(0.6)

## Check everything went ok
if app is None:
    process and process.kill()
    assert False, f"La aplicación {path} no aparece en el escritorio"


# WHEN I click the button "Contar"

## Search the button
```

(continues on next page)

(continued from previous page)

```
for obj in tree_walk(app):
    if (obj.get_role_name() == 'push button' and
        obj.get_name() == 'Contar'):
        break
else:
    assert False, "No pude encontrar el botón 'Contar'"

## Search the action 'click'
for idx in range(obj.get_n_actions()):
    if obj.get_action_name(idx) == 'click':
        break
else:
    assert False, "El botón 'Contar' no tiene una acción 'click'"

## Perform the action
obj.do_action(idx)


# THEN I see the label "Has pulsado 1 vez"

## Search the label
for obj in tree_walk(app):
    if (obj.get_role_name() == 'label' and
        obj.get_text(0, -1).startswith("Has pulsado")):
        break
else:
    assert False, "No pude encontrar la etiqueta 'Has pulsado ...'"

## Check the text
assert obj.get_text(0, -1) == "Has pulsado 1 vez"


# Clean-up the mess & finish
process and process.kill()
```

As seen, using this api implies writing a lot of repetitive and cumbersome code only to search for objects, getting their attributes, performing actions on them, ...

After writing two or three tests, we'll probably find ourselves writing a library to reuse this common code. And this is how the idea of writing this library come in place the first time. But instead of sticking around with just abstracting the repeated code, I wanted the library to provide a more high-level api closer to the *gherkin language* than to the *c api*.

After implementing the same example using the library, we can get the following code:

```
# GIVEN I started the application
process, app = e2e.run("./contador.py")
## Check everything went ok
if app is None:
    process and process.kill()
    assert False, f"La aplicación {path} no aparece en el escritorio"
do, shows = e2e.perform_on(app)

# WHEN I click the button 'Contar'
```

(continues on next page)

(continued from previous page)

```
do('click', role= 'push button', name= 'Contar')

# THEN I see the label "Has pulsado 1 vez"
assert shows(role= "label", text= "Has pulsado 1 vez")

# Clean-up the mess & finish
process and process.kill()
```

CHAPTER

THREE

E2E MODULE API

Functions to perform interactions with graphical user interfaces on behalf of regular users.

This library offers a functional api to perform programmatically common interactions with the graphical interface. In order to do its job, this library uses the at-spi api, so the corresponding service must be available and the applications must implement the api.

If you rather like it, think of this library as an abstraction of the at-spi api. This abstraction is intended to ease the use of the api.

Examples

Implementation of Gherkin steps:

```
# GIVEN I started the application
process, app = e2e.run("./contador.py")
## ok ?
if app is None:
    process and process.kill()
    assert False, f"There is no aplication {path} in the desktop"
do, shows = e2e.perform_on(app)

# WHEN I click the button 'Contar'
do('click', role= 'push button', name= 'Contar')

# THEN I see the text "Has pulsado 1 vez"
assert shows(role= "label", text= "Has pulsado 1 vez")

## Before leaving, clean up your mess
process and process.kill()
```

ipm.e2e.**Either**

The classic Either type, i.e. a value of type T or an error.

N.B.: The typing and implementation of the functions using this type is quite relaxed (unorthodox).

alias of Union[Exception, ipm.e2e.T]

ipm.e2e.**MatchArgs**

A dict containing a list of patterns an at-spi object should match.

An object matches the list of patterns if and only if it matches all the patterns in the list.

Each pair `name: value` is interpreted as follows:

name = 'when': Predicate on the object.

The value is a function `Atspi.Object -> bool`. This function returns whether or not the at-spi objects matches this pattern.

name = 'nth': Position of the object.

The value must be the position of the at-spi object among its siblings. As usual the positions start at 0, and negative indexes are refered to the end of the list.

name = 'path': TODO

name = '...': Otherwise, one of the object's atrributes.

The name is interpreted as the name of one object's attribute, and the value is interpreted as follows:

value = str | bytes: String or bytes.

The value must equal to the value of the object's attribute.

value = re.Pattern: A regular expression.

The object's attribute value must match the given re.

value = function(Any -> bool): A predicate on the attribute's value.

The function must return True when called with the object's attribute value as argument.

alias of `Dict[str, Union[AnyStr, re.Pattern, Callable[[gi.repository.Atspi.Object], bool], Callable[[Any], bool]]]`

`ipm.e2e.fail_on_error`(*x: Union[Exception, Any]*) → Any

Raises an exception on error.

Raiese an exception when the python object represents an error, otherwise returns the object itself.

Parameters **x** (*Either[T]*) – The object to check

Returns The python object when it is not an error

Return type T

Raises **Exception** – The exception that corresponds to the error

`ipm.e2e.find_all_objs`(*roots: Union[gi.repository.Atspi.Object, Iterable[gi.repository.Atspi.Object]]*, ***kwargs: Dict[str, Union[AnyStr, re.Pattern, Callable[[gi.repository.Atspi.Object], bool], Callable[[Any], bool]]]*) → list

Searchs for all the at-spi objects that matches the arguments.

This function is similar to *find_obj*. The meaning of the *kwargs* arguments is the same. But it presents the following differences:

- Instead of a root object to start the search from, it's possible to specify a collection of objects to start from every one of them.
- Instead of returning the first object that matches the arguments, it returns a list containing all of them.

Parameters

- **roots** (*Union[Atspi.Object, Iterable[Atspi.Object]]*) – The object/s to start the search from
- ****kwargs** – See *MatchArgs*

Returns The list of all descendants that matches the arguments

Return type list[Atspi.Object]

ipm.e2e.**find_obj**(*root: gi.repository.Atspi.Object*, ***kwargs: Dict[str, Union[AnyStr, re.Pattern, Callable[[gi.repository.Atspi.Object], bool], Callable[[Any], bool]]]*) → Union[Exception, gi.repository.Atspi.Object]

Searchs for the first at-spi object that matches the arguments.

This functions searchs the given *root* object and its descendants (inorder), looking for the first object that matches the patterns in *kwargs*.

When *kwargs* is empty, the *root* object is selected.

Parameters

- **root** (*Atspi.Object*) – The object to start the search from
- ****kwargs** – See *MatchArgs*

Returns

- *Atspi.Object* – The first descendant (inorder) that matches the arguments
- *NotFoundError* – When no object matches the arguments

ipm.e2e.**is_error**(*x: Union[Exception, Any]*) → bool

Checks whether any python object represents an error.

This function is intended to be used with values of type `Either`.

Parameters **x** (*Either[T]*) – The object to check

Returns whether it's an error

Return type bool

ipm.e2e.**obj_children**(*obj: gi.repository.Atspi.Object*) → list

Obtains the list of children of an at-spi object.

Parameters **obj** (*Atspi.Object*) – The object whose children will be queried.

Returns The list of object's children.

Return type list[Atspi.Object]

ipm.e2e.**obj_get_attr**(*obj: gi.repository.Atspi.Object*, *name: str*) → Union[Exception, str]

Returns the value of an at-spi object's attribute.

Some attributes are not actual attributes in at-spi, and must be retrieved using an at-spi function like *get_text()*. This function can chooice the right way to access attributes.

Parameters

- **obj** (*Atspi.Object*) – The object from which to retrieve the value of the attriute
- **name** (*str*) – The name of the attribute

Returns

- *str* – The value of the attribute
- *AttributeError* – When the object has no such attribute

ipm.e2e.**perform_on**(*root: gi.repository.Atspi.Object*, ***kwargs: Dict[str, Union[AnyStr, re.Pattern, Callable[[gi.repository.Atspi.Object], bool], Callable[[Any], bool]]]*) → tuple

Constructs functions that interact with one part of the user interface.

For the given at-spi object, this function finds the first descendant that matches the patterns in *kwargs*. Then, it selects the part of the UI given by the subtree which root is the found object and constructs the following functions that performs actions on that subtree:

ipm.e2e.**do**(*action_name: str*, ***kwargs:* MatchArgs) → None
Performs an action on the first descendant at-spi object that matches the patterns.

Parameters

- **action_name** (*str*) – The name of the action
- ****kwargs** – See *MatchArgs*

Raises

- **NotFoundError** – if no object mathces the patterns in kwargs
- **NotFoundError** – if the matching object doesn't provide the give action

ipm.e2e.**shows**(***kwargs:* MatchArgs) → bool
Checks whether the UI shows the information given by the patterns.

Note that the pattern should specify both the information to be shown and the at-spi object that contains that information.

Parameters ****kwargs** – See *MatchArgs*

Returns Whether any object matches the patterns

Return type bool

These functions interact, using at-spi, with the subtree rooted at the at-spi object found. They implement two basic interactions, intended to replace the users' interaction.

Parameters

- **root** (*Atspi.Object*) – The at-spi object to start the search from
- ****kwargs** – See *MatchArgs*

Returns A tuple with the two functions: *do()* and *shows()*

Return type UIInteraction

Raises **NotFoundError** – If no object matches the patterns in *kwargs*

ipm.e2e.**perform_on_each**(*roots: Iterable[gi.repository.Atspi.Object]*, ***kwargs: Dict[str, Union[AnyStr, re.Pattern, Callable[[gi.repository.Atspi.Object], bool], Callable[[Any], bool]]]*) → tuple
Constructs functions that interact with some parts of the user interface.

This function is similar to *perform_on()*, but instead of selecting one part (subtree) of the UI, it selects a collection of them.

For each object in *roots*, it selects a subtree that starts at the first descendant that matches the patterns in *kwargs*. Then it constructs the following functions that performs actions on that collections of subtrees:

ipm.e2e.**foreach_do**(*action_name: str*, ***kwargs:* MatchArgs) → None
For each subtree performs an action on the first descendant that matches the patterns.

Parameters

- **action_name** (*str*) – The name of the action
- ****kwargs** – See *MatchArgs*

Raises

- **NotFoundError** – If no object matches the patterns in kwargs
- **NotFountError** – If the matching object doesn't provide the given action

ipm.e2e.**each_shows**(***kwargs:* MatchArgs) → Iterator[bool]

For each subtree checks wheter the UI shows the information given by the patterns.

Note that the pattern should specify both the information to be shown and the at-spi object that contains that information.

Note that the result is an iterator which data can be aggregated using the builtins `any` or `all`.

Parameters ****kwargs** – See *MatchArgs*

Returns A collection of booleans indicating whether any object matches the patterns for each root object

Return type Iterator[bool]

These functions will perform the interaction on every subtree. As in *perform_on()*, they implement, using at-spi, two basic interactions, intended to replace the users' interaction.

Parameters

- **roots** (*Union[Atspi.Object, Iterable[Atspi.Object]]*) – The object/s to start the search from
- ****kwargs** – See *MatchArgs*

Returns A tuple with the two functions: *foreach_do()* and *each_shows()*

Return type UIMultipleInteraction

Raises **NotFoundError** – If no object matches the patterns in *kwargs*

ipm.e2e.**run**(*path: Union[str, pathlib.Path]*, *name: Optional[str] = None*, *timeout: Optional[float] = None*) → tuple

Runs the command in a new os process. Waits for application to appear in desktop.

Starts a new os process and runs the given command in it. The command should start an application that implements the at-spi and provides a user interface.

After running the command, the function will wait until the corresponding application appears in the desktop and it is accessible through the at-spi.

Finally it will return the Popen object that controls the process and the at-spi object that controls the accessible application.

When, after a given timeout, it cannot find the application, it will stop waiting and return None instead of the at-spi object.

Parameters

- **path** (*str | pathlib.Path*) – The file path of the command
- **name** (*str, optional*) – The application's name that will be shown in the desktop. When no name is given, the function will forge one.
- **path** – str | pathlib.Path The file path of the command

Returns Popen object that is in charge of running the command in a new process. The Atspi object that represents the application in the desktop, or None if it couldn't find the application after a given timeout.

Return type (subprocess.Popen, Atspi.Object)

ipm.e2e.**tree_walk**(*root: gi.repository.Atspi.Object*, *path: tuple = (NthOf(i=0, n=1),)*) → Iterator[tuple]

Creates a tree traversal.

This function performs an inorder tree traversal, starting at the given _root_ (at-spi object). Foreach visited node, it yields the path from the root to the node, and the node itself.

The path includes the position and the number of siblings of each node.

Parameters

- **root** (*Atspi.Object*) – The root node where to start the traversal.
- **path** (*TreePath, optional*) – A prefix for the paths yielded.

Yields *(TreePath, Atspi.Object)* – A tuple containing the path to the node and the node itself

CHAPTER

FOUR

CLI COMMANDS

4.1 atspi-dump

Usage:

```
$ atspi-dump
```

, prints the names of the applications running in the desktop

```
$ atspi-dump <name>
```

, prints the the tree of at-spi objects of the application `name`

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX