

# ansys-aedt-toolkits-common

ANSYS, Inc.
Southpointe
2600 Ansys Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
http://www.ansys.com
(T) 724-746-3304
(F) 724-514-9494

Feb 16, 2026

# CONTENTS

**Useful links**: *Installation* | Source repository | Issues

The PyADT Common Toolkit provides common methods and best practices for creating an Ansys Electronics Desktop (AEDT) toolkit.

Getting started Learn how to install the PyADT Common Toolkit, understand its architecture and use this toolkits common method to developer an example AEDT toolkit.

*Getting started* Backend API reference Understand how to use the backend API.

*Backend API reference* UI API reference Understand how to use the UI API.

*UI API reference* Examples Explore examples that show how to use the Common AEDT API.

*Examples* Contribute Learn how to contribute to the PyAEDT Common Toolkit codebase or documentation.

*Contribute* How to distribute Learn how to package your toolkit and its dependencies into a distributable format for the end user.

*Distribute*

CHAPTER

ONE

# GETTING STARTED

This section explains how to install the PyAEDT Common Toolkit. It then explains this toolkits architecture and provides an example of how to use its common methods to develop a new Ansys Electronics Desktop (AEDT) toolkit.

Installation Learn how to install the PyAEDT Common Toolkit.

*Installation* Architecture Learn about this toolkits architecture.

*Architecture* Example toolkit Use this toolkits common methods to create an example toolkit.

*Toolkit example*

## 1.1 Installation

The PyAEDT Common Toolkit can be installed like any other open source package and then added as a dependency to a new toolkit project.

From PyPI, you can either install both the backend and user interface (UI) methods or install only the backend methods.

To install both the backend and UI methods, run this command:

```
pip install ansys-aedt-toolkits-common[all]
```

If you only need the common API, install only the backend methods with this command:

```
pip install ansys-aedt-toolkits-common
```

## 1.2 Architecture

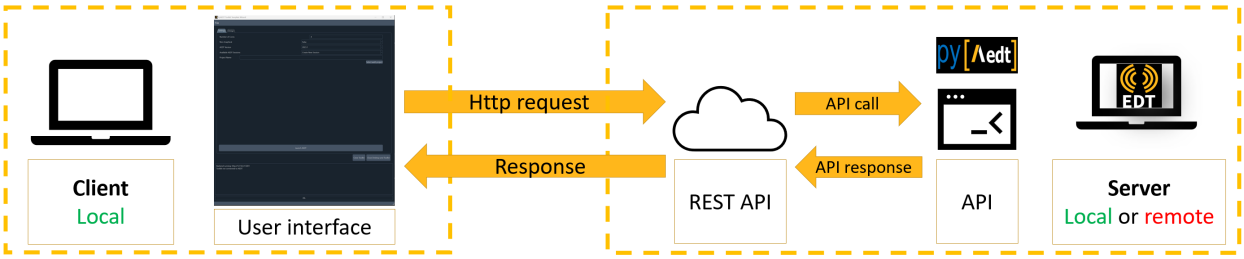
The AEDT Common Toolkit provides a common API for creating new AEDT toolkits. Thereby standardizing their implementations.

The API provides methods for connecting to an existing AEDT session, opening an existing AEDT project, and initializing a new AEDT session, which are basic capabilities required by any toolkit.

The architecture is divided in two main parts:

- **Backend**: Consists of the API and REST API. The API is built on PyAEDT. The REST API uses Flask to facilitate the creation of a REST API that enables interactions between different services through HTTP requests. By leveraging Flask, the toolkit becomes interface-agnostic, allowing flexibility in choosing different user interfaces such as a Web UI.
- **User Interface**: Provides UI creation capability using Pyside6. Pyside6 includes a designer tool for creating UIs translated directly to Python.

### 1.2.1 Toolkit architecture diagram




### 1.2.2 Toolkit backend and UI

The repository for the PyAEDT Common Toolkit is structured as follows:

```
pyaedt-toolkits-common
 .github
   workflows
       ci_cd.yml
 doc
   source
       examples
       getting_started
       toolkit
 src.ansys.aedt.toolkits
    common
        backend
           api.py
           rest_api.py
           common_properties.toml
           models.py
        ui
           common_windows
           main_window
           utils
           common_properties.toml
           models.py
 tests
    backend
 pyproject.toml
 README.rst
```

- .github: GitHub Action configuration.
- doc: Documentation structure.
- common: Toolkit code, split into backend and UI.
  - **backend:**
    Non-user-facing part of the toolkit for handling requests and preparing data for the UI. Key files include:
    - `rest_api.py`: Defines Flask entrypoints.
    - `api.py`: Defines the toolkit API.
    - `common_properties.toml`: Defines common backend properties.
    - `models.py`: Defines the class for storing backend properties.

  - ui: UI part of the toolkit. Key files include:
    - `common_properties.toml`: Defines common UI properties.
    - `models.py`: Defines the class for storing UI properties.
- tests: Folder containing the backend unit tests.

### 1.2.3 Models and properties

The models.py file stores the backend properties that are shared between the backend and UI. Properties are loaded by loading the content of the properties in the class properties.

To understand how the backend and UI interact, see the actions_generic.py file in the repository. For example, when an event is triggered by the frontend, the get_properties() method builds the GET HTTP request to send to the backend to retrieve properties from the backend. The event of setting up a property calls the set_properties() method, which builds the PUT HTTP request that is sent to the backend.

### 1.2.4 API

The *Backend API reference* contains three classes, `Common`, `AEDTCommon`, and `EDBCommon`, which provide methods for controlling the toolkit workflow.

### 1.2.5 REST API

REST APIs are standard web interfaces allowing clients to communicate with services via HTTP requests. JSON is the standard for transferring data. In fact, REST APIs accept JSON for request payload and also send responses to JSON.

In the client-server architecture model, the client sends the request to the server to fetch some information. Server-side technologies decode JSON information and transmit back the response to the client. This interaction is handled by the HTTP protocol.

### 1.2.6 UI and backend interaction

The UI sends HTTP requests to retrieve data, while the backend returns appropriate results.

These operations are tied to a timeout (set to 30 seconds by default) to prevent hanging requests, which could be exploited to cause denial of service (DoS) conditions (for more information, see here). Users can adjust this timeout (expressed in seconds) by setting the environment variable `PYAEDT_TOOLKIT_REQUESTS_TIMEOUT` and assigning it a value higher than the default, for example for a Windows user:

```
set PYAEDT_TOOLKIT_REQUESTS_TIMEOUT=60
```

Setting a longer timeout may be necessary if certain actions triggered via the UI (such as connecting to AEDT) fail to complete within the default time limit. This can for instance manifest through errors in validating or updating toolkit properties, or a stuck progress bar.

The toolkit uses CRUD (Create, Read, Update & Delete) operations that are simply HTTP request methods that specify the action to perform through the request.

### 1.2.7 UI

For more information on the UI, see *UI API reference*.

## 1.3 Toolkit example

The examples/toolkit/pyaedt_toolkit folder contains all files for creating a toolkit using the PyAEDT Common Toolkit.

### 1.3.1 Example walkthrough

Follow the steps outlined in the example to gain practical insights into toolkit implementation:

1. **Access the example**: Navigate to the examples/toolkit/pyaedt_toolkit folder.
2. **Understand the toolkit structure**:
    - Explore the directory and file structure of the example toolkit.
    - Gain insights into best practices for organizing toolkit components.

### 1.3.2 Toolkit structure

For optimal organization and maintainability, toolkits should adhere to the following structure:

```
pyaedt-toolkits-example
 backend
    api.py
    models.py
    backend_properties.json
    run_backend.py

 ui
    run_frontend.py
    actions.py
    frontend_properties.json
    windows
    models.json

 run_toolkit.py
```

### 1.3.3 Backend and UI

As described in *Architecture*, toolkits must have a separation between the backend and UI.

- The `backend` directory houses backend functionalities, including API and REST API definitions, data processing, and communication with the common library.
- The `ui` folder focuses on frontend interactions, managing the UI and connecting with backend functionalities.

### 1.3.4 API

The `toolkit` API controls the workflow, enabling the creation of an automated workflow without a UI.

```
ToolkitBackend(AEDTCommon)
```

The following code shows how to inherit common methods. You can play with the API in the Python console:

```
# Import API
from examples.toolkit.pyaedt_toolkit.backend.api import ToolkitBackend

# Object with generic methods to control the toolkits
```

(continues on next page)

(continued from previous page)

```
toolkit_api = ToolkitBackend()

# Launch AEDT, using common API method
toolkit_api.launch_aedt()
toolkit_api.wait_to_be_idle()

# Create geometry, using specific API method
toolkit_api.create_geometry()
```

### 1.3.5 Models and properties

To introduce new properties to the toolkit, define them using models. Properties have a fixed type, so they are protected. In models, specify the type. In this example, two new properties, `multiplier` and `geometry`, are defined as float and string, respectively.

In the `backend_properties.json` file, define default values for both common and new properties. These properties are correctly loaded by being imported into the toolkit API, as seen here:

```
from models import properties
```

### 1.3.6 Run backend

A script, conventionally named `rest_api.py` for its role in managing the REST API of the toolkit, is referred to as `run_backend.py` in this example. Upon execution, the script launches a server that listens for incoming requests.

Similar to the API, this file inherits the common REST API, containing only the specific REST API functionalities required for the toolkit. The following Python code imports the REST API application from the common library:

```
from ansys.aedt.toolkits.common.backend.rest_api import app
```

This code then creates an instance of the toolkit API object:

```
toolkit_api = ToolkitBackend()
```

### 1.3.7 Run frontend

The `run_frontend.py` script serves as the application launcher for the UI, built using PySide6. The file concludes with the following code, ensuring proper initialization using PySide6:

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = ApplicationWindow()
    window.show()
    sys.exit(app.exec())
```

The initialization of the `ApplicationWindow` class calls different common pages defined in *UI reference*.

If additional pages are to be added to the toolkit, include them along with any required actions inside the `windows` directory.

### 1.3.8 Common actions

Common actions define the calls to the REST API, as described in *Actions*.

### 1.3.9 UI properties

Similar to the backend, the UI has its own properties. The `frontend_properties.json` file enables customization of the UI theme, addition of new tabs, and modification of the URL and port for backend communication.

### 1.3.10 Run toolkit

The `run_toolkit.py` script facilitates the simultaneous execution of both the backend and UI in two different threads. This eliminates the need for launching the backend and UI separately. In cases where the backend is running remotely, execute the backend on the remote machine before running this script.

CHAPTER

TWO

# BACKEND API REFERENCE

The backend API contains three classes, *AEDTCommon* `, EDBCommon, and Common, which provide methods for controlling the toolkit workflow:

- AEDTCommon: Provides methods for controlling AEDT. This class inherits the Common class.
- EDBCommon: Provides methods for controlling EDB. This class inherits the Common class.
- Common: Provides methods for controlling the toolkit flow.

In the following descriptions, you can click the class name to view detailed API information.

| | |
|---|---|
| *AEDTCommon*([backend_properties]) | Provides common functions for controlling AEDT. |
| *EDBCommon*([backend_properties]) | Provides the generic API for controlling EDB. |
| *Common*([backend_properties]) | Provides the API for controlling the toolkits. |

## 2.1 ansys.aedt.toolkits.common.backend.api.AEDTCommon

**class** ansys.aedt.toolkits.common.backend.api.**AEDTCommon**(*backend_properties: ThreadManager | None = None*)

Provides common functions for controlling AEDT.

This class provides basic functions for controlling AEDT and properties to share between the backend and UI.

**Parameters**

**backend_properties**
[backend.models.Properties] Updated properties.

### Examples

```
>>> from ansys.aedt.toolkits.common.backend.api import AEDTCommon
>>> toolkit_api = AEDTCommon()
>>> msg = toolkit_api.launch_aedt()
```

**__init__**(*backend_properties: ThreadManager | None = None*)

### Methods

| | |
|---|---|
| *__init__*([backend_properties]) | |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| aedt_sessions() | Get information for the active AEDT sessions. |
| connect_aedt() | Connect to an existing AEDT session. |
| connect_design([app_name]) | Connect to an application design. |
| export_aedt_model([obj_list, export_path, ...]) | Export the model in the OBJ format and then encode the file if the encode parameter is enabled. |
| get_design_names() | Get the design names for a specific project. |
| get_project_name(project_path) | Get the project name from the project path. |
| get_properties() | Get the toolkit properties. |
| get_thread_status() | Get the toolkit thread status. |
| installed_aedt_version() | Get the installed AEDT versions. |
| is_aedt_connected() | Check if AEDT is connected. |
| launch_aedt() | Launch AEDT. |
| launch_thread(process, *args) | Launch the thread. |
| open_project([project_name]) | Open an AEDT project. |
| release_aedt([close_projects, close_on_exit]) | Release AEDT. |
| save_project([project_path, release_aedt]) | Save the project. |
| serialize_obj_base64(file_path) | Encode a bytes-like object. |
| set_properties(data) | Assign the passed data to the internal data model. |
| wait_to_be_idle([timeout]) | Wait for the thread to be idle and ready to accept a new task. |

## 2.2 ansys.aedt.toolkits.common.backend.api.EDBCommon

**class** ansys.aedt.toolkits.common.backend.api.**EDBCommon**(*backend_properties=None*)

Provides the generic API for controlling EDB.

This class provides basic functions to control EDB and properties to share between the backend and UI.

**Parameters**

**backend_properties**
[backend.models.Properties] Updated properties.

### Examples

```
>>> from ansys.aedt.toolkits.common.backend.api import EDBCommon
>>> toolkit_api = EDBCommon()
>>> toolkit_api.load_edb("path/to/file")
```

**__init__**(*backend_properties=None*)

### Methods

| | |
|---|---|
| *__init__*([backend_properties]) | |
| aedt_sessions() | Get information for the active AEDT sessions. |
| close_edb() | Close the EDB project. |
| get_properties() | Get the toolkit properties. |
| get_thread_status() | Get the toolkit thread status. |
| installed_aedt_version() | Get the installed AEDT versions. |
| launch_thread(process, *args) | Launch the thread. |

continues on next page

Table 3 – continued from previous page

| | |
|---|---|
| load_edb([edb_path]) | Load the EDB project. |
| save_edb([edb_path]) | Save the EDB project. |
| serialize_obj_base64(file_path) | Encode a bytes-like object. |
| set_properties(data) | Assign the passed data to the internal data model. |
| wait_to_be_idle([timeout]) | Wait for the thread to be idle and ready to accept a new task. |

## 2.3 ansys.aedt.toolkits.common.backend.api.Common

**class** ansys.aedt.toolkits.common.backend.api.**Common**(*backend_properties=None*)

Provides the API for controlling the toolkits.

This class provides basic functions to control AEDT and EDB and the properties to share between the backend and UI.

**Parameters**

**backend_properties**
[backend.models.Properties] Updated properties.

### Examples

```
>>> from ansys.aedt.toolkits.common.backend.api import Common
>>> toolkit_api = Common()
>>> toolkit_properties = toolkit_api.get_properties()
>>> new_properties = {"aedt_version": "2025.2"}
>>> toolkit_api.set_properties(new_properties)
>>> new_properties = toolkit_api.get_properties()
```

**__init__**(*backend_properties=None*)

### Methods

| | |
|---|---|
| *__init__*([backend_properties]) | |
| aedt_sessions() | Get information for the active AEDT sessions. |
| get_properties() | Get the toolkit properties. |
| get_thread_status() | Get the toolkit thread status. |
| installed_aedt_version() | Get the installed AEDT versions. |
| launch_thread(process, *args) | Launch the thread. |
| serialize_obj_base64(file_path) | Encode a bytes-like object. |
| set_properties(data) | Assign the passed data to the internal data model. |
| wait_to_be_idle([timeout]) | Wait for the thread to be idle and ready to accept a new task. |

This code shows how to use the AEDTCommon class:

```
# Import API
from ansys.aedt.toolkits.common.backend.api import AEDTCommon
from ansys.aedt.toolkits.common.utils import ToolkitThreadStatus
```

(continues on next page)

(continued from previous page)

```
# Object with generic methods to control the toolkits
toolkit = AEDTCommon(properties)

# Get the default properties
properties_from_backend = toolkit.get_properties()

# Set properties, which is useful for setting more than one property
new_properties = {"use_grpc": True, "debug": False}
flag1, msg1 = toolkit.set_properties(new_properties)

# Get new properties
new_properties1 = toolkit.get_properties()

# Get AEDT installed versions
versions = toolkit.installed_aedt_version()

# Launch AEDT in a thread
msg3 = toolkit.launch_thread(toolkit.launch_aedt)

# Wait until thread is finished
toolkit.wait_to_be_idle()

# Get new properties, which should now contain project information
new_properties4 = toolkit.get_properties()

# Connect to the design
flag2 = toolkit.connect_design()

# Create a box
box = toolkit.aedtapp.modeler.create_box([10, 10, 10], [20, 20, 20])
box_name = box.name

# Release AEDT
flag3 = toolkit.release_aedt()
```

CHAPTER

# THREE

# UI API REFERENCE

The PyADT Common Toolkit is designed to streamline the process of creating standard AEDT applications using Pyside6. The UI API provides a set of pre-built components, utilities, and an API that simplifies the development of robust and user-friendly applications.

The UI API contains three main modules: `Utils`, `Windows`, and `Generic actions`:

Utils Common user interface classes to define widgets and load templates.

*Utils* Windows Default Windows initialization.

*Windows* Generic actions Generic methods to call the REST API.

*Actions*

## 3.1 Utils

This section describes the classes for common widgets, objects, and layout templates, which are all designed for versatile application use. While a description of each class follows, you can click the class name to view detailed API reference information.

**``CommonWindowUtils``**

| | |
|---|---|
| *CommonWindowUtils*() | Class representing a common window with various UI functionalities. |

### 3.1.1 ansys.aedt.toolkits.common.ui.utils.windows.common_window_utils.CommonWindo

**class** ansys.aedt.toolkits.common.ui.utils.windows.common_window_utils.**CommonWindowUtils**

Class representing a common window with various UI functionalities.

**__init__**()

#### Methods

| | |
|---|---|
| *__init__*() | |
| add_combobox(layout[, height, width, label, ...]) | Adds a label and combobox to a layout. |
| add_icon_button(layout, icon[, height, ...]) | Add icon button. |

continues on next page

Table 2 – continued from previous page

| | |
|---|---|
| add_n_buttons([layout, num_buttons, height, ...]) | Add a specified number of buttons to a layout object. |
| add_textbox(layout[, height, width, label, ...]) | Adds a label and textbox to a layout. |
| add_toggle(layout[, height, width, label, ...]) | Add a label and a toggle button to a specified layout. |
| add_vertical_line(layout[, top_spacer, ...]) | Add a vertical line. |
| clear_layout(layout) | Clear all layout. |
| create_animation(obj, property_name, ...) | Creates an animation with specified parameters. |
| get_left_menu(object_name) | Retrieves the QPushButton object in the left menu of the CommonWindow UI. |
| get_title_bar(object_name) | Get title. |
| is_left_column_visible() | Check if the left column is visible. |
| is_progress_visible() | Checks if the progress bar is visible. |
| is_right_column_visible() | Checks if the right column is visible. |
| item_index(layout, item) | Item index. |
| remove_item(layout, index) | Remove item by index. |
| set_left_column_menu(menu, title, icon_path) | Configures the left column of the CommonWindow UI by setting the current widget as the provided *menu*, |
| set_page(page) | Set the current page in the load_pages widget. |
| set_right_column_menu(title) | Sets the title of the right column menu. |
| setup_animation(left_start, right_start, ...) | Sets up an animation for the left and right columns of the UI. |
| start_box_animation(direction) | Starts a box animation in the specified direction. |
| toggle_left_column() | Toggles the left column of the CommonWindow by starting a box animation. |
| toggle_progress([mode]) | Toggles the visibility of the progress row. |
| toggle_right_column() | Toggles the display of the right column in a common window. |
| update_logger(text) | Clear all layout. |
| update_progress(progress_value) | Clear all layout. |
| window_refresh() | Window refresh |

**``LoadImages``**

| | |
|---|---|
| *LoadImages*([path]) | A utility class for managing image and icon paths in a PySide6 desktop application. |

### 3.1.2 ansys.aedt.toolkits.common.ui.utils.images.load_images.LoadImages

**class** ansys.aedt.toolkits.common.ui.utils.images.load_images.**LoadImages**(*path=None*)

A utility class for managing image and icon paths in a PySide6 desktop application.

This class facilitates the retrieval of image and icon paths, allowing the application to access and display graphical assets.

**Parameters**

**path**

[str, optional] The base path to the directory containing images. If not provided, the default path is the directory where the script is located.

#### Examples

```
>>> image_loader = LoadImages()
>>> icon_path = image_loader.icon_path("my_icon.png")
>>> image_path = image_loader.image_path("my_image.png")
```

**__init__**(*path=None*)

#### Methods

| | |
|---|---|
| *__init__*([path]) | |
| icon_path(icon_name) | Get the full path for the specified icon. |
| image_path(file_name) | Get the full path for the specified image file. |

``**ThemeHandler**``

| | |
|---|---|
| *ThemeHandler*() | A class for managing themes in a PySide6 desktop application. |

### 3.1.3 ansys.aedt.toolkits.common.ui.utils.themes.json_themes.ThemeHandler

**class** ansys.aedt.toolkits.common.ui.utils.themes.json_themes.**ThemeHandler**

A class for managing themes in a PySide6 desktop application.

This class handles the loading, exporting, and management of themes used in the application.

#### Examples

```
>>> theme_handler = ThemeHandler()
>>> theme_handler.export_theme()
```

**__init__**() → None

#### Methods

| | |
|---|---|
| *__init__*() | |
| export_theme() | Export the current theme to the theme file. |
| read_theme() | Read and load theme settings from the theme file. |

You use the `CommonWindowUtils` class to create custom widgets in the UI. The PyAEDT Common Toolkit also provides the additional widgets described in *Widgets*.

In addition to wrapped PySide6 widgets, the PyAEDT Common Toolkit provides these UI templates to enhance the overall layout:

- `left_column.ui`
- `right_column.ui`
- `main_pages.ui`

These templates serve as a foundation for creating default layouts. You can explore these templates in the ui templates directory of the repository.

### 3.1.4 Widgets

#### PyComboBox

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_combo_box.py_combo_box.**PyComboBox**(*text_list*, *radius=5*, *bg_color='#FFFFFF'*, *bg_color_hover='#F*, *text_color='#000000'*, *font_size=12*)

Combo box widget with customizable elements.

Inherits QComboBox and includes customizable elements including text, radius, color, and background colors in different states.

**Parameters**

**text_list**
[`list`] List of options in combo box.

**radius**
[`int`, `optional`] Radius of combo box corners. The default is `5`.

**bg_color**
[`str`, `optional`] Background color of the combo box. The default is `"#FFFFFF"`.

**bg_color_hover**
[`str`, `optional`] Background color when mouse hovers over the combo box. The default is `"#FFFFFF"`.

**text_color**
[`str`, `optional`] Text color in the combo box. The default is `"#000000"`.

**font_size**
[`int`, `optional`] The font size of the text on the button.

#### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import *
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import *
```

```
>>> class MyApp(QMainWindow):
...     def __init__(self):
...         super().__init__()
...         self.combo_box = PyComboBox(text_list=['Option 1', 'Option 2'],␣
↪radius=5)
...         self.combo_box.show()
```

```
>>> if __name__ == "__main__":
...     app = QApplication([])
...     window = MyApp()
...     sys.exit(app.exec())
```

### PyCredits

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_credits.py_credits.**PyCredits**(*text='© 2024 MyApp Co.', version='0.0.1', bg='#FFFFFF', font_family='Segoe UI', text_size=9, text_description_color='#0( radius=8, padding=10*)

Credits information widget with customizable elements.

Inherits QWidget and includes UILabels for credits and version information, with customizable styles.

**Parameters**

**text**
[str, optional] Copyright text to be displayed. The default is "© 2024 MyApp Co.".

**version**
[str, optional] Version information text to be displayed. The default is "0.0.1".

**bg**
[str, optional] Background color for the widget. The default is "FFFFFF".

**font_family**
[str, optional] Font family name for the text. The default is "Segoe UI".

**text_size**
[int, optional] Size of the text. The default is 9.

**text_description_color**
[str, optional] Color of the text. The default is "#FFFFFF".

**radius**
[int, optional] Radius of the widgets corners. The default is 9.

**padding**
[int, optional] Padding applied to the text in the labels. The default is 10.

#### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import *
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import *
```

```
>>> class MyApp(QMainWindow):
...     def __init__(self):
...         super().__init__()
...         self.credits = PyCredits()
...         self.credits.show()
```

```
>>> if __name__ == "__main__":
...     app = QApplication([])
...     window = MyApp()
...     sys.exit(app.exec())
```

### PyDiv

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_div.py_div.**PyDiv**(*color='#000000'*, *height=0*, *width=20*)

Vertical divider widget with customizable elements.

**Parameters**

**color**
[str, optional] The color of the divider in hex color code. The default is "#000000".

**height**
[float, optional] Divider height. The default is 0.

**width**
[float, optional] Divider width. The default is 20.

#### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import QApplication, QVBoxLayout, QPushButton
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import *
```

```
>>> class Example(QWidget):
>>>     def __init__(self):
>>>         super().__init__()
>>>         layout = QVBoxLayout(self)
>>>         layout.addWidget(QPushButton("Button 1"))
>>>         layout.addWidget(PyDiv("#FF0000", 20))
>>>         layout.addWidget(QPushButton("Button 2"))
```

```
>>> if __name__ == "__main__":
>>>     app = QApplication([])
>>>     window = Example()
>>>     window.show()
>>>     app.exec()
```

### PyIcon

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_icon.py_icon.**PyIcon**(*icon_path*, *icon_color='#000000'*)

Icon widget with customizable elements.

The icon and color can be customized during initialization.

**Parameters**

**icon_path**
[str] Path to the icon image file.

**icon_color**
[str, optional] The color of the icon in hex color code. The default is "#000000".

### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import QApplication, QVBoxLayout, QPushButton
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import *
```

```
>>> class Example(QWidget):
>>>     def __init__(self):
>>>         super().__init__()
>>>         layout = QVBoxLayout(self)
>>>         layout.addWidget(QPushButton("Button 1"))
>>>         layout.addWidget(PyIcon('path_to_icon.svg', "#FF0000"))
>>>         layout.addWidget(QPushButton("Button 2"))
```

```
>>> if __name__ == "__main__":
>>>     app = QApplication([])
>>>     window = Example()
>>>     window.show()
>>>     app.exec()
```

**set_icon**(*icon_path*, *icon_color=None*)

Set icon of the PyIcon widget.

The icon and color can be customized during initialization.

**Parameters**

**icon_path**
[str] Path to the icon image file.

**icon_color**
[str, optional] The color of the icon in hex color code. The default is "#000000".

### PyIconButton

class ansys.aedt.toolkits.common.ui.utils.widgets.py_icon_button.py_icon_button.**PyIconButton**(*icon_path=N*
*tooltip_text=*
*btn_id=None*
*width=30*,
*height=30*,
*ra-*
*dius=8*,
*bg_color='#3*
*bg_color_hov*
*bg_color_pre*
*icon_color='#*
*icon_color_h*
*icon_color_p*
*icon_color_a*
*dark_one='#*
*text_foregrou*
*con-*
*text_color='#*
*top_margin=*
*is_active=Fa*

Icon button widget that can be used as a colored icon.

The icon and color can be customized during initialization.

**Parameters**

**icon_path**
[str] Path to the icon image file.

**tooltip_text**
[str, optional] Text for tooltip.

**btn_id**
[str, optional] Identifier for the button.

**width**
[int, optional] Width of the button.

**height**
[int, optional] Height of the button.

**radius**
[int, optional] Radius for rounded corners.

**bg_color**
[str, optional] Background color in hex color code.

**bg_color_hover**
[str, optional] Background color when hovered in hex color code.

**bg_color_pressed**
[str, optional] Background color when being pressed in hex color code.

**icon_color**
[str, optional] Icon color in hex color code.

**icon_color_hover**
[str, optional] Icon color when hovered in hex color code.

**icon_color_pressed**
[str, optional] Icon color when being pressed in hex color code.

**icon_color_active**
[str, optional] Active icon color in hex color code.

**dark_one**
[str, optional] Color for dark theme in hex color code.

**text_foreground**
[str, optional] Text color in hex color code.

**context_color**
[str, optional] Context color in hex color code.

**top_margin**
[int, optional] Top margin for tooltip.

**is_active**
[bool, optional] Whether the button is currently active.

**The rest of the parameters customize the look and emit signals for different user interactions.**

### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import QApplication, QVBoxLayout, QPushButton
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import *
```

```
>>> class Example(QWidget):
>>>     def __init__(self):
>>>         super().__init__()
>>>         layout = QVBoxLayout(self)
>>>         layout.addWidget(QPushButton("Button 1"))
>>>         layout.addWidget(PyIconButton('icon_signal.svg', "#FF0000", tooltip_
↪text="Example")
>>>         )
>>>         layout.addWidget(QPushButton("Button 2"))
```

```
>>> if __name__ == "__main__":
>>>     app = QApplication([])
>>>     window = Example()
>>>     window.show()
>>>     app.exec()
```

**change_style**(*event*)

Change the style of the button based on the given event.

**Parameters**

**event**
[QEvent] Event triggering the style change.

**enterEvent**(*event*)

Handle the enter event.

**Parameters**

**event**
[QEvent] Enter event.

`icon_paint`(*qp*, *image*, *rect*)

Paint the icon on the button.

**Parameters**

**qp**
[QPainter] QPainter object.

**image**
[str] Path to the icon image file.

**rect**
[QRect] Rectangle for the icon placement.

`is_active`()

Check if the button is in an active state.

**Returns**

**bool**
True if the button is active, False otherwise.

`leaveEvent`(*event*)

Handle the leave event.

**Parameters**

**event**
[QEvent] Leave event.

`mousePressEvent`(*event*)

Handle the mouse press event.

**Parameters**

**event**
[QEvent] Mouse press event.

`mouseReleaseEvent`(*event*)

Handle the mouse release event.

**Parameters**

**event**
[QEvent] Mouse release event.

`paintEvent`(*event*)

Paint the button.

**Parameters**

**event**
[QEvent] Paint event.

`set_active`(*is_active*)

Set the active state of the button.

**Parameters**

**is_active**
[bool] Whether the button is active or not.

**set_icon**(*icon_path*)

Set the icon for the button.

**Parameters**

**icon_path**
[str] Path to the icon image file.

### PyLabel

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_label.py_label.**PyLabel**(*text=''*, *color='#000000'*, *font_size=8*, *font_weight='bold'*)

Label widget with customizable elements.

**Parameters**

**text**
[str, optional] Text to be displayed on the QLabel, by default an empty string.

**color**
[str, optional] Color for the text, in hex format, by default #000000 (black).

**font_size**
[int, optional] Size for the font, by default is 8.

**font_weight**
[str, optional] Weight for the font, by default is bold.

**apply_stylesheet**(*color*, *font_size*, *font_weight*)

Apply the custom styles defined in the class to the QLabel.

**Parameters**

**color**
[str] Text color for the QLabel.

**font_size**
[int] Font size for the QLabel.

**font_weight**
[str] Font weight for the QLabel.

### PyLeftColumn

class ansys.aedt.toolkits.common.ui.utils.widgets.py_left_column.py_left_column.**PyLeftColumn**(*text_title='Ti*
*text_title_size*
*text_title_colo*
*dark_one='#*
*bg_color='#3*
*btn_color='#*
*btn_color_ho*
*btn_color_pr*
*icon_path='n*
*icon_color='*
*icon_color_h*
*icon_color_p*
*con-*
*text_color='#*
*icon_close_p*
*ra-*
*dius=8*)

Custom widget representing a left column with a title, an icon, and a close button.

**Parameters**

**text_title**
[str] The title text for the left column.

**text_title_size**
[int] The font size of the title text.

**text_title_color**
[str] The color of the title text.

**dark_one**
[str] Color representing a dark shade.

**bg_color**
[str] Background color of the left column.

**btn_color**
[str] Color of the close button.

**btn_color_hover**
[str] Color of the close button when hovered.

**btn_color_pressed**
[str] Color of the close button when pressed.

**icon_path**
[str] Path to the icon image file.

**icon_color**
[str] Color of the icon.

**icon_color_hover**
[str] Color of the icon when hovered.

**icon_color_pressed**
[str] Color of the icon when pressed.

**context_color**
[str] Color representing a context or active state.

**icon_close_path**
[str] Path to the close icon image file.

**radius**
[int] Border radius of the left column.

`btn_clicked()`
Emit signal when the close button is clicked.

`btn_released()`
Emit signal when the close button is released.

`setup_ui()`
Set up the user interface for the left column.

### PyLeftButton

`class ansys.aedt.toolkits.common.ui.utils.widgets.py_left_column.py_left_button.PyLeftButton(btn_id=None, width=30, height=30, radius=8, bg_color='#3 bg_color_hov bg_color_pre icon_color=' icon_color_h icon_color_p icon_color_a icon_path='n dark_one='# con- text_color='# text_foregrou is_active=Fa`

Left button widget designed to function as a left-aligned button with various style and interaction options.

**Parameters**

**btn_id**
[str, optional] Button identifier.

**width**
[int, optional] Button width.

**height**
[int, optional] Button height.

**radius**
[int, optional] Button corner radius.

**bg_color**
[str, optional] Button background color.

**bg_color_hover**
[str, optional] Button background color when hovered.

**bg_color_pressed**
[str, optional] Button background color when pressed.

**icon_color**
[str, optional] Icon color.

**icon_color_hover**
[str, optional] Icon color when hovered.

**icon_color_pressed**
[str, optional] Icon color when pressed.

**icon_color_active**
[str, optional] Active icon color.

**icon_path**
[str, optional] Path to icon file.

**dark_one**
[str, optional] Dark color for theming.

**context_color**
[str, optional] Context color for theming.

**text_foreground**
[str, optional] Text foreground color.

**is_active**
[bool, optional] Whether the button is active.

`change_style`(*event*)

Change the button style based on the event type.

**Parameters**

**event**
[QEvent] Event triggering the style change.

`enterEvent`(*event*)

Handle the enter event.

**Parameters**

**event**
[QEvent] Enter event.

`icon_paint`(*qp*, *image*, *rect*)

Paint the icon on the button.

**Parameters**

**qp**
[QPainter] QPainter object.

**image**
[str] Path to the icon image.

**rect**
[QRect] Rectangle to paint the icon within.

`is_active`()

Check if the button is active.

**Returns**

**bool**
True if the button is active, False otherwise.

`leaveEvent`(*event*)

Handle the leave event.

**Parameters**

**event**
[QEvent] Leave event.

`mousePressEvent`(*event*)

Handle the mouse press event.

**Parameters**

**event**
[QEvent] Mouse press event.

`mouseReleaseEvent`(*event*)

Handle the mouse release event.

**Parameters**

**event**
[QEvent] Mouse release event.

`paintEvent`(*event*)

Paint the button.

**Parameters**

**event**
[QEvent] Paint event.

`set_active`(*is_active*)

Set the active state of the button.

**Parameters**

**is_active**
[bool] Whether the button is active.

`set_icon`(*icon_path*)

Set the icon for the button.

**Parameters**

**icon_path**
[str] Path to the icon image file.

### PyLeftMenu

class ansys.aedt.toolkits.common.ui.utils.widgets.py_left_menu.py_left_menu.**PyLeftMenu**(*parent=None*, *app_parent=None*, *dark_one='#1b1e23'*, *dark_three='#21252d'*, *dark_four='#272c36'*, *bg_one='#2c313c'*, *icon_color='#c3ccdf'*, *icon_color_hover='#*, *icon_color_pressed='*, *icon_color_active='#*, *context_color='#568af2'*, *text_foreground='#8*, *text_active='#dce1ec'*, *radius=8*, *minimum_width=50*, *maximum_width=240*, *icon_path='icon_mer'*, *icon_path_close='ico'*, *toggle_text='Hide Menu'*, *toggle_tooltip='Show menu'*)

Custom widget representing a left menu with toggle button, top and bottom layouts, and animated toggle behavior.

**Parameters**

**parent: QWidget, optional**
The parent widget.

**app_parent: QWidget, optional**
The parent widget of the application.

**dark_one: str, optional**
Color representing a dark shade.

**dark_three: str, optional**
Color representing a darker shade.

**dark_four: str, optional**
Color representing an even darker shade.

**bg_one: str, optional**
Background color of the left menu.

**icon_color: str, optional**
Color of the icons in the left menu.

**icon_color_hover: str, optional**
Color of the icons when hovered.

**icon_color_pressed: str, optional**
Color of the icons when pressed.

**icon_color_active: str, optional**
Color of the icons in an active state.

**context_color: str, optional**
Color representing a context or active state.

**text_foreground: str, optional**
Color of the text in the left menu.

**text_active: str, optional**
Color of the text in an active state.

**radius: int, optional**
Border radius of the left menu.

**minimum_width: int, optional**
Minimum width of the left menu. The default is `50`.

**maximum_width: int, optional**
Maximum width of the left menu. The default is `240`.

**icon_path: str, optional**
Path to the icon image file for the toggle button.

**icon_path_close: str, optional**
Path to the icon image file for the toggle button when the menu is closed.

**toggle_text: str, optional**
Text for the toggle button.

**toggle_tooltip: str, optional**
Tooltip text for the toggle button.

`add_menus`(*parameters*)

Add menus to the left menu.

**Parameters**

**parameters: list**
List of dictionaries containing parameters for each menu item.

`btn_clicked`()

Emit signal when a menu button is clicked.

`btn_released`()

Emit signal when a menu button is released.

`deselect_all`()

Deactivate all menu buttons.

`deselect_all_tab`()

Deactivate all menu tabs.

`select_only_one`(*widget: str*)

Set the active state for a specific menu button and deactivate others.

**Parameters**

**widget**
[`str`] ID of the menu button to set as active.

`select_only_one_tab`(*widget: str*)

Set the active tab state for a specific menu button and deactivate others.

**Parameters**

**widget**
[str] ID of the menu button to set as active tab.

`set_visible_button`(*widget: str*, *visible: bool = True*)

Sets the visibility of a QPushButton widget within the UI.

This method searches for a QPushButton widget by its object name and sets its visibility.

**Parameters**

**widget**
[str] The object name of the QPushButton to modify.

**visible**
[bool, optional] A boolean indicating whether the button should be visible. The default is *False*.

`setup_ui`()

Set up the user interface for the left menu.

`toggle_animation`()

Toggle animation for hiding/showing the left menu.

### PyLineEdit

`class` ansys.aedt.toolkits.common.ui.utils.widgets.py_line_edit.py_line_edit.**PyLineEdit**(*text=''*, *place_holder_text=''*, *radius=8*, *border_size=2*, *color='#FFF'*, *selection_color='#FFF'*, *bg_color='#333'*, *bg_color_active='#2*
*context_color='#00ABE*
*font_size=12*)

Custom QLineEdit widget with enhanced styling.

**Parameters**

**text**
[str, optional] The initial text for the line edit. Default is an empty string.

**place_holder_text**
[str, optional] The placeholder text to be displayed when the line edit is empty. Default is an empty string.

**radius**
[int, optional] The border radius of the line edit. Default is 8.

**border_size**
[int, optional] The border size of the line edit. Default is 2.

**color**
[str, optional] The text color of the line edit. Default is #FFF (white).

**selection_color**
[str, optional] The text selection color of the line edit. Default is #FFF (white).

**bg_color**
[str, optional] The background color of the line edit. Default is #333 (dark gray).

**bg_color_active**
[str, optional] The background color of the line edit when active. Default is #222 (darker gray).

**context_color**
[str, optional] The color representing a context or active state. Default is #00ABE8 (blue).

**font_size**
[int, optional] The font size of the text on the button.

**set_stylesheet**(*radius*, *border_size*, *color*, *selection_color*, *bg_color*, *bg_color_active*, *context_color*, *font_size*)

Set the stylesheet for the PyLineEdit.

**Parameters**

**radius**
[int] Border radius of the line edit.

**border_size**
[int] Border size of the line edit.

**color**
[str] Text color of the line edit.

**selection_color**
[str] Text selection color of the line edit.

**bg_color**
[str] Background color of the line edit.

**bg_color_active**
[str] Background color when the line edit is active.

**context_color**
[str] Color representing a context or active state.

**font_size**
[int] The font size of the text on the button.

### PyLogger

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_logger.py_logger.**PyLogger**(*text_color='#f5f6f9'*, *background_color='#000000'*, *font_size=10*, *font_family='Segoe UI'*, *height=50*)

Logger widget.

Inherits QTextEdit and provides a simple interface for logging strings.

**Parameters**

**text_color**
[str, optional] Text color. The default is "#f5f6f9".

**background_color: str, optional**
Color of background. The default is "#000000".

**font_size: float or int, optional**
Font size. The default is 10.

**font_family: str, optional**
Font size. The default is "Segoe UI.

**height: float or int**
Logger height. The default is 10.

`log`(*message*)

Logs a message to the widget.

**Parameters:**
message: The string message to log.

### PyProgress

`class` ansys.aedt.toolkits.common.ui.utils.widgets.py_progress.py_progress.**PyProgress**(*progress=0*, *progress_color='#ff79c6'*, *background_color='#151617'*, *text_color='#FFFFFF'*, *font_size=10*, *font_family='Segoe UI'*, *width=10*)

A progress bar widget.

Inherits QWidget and includes customizable elements including progress, background color, progress color, and width.

**Parameters**

**progress**
[float or int, optional] Current progress value. The default is 0.

**progress_color: str, optional**
Color of progress bar. The default is "#ff79c6".

**background_color: str, optional**
Color of background. The default is "#151617".

**width: float or int**
Width of the progress bar. The default is 10.

### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import *
>>> from ansys.aedt.toolkits.common.ui.utils.widgets.py_progress.py_progress import␣
↪PyProgress
>>> from random import randint
>>> from PySide6.QtCore import QTimer
```

```
>>> class MyApp(QMainWindow):
...     def __init__(self):
...         super().__init__()
...         self.progress_bar = PyProgress(progress=0,
...                                        progress_color="#21252d",
...                                        background_color="#313131",
...                                        width=10)
...         timer = QTimer()
...         timer.timeout.connect(lambda: self.progress_bar.__setattr__("progress",␣
↪randint(0, 100)))
...         timer.start(1000)
...         self.progress_bar.show()
...         sys.exit(app.exec())
```

```
>>> if __name__ == "__main__":
...     app = QApplication([])
...     window = MyApp()
...     sys.exit(app.exec())
```

**paintEvent**(*e*)

Paint the progress bar.

**Parameters**

**e**
[QPaintEvent] Paint event.

**property progress**

Get the current progress value.

**Returns**

**float** or **int**
The current progress value.

## PyPushButton

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_push_button.py_push_button.**PyPushButton**(*text*, *radius*, *color*, *bg_color*, *bg_color_hov*, *bg_color_pre*, *font_size*, *parent=None*)

Initialize the PyPushButton.

**Parameters**

**text**
[str] The title text for the right column.

**radius**
[int] The border radius of the button.

**color**
[str] The text color of the button.

**bg_color**
[str] The background color of the button.

**bg_color_hover**
[int] The background color of the button when hovered.

**bg_color_pressed**
[str] The background color of the button when pressed.

**font_size**
[int] The font size of the text on the button.

**parent**
[str, optional] The parent widget. The default is None.

### PyComboBox

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_right_column.py_right_column.**PyRightColumn**(*text_title*, *text_title_* *text_title_* *dark_one* *bg_color*, *btn_color* *btn_color* *btn_color* *icon_path* *icon_colo* *icon_colo* *icon_colo* *con-* *text_color* *ra-* *dius=8*)

Custom widget representing a right column with a title, an icon, and a content area.

**Parameters**

**text_title**
[str] The title text for the right column.

**text_title_size**
[int] The font size of the title text.

**text_title_color**
[str] The color of the title text.

**dark_one**
[str] Color representing a dark shade.

**bg_color**
[str] Background color of the right column.

**btn_color**
[str] Color of the buttons in the right column.

**btn_color_hover**
[str] Color of the buttons when hovered.

**btn_color_pressed**
[str] Color of the buttons when pressed.

**icon_path**
[str] Path to the icon image file.

**icon_color**
[str] Color of the icon.

**icon_color_hover**
[str] Color of the icon when hovered.

**icon_color_pressed**
[str] Color of the icon when pressed.

**context_color**
[str] Color representing a context or active state.

**radius**
[int] Border radius of the right column.

`setup_ui()`

Set up the user interface for the title bar.

### PySlider

`class ansys.aedt.toolkits.common.ui.utils.widgets.py_slider.py_slider.PySlider(`*margin=0*, *bg_size=20*, *bg_radius=10*, *bg_color='#1b1e23'*, *bg_color_hover='#1e2229'*, *handle_margin=2*, *handle_size=16*, *handle_radius=8*, *handle_color='#568af2'*, *handle_color_hover='#6c99f4'*, *handle_color_pressed='#3f6fd1'*`)`

Custom slider widget with customizable styles.

**Parameters**

**margin**
[int, optional] The margin of the slider, by default 0.

**bg_size**
[int, optional] The background size, by default 20.

**bg_radius**
[int, optional] The background border radius, by default 10.

**bg_color**
[str, optional] The background color, by default #1b1e23.

**bg_color_hover**
[str, optional] The background color on hover, by default #1e2229.

**handle_margin**
[int, optional] The margin of the slider handle, by default 2.

**handle_size**
[int, optional] The size of the slider handle, by default 16.

**handle_radius**
[int, optional] The border radius of the slider handle, by default 8.

**handle_color**
[str, optional] The color of the slider handle, by default #568af2.

**handle_color_hover**
[str, optional] The color of the slider handle on hover, by default #6c99f4.

**handle_color_pressed**
[str, optional] The color of the slider handle when pressed, by default #3f6fd1.

### PyTitleBar

class ansys.aedt.toolkits.common.ui.utils.widgets.py_title_bar.py_title_bar.**PyTitleBar**(*parent*, *app_parent*, *logo_image='ansys-primary-logo-white.svg'*, *logo_width=10*, *dark_one='#1b1e23'*, *bg_color='#343b48'*, *div_color='#3c4454'*, *btn_bg_color='#343b*, *btn_bg_color_hover=*, *btn_bg_color_presse*, *icon_color='#c3ccdf'*, *icon_color_hover='#*, *icon_color_pressed=*, *icon_color_active='#*, *context_color='#6c99f4'*, *text_foreground='#8d*, *radius=8*, *font_family='Segoe UI'*, *title_size=10*)

Custom title bar for the application window.

**Parameters**

**parent**
[QWidget] The parent widget.

**app_parent**
[QWidget] The main application window.

**logo_image**
[str, optional] The path to the logo image file, by default ansys-primary-logo-white.svg.

**logo_width**
[int, optional] The width of the logo, by default 10.

**dark_one**
[str, optional] The color for the dark theme, by default #1b1e23.

**bg_color**
[str, optional] The background color, by default #343b48.

**div_color**
[str, optional] The color for dividers, by default #3c4454.

**btn_bg_color**
[str, optional] The background color for buttons, by default #343b48.

**btn_bg_color_hover**
[str, optional] The background color for buttons on hover, by default #3c4454.

**btn_bg_color_pressed**
[str, optional] The background color for buttons on pressed state, by default #2c313c.

**icon_color**
[str, optional] The default icon color, by default #c3ccdf.

**icon_color_hover**
[str, optional] The icon color on hover, by default #dce1ec.

**icon_color_pressed**
[str, optional] The icon color on pressed state, by default #edf0f5.

**icon_color_active**
[str, optional] The icon color for the active state, by default #f5f6f9.

**context_color**
[str, optional] The context color, by default #6c99f4.

**text_foreground**
[str, optional] The text color, by default #8a95aa.

**radius**
[int, optional] The border radius, by default 8.

**font_family**
[str, optional] The font family, by default Segoe UI.

**title_size**
[int, optional] The font size for the title, by default 10.

`add_menus`(*parameters*)

Add custom menus to the title bar.

**Parameters**

**parameters**
[list of dict] List of dictionaries, each containing information about a menu button. Each dictionary should have the following keys: - btn_icon: str, the icon file for the button. - btn_id: str, the ID of the button. - btn_tooltip: str, the tooltip text for the button. - is_active: bool, whether the button is initially active or not.

`btn_clicked`()

Handle the button click event.

`btn_released`()

Handle the button release event.

`maximize_restore`()

Maximize or restore the application window.

`set_title`(*title*)

Set the title of the application.

**Parameters**

**title**
[str] The title to be set.

`setup_ui`()

Set up the user interface for the title bar.

### PyTitleButton

class ansys.aedt.toolkits.common.ui.utils.widgets.py_title_bar.py_title_button.**PyTitleButton**(*parent*, *app_parent=*, *tooltip_text=*, *btn_id=None*, *width=30*, *height=30*, *radius=8*, *bg_color='#3*, *bg_color_hov*, *bg_color_pre*, *icon_color='*, *icon_color_h*, *icon_color_p*, *icon_color_a*, *icon_path='n*, *dark_one='#*, *context_color='#*, *text_foregrou*, *is_active=Fa*)

Customizable title button.

Inherits QPushButton and provides a customizable title button.

**Parameters**

**parent**
[QWidget] Parent widget.

**app_parent**
[QWidget, optional] Application parent widget. The default is None.

**tooltip_text**
[str, optional] Tooltip text for the button. The default is an empty string.

**btn_id**
[str, optional] Button ID. The default is None.

**width**
[int, optional] Width of the button. The default is 30.

**height**
[int, optional] Height of the button. The default is 30.

**radius**
[int, optional] Border radius of the button. The default is 8.

**bg_color**
[str, optional] Background color of the button. The default is "#343b48".

**bg_color_hover**
[str, optional] Background color when the mouse hovers over the button. The default is "#3c4454".

**bg_color_pressed**
[str, optional] Background color when the button is pressed. The default is "#2c313c".

**icon_color**
[str, optional] Icon color of the button. The default is "#c3ccdf".

**icon_color_hover**
[str, optional] Icon color when the mouse hovers over the button. The default is "#dce1ec".

**icon_color_pressed**
[str, optional] Icon color when the button is pressed. The default is "#edf0f5".

**icon_color_active**
[str, optional] Icon color when the button is active. The default is "#f5f6f9".

**icon_path**
[str, optional] Path to the icon image. The default is "no_icon.svg".

**dark_one**
[str, optional] Dark color for styling. The default is "#1b1e23".

**context_color**
[str, optional] Context color for styling. The default is "#568af2".

**text_foreground**
[str, optional] Text foreground color. The default is "#8a95aa".

**is_active**
[bool, optional] Initial state of the button (active or not). The default is False.

### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import QApplication, QWidget
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import PyTitleButton
```

```
>>> class MyApp(QWidget):
...     def __init__(self):
...         super().__init__()
...         self.title_button = PyTitleButton(self, tooltip_text="Click me!")
...         self.title_button.clicked.connect(self.on_button_clicked)
...         self.title_button.released.connect(self.on_button_released)
...
...     def on_button_clicked(self):
...         print("Button Clicked!")
...
...     def on_button_released(self):
...         print("Button Released!")
...
>>> if __name__ == "__main__":
...     app = QApplication([])
...     window = MyApp()
...     sys.exit(app.exec())
```

**change_style**(*event*)

Change the style of the button based on the given event.

**Parameters**

**event**
[QEvent] The event triggering the style change.

`enterEvent`(*event*)

Event triggered when the mouse enters the button.

**Parameters**

**event**
[QEvent] Mouse enter event.

`icon_paint`(*qp*, *image*, *rect*)

Draw the icon with specified colors.

**Parameters**

**qp**
[QPainter] The QPainter object.

**image**
[str] Path to the icon image.

**rect**
[QRect] Rectangle representing the buttons area.

`is_active`()

Check if the button is in an active state.

**Returns**

**bool**
True if the button is active, False otherwise.

`leaveEvent`(*event*)

Event triggered when the mouse leaves the button.

**Parameters**

**event**
[QEvent] Mouse leave event.

`mousePressEvent`(*event*)

Event triggered when the left mouse button is pressed.

**Parameters**

**event**
[QEvent] Mouse press event.

`mouseReleaseEvent`(*event*)

Event triggered when the left mouse button is released.

**Parameters**

**event**
[QEvent] Mouse release event.

`move_tooltip`()

Move the tooltip to the appropriate position relative to the button.

`paintEvent`(*event*)

Paint the button and its icon.

**Parameters**

**event**
[QEvent] Paint event.

**set_active**(*is_active*)

Set the active state of the button.

**Parameters**

**is_active**
[bool] True to set the button as active, False otherwise.

**set_icon**(*icon_path*)

Set the icon of the button.

**Parameters**

**icon_path**
[str] Path to the icon image.

### PyToggle

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_toggle.py_toggle.**PyToggle**(*width=50*, *bg_color='#777'*, *circle_color='#DDD'*, *active_color='#00BCFF'*, *text_color_on='#FFFFFF'*, *text_color_off='#FFFFFF'*, *show_on_off=False*)

Customizable toggle switch.

Inherits QCheckBox and provides a customizable toggle switch with options for width, background color, circle color, active color, and animation curve.

**Parameters**

**width**
[int, optional] Width of the toggle switch. The default is 50.

**bg_color**
[str, optional] Background color of the toggle switch. The default is "#777".

**circle_color**
[str, optional] Color of the circle in the toggle switch. The default is "#DDD".

**active_color**
[str, optional] Color of the toggle switch when active. The default is "#00BCFF".

**text_color**
[str, optional] Color of the toggle text. The default is "#FFFFFF".

**show_on_off: bool, optional**
Show on and off text in the toggle. The default value is False.

#### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import QApplication
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import PyToggle
```

```
>>> class MyApp(QWidget):
...     def __init__(self):
...         super().__init__()
...         self.toggle = PyToggle()
...         self.toggle.stateChanged.connect(self.toggle_state_changed)
...         self.toggle.show()
```

def toggle_state_changed(self, state): print(Toggle State:, state)

```
>>> if __name__ == "__main__":
...     app = QApplication([])
...     window = MyApp()
...     sys.exit(app.exec())
```

**hitButton**(*pos: QPoint*)

Determine if a button press occurred within the toggle switch.

**Parameters**

**pos**
[QPoint] The position of the button press.

**Returns**

**bool**
True if the button press occurred within the toggle switch, False otherwise.

### PyWindow

**class** ansys.aedt.toolkits.common.ui.utils.widgets.py_window.py_window.**PyWindow**(*parent*, *margin=0*, *spacing=2*, *bg_color='#2c313c'*, *text_color='#fff'*, *text_font="9pt 'Segoe UI'"*, *border_radius=10*, *border_size=2*, *border_color='#343b48'*)

Custom window frame widget with customizable styling and drop shadow effect.

Inherits QFrame and provides a customizable window frame.

**Parameters**

**parent**
[QWidget] The parent widget for this PyWindow.

**margin**
[int, optional] The margin size around the window frame. Default is 0.

**spacing**
[int, optional] The spacing between layout items. Default is 2.

**bg_color**
[str, optional] The background color of the window frame. Default is #2c313c.

**text_color**
[str, optional] The text color of the window frame. Default is #fff.

**text_font**
[str, optional] The font of the text in the window frame. Default is 9pt Segoe UI.

**border_radius**
[int, optional] The border radius of the window frame corners. Default is 10.

**border_size**
[int, optional] The size of the border around the window frame. Default is 2.

**border_color**
[str, optional] The color of the border around the window frame. Default is #343b48.

**set_stylesheet**(*bg_color=None*, *border_radius=None*, *border_size=None*, *border_color=None*, *text_color=None*, *text_font=None*)

Sets the style sheet of the PyWindow with customizable attributes.

**Parameters**

**bg_color**
[str, optional] The background color of the window frame.

**border_radius**
[int, optional] The border radius of the window frame corners.

**border_size**
[int, optional] The size of the border around the window frame.

**border_color**
[str, optional] The color of the border around the window frame.

**text_color**
[str, optional] The text color of the window frame.

**text_font**
[str, optional] The font of the text in the window frame.

### Examples

```
>>> import sys
>>> from PySide6.QtWidgets import QApplication, QMainWindow
>>> from ansys.aedt.toolkits.common.ui.utils.widgets import PyWindow
```

```
>>> class MyApp(QMainWindow):
...     def __init__(self):
...         super().__init__()
...         self.window = PyWindow(self)
...         self.setCentralWidget(self.window)
...         self.show()
```

```
>>> if __name__ == "__main__":
...     app = QApplication([])
...     window = MyApp()
...     sys.exit(app.exec())
```

## 3.2 Windows

The Windows layout template is in the main_window directory in the repository.

The Common_windows directory contains the files for setting up the **main window**, **home menu**, and **settings column**.

The following script shows how to use the previous files to initialize the application.

```
# Import API
import os
import sys
from PySide6.QtWidgets import QApplication

from ansys.aedt.toolkits.common.ui.common_windows.home_menu import HomeMenu
from ansys.aedt.toolkits.common.ui.common_windows.main_window import MainWindow
from ansys.aedt.toolkits.common.ui.common_windows.settings_column import SettingsMenu
from ansys.aedt.toolkits.common.ui.main_window.main_window_layout import (
    MainWindowLayout,
)
from ansys.aedt.toolkits.common.ui.actions_generic import FrontendGeneric


# Object with generic methods to control the toolkits
class ApplicationWindow(FrontendGeneric):
    def __init__(self):

        FrontendGeneric.__init__(self)

        # Create UI object
        self.ui = MainWindowLayout(self)
        self.ui.setup()

        # Set up main window
        self.main_window = MainWindow(self)
        self.main_window.setup()

        # Set up settings menu
        self.settings_menu = SettingsMenu(self)
        self.settings_menu.setup()

        self.home_menu = HomeMenu(self)
        self.home_menu.setup()


if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = ApplicationWindow()
    window.show()
    sys.exit(app.exec())
```

## 3.3 Actions

The `FrontendGeneric` class provides a generic UI for controlling the toolkit. A backend must be running previously.

**class** ansys.aedt.toolkits.common.ui.actions_generic.**FrontendGeneric**

This class provides a generic frontend for controlling the toolkit.

**backend_busy**()

Check if the backend is currently busy.

**Returns**

**bool**
`True` if the backend is busy, `False` otherwise.

**check_connection**()

Check the backend connection.

**Returns**

**bool**
`True` when successful, `False` when failed.

**closeEvent**(*event*)

Handle the close event of the application window.

**find_process_ids**(*version*, *non_graphical*)

Find AEDT sessions based on the selected version and graphical mode.

**Parameters**

**version**
[`str`] AEDT version.

**non_graphical**
[bool] Flag indicating graphical or non-graphical mode.

**Returns**

**list** **or** **False**
A list of found AEDT sessions if successful, `False` otherwise.

**get_aedt_data**()

Get a list of AEDT projects.

**Returns**

**list**
A list of AEDT project names. Returns [No Project] if no projects are available.

**get_aedt_model**(*project_selected*, *design_selected*, *air_objects=True*, *encode=True*, *obj_list=None*, *export_path=None*, *export_as_multiple_objects=False*)

Get AEDT model.

**Parameters**

**project_selected**
[`str`] Project name.

**design_selected**
[`str`] Design name.

**air_objects**
[bool, optional] Define if air and vacuum objects will be exported.

**encode**
[bool, optional] Whether to encode the file. The default is True.

**obj_list**
[list, optional] List of objects to export. The default is None, in which case every model object except 3D, vacuum, and air objects are exported.

**export_path**
[str, optional] Full path of the exported OBJ file. The default is None, in which case the file is exported in the working directory.

**export_as_multiple_objects**
[bool, optional] Whether to export the model as multiple objects or not. Default is False in which case the model is exported as single object.

**Returns**

**bool**
True when successful, False when failed.

`static get_project_name(project_path)`

Get project name from project path.

**Returns**

**str**
Project name

`get_properties()`

Get properties from the backend.

**Returns**

**dict or False**
A dictionary of properties if successful, False otherwise.

`installed_versions()`

Get the installed versions of AEDT.

**Returns**

**list or False**
A list of installed AEDT versions if successful, False otherwise.

`launch_aedt(selected_version, selected_process, non_graphical=False)`

Launch AEDT.

**Parameters**

**selected_version**
[str] The selected AEDT version.

**selected_process**
[str] The selected AEDT process.

**non_graphical**
[bool, optional] Flag indicating whether to run AEDT in non-graphical mode. The default is False.

**log_and_update_progress**(*msg*, *log_level: str = 'debug'*, *progress: int | None = None*)

Log a message and update the progress bar.

This method logs the given message at the specified log level, and updates the progress bar to the given progress percentage if provided.

**Parameters**

**msg**
[str] The log message.

**log_level**
[str, optional] The log level (debug, info, warning, error, critical). The default is debug.

**progress**
[int, optional] The progress percentage. If provided, it updates the progress bar.

**on_cancel_clicked**()

Handle cancel button click.

**open_project**(*selected_project*)

Open an AEDT project.

**Parameters**

**selected_project**
[str] The path to the selected AEDT project.

**static poll_url**(*url: str*, *timeout: int = 10*, *interval: float = 0.5*)

Poll a URL repeatedly until a successful response or a timeout is reached.

This function sends repeated GET requests to the given URL at a fixed interval, stopping when a success response is received or when the specified timeout is exceeded.

**Parameters**

**url**
[str] The URL to poll.

**timeout**
[int, optional] Maximum total time (in seconds) to continue polling before giving up. Default is 10.

**interval**
[float, optional] Time (in seconds) to wait between each request attempt. Default is 0.5.

**Returns**

**tuple[bool, str | dict]**
A 2-tuple containing a boolean and a dict or string. The first element is a boolean stating if the GET requests succeeded. The second element is either the response content (parsed as JSON if successful) or a string describing the failure reason.

**release_and_close**()

Release and close the AEDT desktop.

**release_only**()

Release the AEDT desktop without closing projects.

**save_project**()

Save the current AEDT project.

Opens a file dialog to select a location to save the AEDT project. The project is saved with a .aedt extension.

**Note:**
This method relies on backend communication to save the project.

**Returns**

None

**set_properties**(*data*)

Set properties in the backend.

**Parameters**

**data**
[dict] Dictionary of properties to set.

**update_design_names**(*active_project=None*)

Update design names based on the active project.

**Parameters**

**active_project**
[str, optional] The active AEDT project. If not provided, the current active project will be used.

**Returns**

**list**
A list of design names.

**wait_thread**()

Wait thread until backend is idle.

**Parameters**

**timeout**
[int, optional] Time out in seconds. The default is 10 seconds.

**Returns**

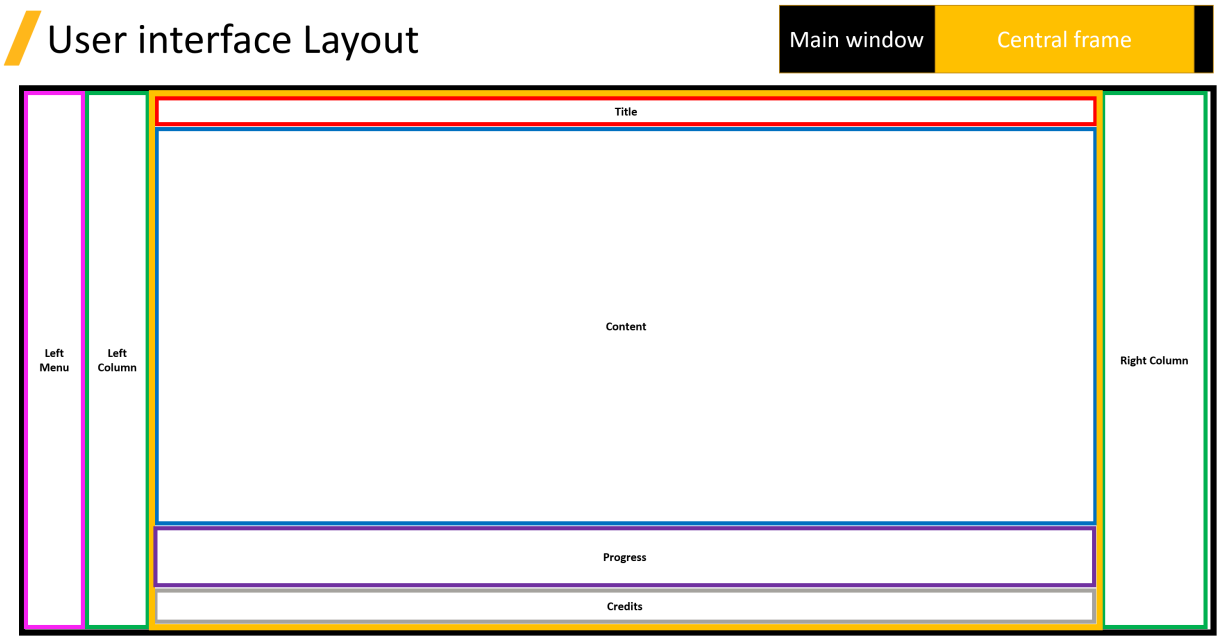
**bool**
True when the backend is idle, False otherwise.

You can modify the default properties with the following script:

```
# Import API
from ansys.aedt.toolkits.common.ui.models import general_settings

general_settings.high_resolution = False
```

This image shows the structure of the UI:




The UI is contained inside the main window. The main window contains some common widgets, such as the credits and title, that are initialized by default. You use the content widget to add new pages.

For initialization information, see the UI example.

CHAPTER

FOUR

# EXAMPLES

End-to-end examples show how to use the three classes in the backend API of the PyAEDT Common Toolkit and its REST API.

## 4.1 The AEDTCommon class

These examples show how to use the AEDTCommon class of the backend API:

### 4.1.1 AEDT simple example

This example shows how to use the AEDTCommon class to launch a new AEDT session in a thread, create an HFSS design, and create a coaxial.

#### Perform required imports

Perform the required imports.

[1]:
```
import sys
from ansys.aedt.toolkits.common.backend.api import AEDTCommon
```

#### Initialize toolkit

Initialize the toolkit.

[2]:
```
toolkit = AEDTCommon()
```

#### Get toolkit properties

Get the toolkit properties.

[3]:
```
properties_from_backend = toolkit.get_properties()
```

#### Set properties

Set non-graphical mode.

[4]:
```
set_properties = {"non_graphical": True}
flag_set_properties, msg_set_properties = toolkit.set_properties(set_properties)
```

```
INFO - Updating internal properties.
```

### Initialize AEDT

Launch a new AEDT session in a thread.

[5]:
```
thread_msg = toolkit.launch_thread(toolkit.launch_aedt)
```

### Wait for the toolkit thread to be idle

Wait for the toolkit thread to be idle and ready to accept a new task.

[6]:
```
idle = toolkit.wait_to_be_idle()
if not idle:
    print("AEDT not initialized.")
    sys.exit()
```

```
INFO - AEDT is released.
```

### Connect design

Connect or create a new design.

[7]:
```
toolkit.connect_design("HFSS")
```

```
INFO - Updating internal properties.
INFO - Toolkit is connected to AEDT design.
```

[7]:
```
True
```

### Get toolkit properties

Get the toolkit properties, which contain the project information.

[8]:
```
new_properties = toolkit.get_properties()
```

### Create a coaxial

Create a coaxial in the design.

[9]:
```
coax = toolkit.aedtapp.modeler.create_coaxial([0, 0, 0], 1)
```

### Release AEDT

Release AEDT.

[10]:
```
toolkit.release_aedt(False, False)
```

```
INFO - AEDT is released.
```

[10]:
```
True
```

### Export AEDT model

Export the OBJ files.

[11]:
```
files = toolkit.export_aedt_model()
```

```
INFO - Toolkit is connected to AEDT design.
INFO - AEDT is released.
```

#### Release and close AEDT

Release and close AEDT.

[12]:
```
toolkit.release_aedt(True, True)
```

```
INFO - AEDT is released.
```

[12]:
```
True
```

### 4.1.2 AEDT open project example

This example shows how to use the `AEDTCommon` class to launch a new AEDT session in a thread and open an existing AEDT project.

#### Perform required imports

Perform the required imports.

[1]:
```
import sys
import os
import shutil
```

[2]:
```
from ansys.aedt.core import generate_unique_folder_name
```

[3]:
```
from ansys.aedt.toolkits.common.utils import download_file
from ansys.aedt.toolkits.common.backend.api import AEDTCommon
```

#### Initialize temporary folder and project settings

Initialize a temporary folder to copy the input file into and specify project settings.

[4]:
```
URL_BASE = "https://raw.githubusercontent.com/ansys/example-data/master/toolkits/common/"
AEDT_PROJECT = "Test.aedt"
URL = os.path.join(URL_BASE, AEDT_PROJECT)

temp_folder = os.path.join(generate_unique_folder_name())

local_project = os.path.join(temp_folder, AEDT_PROJECT)

download_file(URL, local_project)
```

[4]:
```
'C:\\Users\\ansys\\AppData\\Local\\Temp\\pyaedt_prj_B1F\\Test.aedt'
```

#### Initialize toolkit

Initialize the toolkit.

[5]:
```
toolkit = AEDTCommon()
```

### Initialize AEDT

Launch a new AEDT session in a thread.

[6]:
```
thread_msg = toolkit.launch_thread(toolkit.launch_aedt)
```

### Wait for the toolkit thread to be idle

Wait for the toolkit thread to be idle and ready to accept a new task.

[7]:
```
idle = toolkit.wait_to_be_idle()
if not idle:
    print("AEDT not initialized.")
    sys.exit()
```

```
INFO - AEDT is released.
```

### Open project

Open the project.

[8]:
```
open_msg = toolkit.open_project(local_project)
```

```
INFO - Updating internal properties.
INFO - AEDT is released.
```

### Get toolkit properties

Get the toolkit properties, which contain the project information.

[9]:
```
new_properties = toolkit.get_properties()
```

### Connect design

Connect or create a new design.

[10]:
```
toolkit.connect_design()
```

```
INFO - Updating internal properties.
INFO - Toolkit is connected to AEDT design.
```

[10]:
```
True
```

### Create a box

Create a box in the design.

[11]:
```
toolkit.logger.info("Create Box")
box = toolkit.aedtapp.modeler.create_box([10, 10, 10], [20, 20, 20])
model = toolkit.aedtapp.plot(show=True)
```

```
INFO - Create Box
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

#### Save and release AEDT

Save and release AEDT.

```
[12]: toolkit.release_aedt(True, True)
```

```
INFO - AEDT is released.
```

```
[12]: True
```

#### Remove temporary folder

Remove the temporary folder.

```
[13]: shutil.rmtree(temp_folder, ignore_errors=True)
```

### 4.1.3 AEDT connect session example

This example shows how to use the Common AEDT API to connect to an existing AEDT session, create a HFSS design and create a waveguide.

#### Perform required imports

Perform the required imports.

```
[1]: import os
import sys
from ansys.aedt.toolkits.common.backend.api import AEDTCommon
import ansys.aedt.core
```

#### Initialize toolkit

Initialize the toolkit.

```
[2]: toolkit = AEDTCommon()
```

#### Get toolkit properties

Get the toolkit properties, which contain the project information.

```
[3]: properties_from_backend = toolkit.get_properties()
```

```
[4]: # ## Initialize AEDT
#
# Initialize AEDT using PyAEDT and then release it.
ansys.aedt.core.settings.enable_logger = False
app = ansys.aedt.core.Desktop(version=properties_from_backend["aedt_version"],
                              non_graphical=properties_from_backend["non_graphical"])
app.release_desktop(close_projects=False, close_desktop=False)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\ansys\aedt\core\generic\general_methods.py:287, in deprecate_argument.
↪<locals>.decorator.<locals>.wrapper(*args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
    286 try:
--> 287     bound_args = sig.bind_partial(*args, **kwargs)
    288     bound_args.apply_defaults()

File C:\actions-runner\_work\_tool\Python\3.12.10\x64\Lib\inspect.py:3287, in Signature.
↪bind_partial(self, *args, **kwargs)
   3283 """Get a BoundArguments object, that partially maps the
   3284 passed `args` and `kwargs` to the function's signature.
   3285 Raises `TypeError` if the passed arguments can not be bound.
   3286 """
-> 3287 return self._bind(args, kwargs, partial=True)

File C:\actions-runner\_work\_tool\Python\3.12.10\x64\Lib\inspect.py:3269, in Signature._
↪bind(self, args, kwargs, partial)
   3268     else:
-> 3269         raise TypeError(
   3270             'got an unexpected keyword argument {arg!r}'.format(
   3271                 arg=next(iter(kwargs))))
   3273 return self._bound_arguments_cls(self, arguments)

TypeError: got an unexpected keyword argument 'close_desktop'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
Cell In[4], line 7
      4 ansys.aedt.core.settings.enable_logger = False
      5 app = ansys.aedt.core.Desktop(version=properties_from_backend["aedt_version"],
      6                               non_graphical=properties_from_backend["non_
↪graphical"])
----> 7 app.release_desktop(close_projects=False, close_desktop=False)

File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\ansys\aedt\core\generic\general_methods.py:248, in _function_handler_
↪wrapper.<locals>.wrapper(*args, **kwargs)
    246     msg = msg.capitalize()
    247 _exception(sys.exc_info(), user_function, args, kwargs, msg)
--> 248 return raise_exception_or_return_false(e)

File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\ansys\aedt\core\generic\general_methods.py:210, in raise_exception_or_
↪return_false(e)
    208             else:
    209                 v.release_desktop(False, False)
--> 210     raise e
    211 elif "__init__" in str(e):  # pragma: no cover
    212     return

File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\ansys\aedt\core\generic\general_methods.py:223, in _function_handler_
↪wrapper.<locals>.wrapper(*args, **kwargs)
    221 try:
```

(continues on next page)

(continued from previous page)

```
    222     settings.time_tick = time.time()
--> 223     out = user_function(*args, **kwargs)
    224     _log_method(user_function, args, kwargs)
    225     return out

File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\ansys\aedt\core\generic\general_methods.py:291, in deprecate_argument.
↪<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    288     bound_args.apply_defaults()
    289 except TypeError:
    290     # In case of incomplete binding (e.g. missing required args), skip
--> 291     return func(*args, **kwargs)
    293 if arg_name in bound_args.arguments:
    294     msg_version = ""

TypeError: Desktop.release_desktop() got an unexpected keyword argument 'close_desktop'
```

### Get AEDT sessions

Get AEDT sessions and select the first one.

[5]:
```
sessions = toolkit.aedt_sessions()
first_key, first_value = next(iter(sessions.items()))
if first_value == -1:
    use_grpc = False
    selected_process = first_key
else:
    use_grpc = True
    selected_process = first_value
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
Cell In[5], line 2
      1 sessions = toolkit.aedt_sessions()
----> 2 first_key, first_value = next(iter(sessions.items()))
      3 if first_value == -1:
      4     use_grpc = False

StopIteration:
```

### Set properties

Specify the AEDT session selection.

[6]:
```
new_properties = {"selected_process": selected_process, "use_grpc": use_grpc}
flag, msg = toolkit.set_properties(new_properties)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 1
----> 1 new_properties = {"selected_process": selected_process, "use_grpc": use_grpc}
      2 flag, msg = toolkit.set_properties(new_properties)
```

(continues on next page)

(continued from previous page)

```
NameError: name 'selected_process' is not defined
```

### Initialize AEDT

Launch a new AEDT session in a thread.

[7]:
```
thread_msg = toolkit.launch_thread(toolkit.launch_aedt)
```

### Wait for the toolkit thread to be idle

Wait for the toolkit thread to be idle and ready to accept a new task.

[8]:
```
idle = toolkit.wait_to_be_idle()
if not idle:
    print("AEDT not initialized.")
    sys.exit()
```

```
INFO - AEDT is released.
```

### Connect design

Connect or create a new design.

[9]:
```
toolkit.connect_design("HFSS")
```

```
INFO - Updating internal properties.
INFO - Toolkit is connected to AEDT design.
```

[9]:
```
True
```

### Get toolkit properties

Get the toolkit properties, which contain the project information.

[10]:
```
new_properties = toolkit.get_properties()
```

### Create a waveguide

Create a waveguide in the design.

[11]:
```
wg = toolkit.aedtapp.modeler.create_waveguide([0, 0, 0], 1)
model = toolkit.aedtapp.plot(show=True)
```

```
EmbeddableWidget(value='<iframe srcdoc="<!DOCTYPE html>\n<html lang=&quot;en-US&quot;␣
↪dir=&quot;ltr&quot;>\n
```

### Save and release AEDT

Save and release AEDT.

[12]:
```
toolkit.release_aedt(True, True)
```

```
INFO - AEDT is released.
```

[12]:
```
True
```

## 4.2 The EDBCommon class

These examples show how to use the EDBCommon class of the backend API:

### 4.2.1 EDB simple example

This example shows how to use the EDBCommon class to open an existing EDB project.

#### Perform required imports

Perform the required imports.

[1]:
```
import os
import shutil
```

[2]:
```
from ansys.aedt.core import generate_unique_folder_name
```

[3]:
```
from ansys.aedt.toolkits.common.utils import download_file
from ansys.aedt.toolkits.common.backend.api import EDBCommon
```

#### Initialize temporary folder and project settings

Initialize a temporary folder to copy the input file into and specify project settings.

[4]:
```
URL_BASE = "https://raw.githubusercontent.com/ansys/example-data/master/toolkits/common/"
EDB_PROJECT = "edb_edge_ports.aedb/edb.def"
URL = os.path.join(URL_BASE, EDB_PROJECT)

temp_folder = os.path.join(generate_unique_folder_name())

edb_path = os.path.join(temp_folder, "edb_example.aedb")
os.makedirs(edb_path, exist_ok=True)
local_project = os.path.join(edb_path, "edb.def")

download_file(URL, local_project)
```

[4]:
```
'C:\\Users\\ansys\\AppData\\Local\\Temp\\pyaedt_prj_UWT\\edb_example.aedb\\edb.def'
```

#### Initialize toolkit

Initialize the toolkit.

[5]:
```
toolkit = EDBCommon()
```

#### Initialize EDB project

Open the EDB project.

[6]:
```
load_edb_msg = toolkit.load_edb(edb_path)
```

```
C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\site-
↪packages\pyedb\generic\design_types.py:302: UserWarning: This version of the Ansys␣
↪Electronics Database (EDB) is compatible with the gRPCinterface. You can enable gRPC␣
↪by passing ``grpc=True`` when instantiating the Edb object.For more information please␣
↪check this link: https://edb.docs.pyansys.com/version/dev/grpc_api/index.html
  warnings.warn(GRPC_GENERAL_WARNING, UserWarning)
C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\site-
↪packages\pyedb\dotnet\database\components.py:1271: SyntaxWarning: invalid escape␣
↪sequence '\m'
  >>> edb_file = r"C:\my_edb_file.aedb"
C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\site-
↪packages\pyedb\dotnet\database\edb_data\simulation_configuration.py:2043:␣
↪SyntaxWarning: invalid escape sequence '\m'
  >>> sim_setup.output_aedb = r"C:\temp\my_edb.aedb"
C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\site-
↪packages\pyedb\dotnet\database\edb_data\simulation_configuration.py:2710:␣
↪SyntaxWarning: invalid escape sequence '\T'
  >>> config.export_json(r"C:\Temp\test_json\test.json")
C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\site-
↪packages\pyedb\dotnet\database\edb_data\simulation_configuration.py:2738:␣
↪SyntaxWarning: invalid escape sequence '\T'
  >>> test.import_json(r"C:\Temp\test_json\test.json")
C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\site-
↪packages\pyedb\dotnet\database\Variables.py:307: SyntaxWarning: invalid escape␣
↪sequence '\d'
  loc = re.search("[-+]?(\d+(\.\d*)?|\.\d+)([eE][-+]?\d+)?", variable_value)
```

```
PyEDB INFO: Star initializing Edb 18:51:49.416544
PyEDB INFO: Edb version 2025.2
PyEDB INFO: Logger is initialized. Log file is saved to C:\Users\ansys\AppData\Local\
↪Temp\pyedb_ansys.log.
PyEDB INFO: legacy v0.69.0
PyEDB INFO: Python version 3.12.10 (tags/v3.12.10:0cc8128, Apr  8 2025, 12:21:36) [MSC v.
↪1943 64 bit (AMD64)]
PyEDB INFO: Database edb_example.aedb Opened in 2025.2
PyEDB INFO: Cell EMDesign1 Opened
PyEDB INFO: Builder was initialized.
PyEDB INFO: open_edb completed in 8.9983 seconds.
PyEDB INFO: EDB initialization completed in 10.2487 seconds.
```

### Get toolkit properties

Get toolkit properties, which contain the project information.

[7]:
```
new_properties = toolkit.get_properties()
edb_project = new_properties["active_project"]
```

### Save project

Copy the current project in a new file.

[8]:
```
directory, old_file_name = os.path.split(edb_project)
new_path = os.path.join(directory, "new_edb.aedb")
toolkit.save_edb(new_path)
```

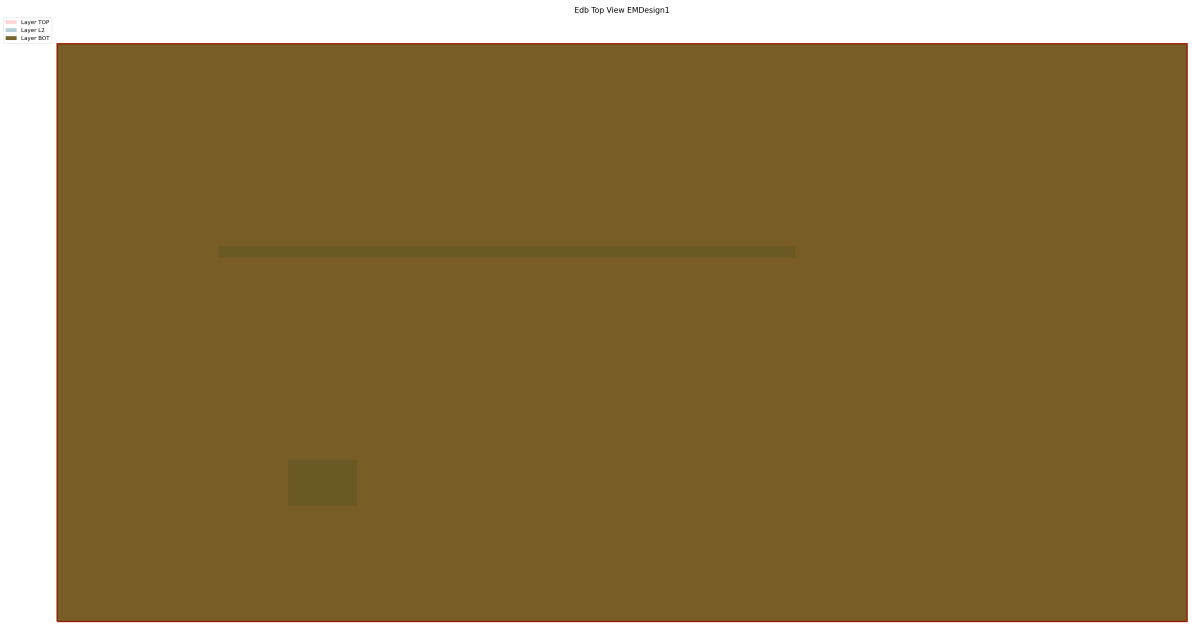
```
PyEDB INFO: EDB file save completed in 0.0405 seconds.
INFO - Project C:\Users\ansys\AppData\Local\Temp\pyaedt_prj_UWT\new_edb.aedb saved
[8]: True
```

### Get cell names

Get cell names using PyEDB.

```
[9]: toolkit.logger.info("Play with EDB")
cell_names = toolkit.edb.cell_names
toolkit.edb.nets.plot()
```

```
INFO - Play with EDB
```




```
PyEDB INFO: Plot Generation time 1.028
[9]: (<Figure size 6000x3000 with 1 Axes>,
 <Axes: title={'center': 'Edb Top View EMDesign1'}>)
```

### Save and release EDB

Save and release EDB.

```
[10]: toolkit.close_edb()
```

```
PyEDB INFO: Close Edb file completed in 0.0230 seconds.
INFO - EDB is closed.
[10]: True
```

#### Remove temporary folder

Remove the temporary folder.

```
[11]: shutil.rmtree(temp_folder, ignore_errors=True)
```

## 4.3 The Common class

These examples show how to use the Common class of the backend API:

### 4.3.1 Properties example

This example shows how to use the Common class, which contains properties models. These properties provide for sharing information through all the workflow.

#### Add new properties

Before importing the common module, you can add new properties. First create a file that contains the new properties type, Models. Then add a TOML file that sets the needed default values. Finally, import the properties.

```
[1]: from models import properties
```

#### Perform required imports

Perform the required imports.

```
[2]: import sys
     from ansys.aedt.toolkits.common.backend.api import Common
```

#### Initialize toolkit

Initialize the toolkit with the new properties.

```
[3]: toolkit = Common(properties)
```

#### Get properties

Get the properties.

```
[4]: toolkit.get_properties()
```

```
[4]: {'aedt_version': '2025.2',
 'non_graphical': False,
 'active_project': '',
 'active_design': '',
 'project_list': [],
 'design_list': {},
 'selected_process': 0,
 'use_grpc': True,
 'is_toolkit_busy': False,
 'url': '127.0.0.1',
 'port': 5001,
 'debug': False,
 'toolkit_name': 'example',
```

(continues on next page)

(continued from previous page)

```
 'log_file': 'example_backend.log',
 'state': '',
 'progress': 0,
 'example': {'invented_property': [10], 'test': 'hola'}}
```

### Set property

Use `set_properties` to set the new property.

[5]:
```
set_properties = {"invented_property": [1, 2, 3]}
toolkit.set_properties(set_properties)
toolkit.get_properties()
```

```
INFO - Updating internal properties.
```

[5]:
```
{'aedt_version': '2025.2',
 'non_graphical': False,
 'active_project': '',
 'active_design': '',
 'project_list': [],
 'design_list': {},
 'selected_process': 0,
 'use_grpc': True,
 'is_toolkit_busy': False,
 'url': '127.0.0.1',
 'port': 5001,
 'debug': False,
 'toolkit_name': 'example',
 'log_file': 'example_backend.log',
 'state': '',
 'progress': 0,
 'example': {'invented_property': [1, 2, 3], 'test': 'hola'}}
```

### Set property directly

Set the property directly.

[6]:
```
properties.invented_property = [10, 20, 30]
toolkit.get_properties()
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[6], line 1
----> 1 properties.invented_property = [10, 20, 30]
      2 toolkit.get_properties()

File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\pydantic\main.py:1033, in BaseModel.__setattr__(self, name, value)
   1031 # if None is returned from _setattr_handler, the attribute was set directly
   1032 elif (setattr_handler := self._setattr_handler(name, value)) is not None:
-> 1033     setattr_handler(self, name, value)  # call here to not memo on possibly␣
↪unknown fields
   1034     self.__pydantic_setattr_handlers__[name] = setattr_handler
```

(continues on next page)

(continued from previous page)

```
File C:\actions-runner\_work\pyaedt-toolkits-common\pyaedt-toolkits-common\.venv\Lib\
↪site-packages\pydantic\main.py:111, in <lambda>(model, name, val)
    105         object.__setattr__(model, '__pydantic_private__', {})
    106     model.__pydantic_private__[name] = val  # pyright:␣
↪ignore[reportOptionalSubscript]
    109 _SIMPLE_SETATTR_HANDLERS: Mapping[str, Callable[[BaseModel, str, Any], None]] = {
    110     'model_field': _model_field_setattr_handler,
--> 111     'validate_assignment': lambda model, name, val:␣
↪model.__pydantic_validator__.validate_assignment(model, name, val),  # pyright:␣
↪ignore[reportAssignmentType]
    112     'private': _private_setattr_handler,
    113     'cached_property': lambda model, name, val: model.__dict__.__setitem__(name,␣
↪val),
    114     'extra_known': lambda model, name, val: _object_setattr(model, name, val),
    115 }
    118 class BaseModel(metaclass=_model_construction.ModelMetaclass):
    119     """!!! abstract "Usage Documentation"
    120         [Models](../concepts/models.md)
    121
   (...)    148         __pydantic_private__: Values of private attributes set on the␣
↪model instance.
    149     """

ValidationError: 1 validation error for Properties
invented_property
  Object has no attribute 'invented_property' [type=no_such_attribute, input_value=[10,␣
↪20, 30], input_type=list]
    For further information visit https://errors.pydantic.dev/2.12/v/no_such_attribute
```

### Set wrong property

Set the wrong property. It is not possible to change the property type.

```
[7]: set_properties = {"invented_property": 1}
     toolkit.set_properties(set_properties)
```

```
INFO - Updating internal properties.
```

```
[7]: (True, 'Properties were updated successfully.')
```

## 4.4 REST API

For an example of using the REST API for the PyAEDT Common Toolkit, see the rest_api_aedt_example.py file in the repository.

CHAPTER

FIVE

# CONTRIBUTE

Overall guidance on contributing to a PyAnsys repository appears in Contributing in the *PyAnsys Developers Guide*. Ensure that you are thoroughly familiar with this guide before attempting to contribute to PyAEDT or its toolkits.

The following contribution information is specific to this library.

## 5.1 Clone the repository

To clone and install the latest version of the PyAEDT Common Toolkit in development mode, run these commands:

```
git clone https://github.com/ansys/pyaedt-toolkits-common.git
cd pyaedt-toolkits-common
python -m pip install --upgrade pip
pip install -e .[all]
```

## 5.2 Post issues

Use the PyAEDT Common Toolkit Issues page to create issues to report bugs and request new features.

## 5.3 View documentation

Documentation for the latest stable release is hosted at PyAEDT Common Toolkit documentation.

In the upper right corner of the documentations title bar, there is an option for switching from viewing the documentation for the latest stable release to viewing the documentation for the development version or previously released versions.

## 5.4 Adhere to code style

The Common Toolkit is compliant with PyAnsys code style. It uses the tool pre-commit to select the code style.

You can install and activate this tool with these commands:

```
pip install pre-commit
pre-commit run --all-files
```

You can also install this as a pre-commit hook with this command:

```
pre-commit install
```

This way, its not possible for you to push code that fails the style checks:

```
$ pre-commit install
$ git commit -am "Add my cool feature."
black....................................................................Passed
isort (python)...........................................................Passed
flake8...................................................................Passed
codespell................................................................Passed
fix requirements.txt.....................................................Passed
blacken-docs.............................................................Passed
```

### 5.4.1 Maximum line length

Best practice is to keep the line length at or below 120 characters for code and comments. Lines longer than this might not display properly on some terminals and tools or might be difficult to follow.

CHAPTER

SIX

# RELEASE NOTES

This document contains the release notes for the project.

## 6.1 0.15.1 - February 11, 2026

### Dependencies

| | |
|---|---|
| Update pypandoc requirement from <1.16,>=1.10.0 to >=1.10.0,<1.17 | #358 |
| Update jupyterlab requirement from <4.5,>=4.0.0 to >=4.0.0,<4.6 | #360 |
| Bump codecov/codecov-action from 5.5.1 to 5.5.2 | #365 |
| Bump actions/download-artifact from 6.0.0 to 7.0.0 | #366 |
| Bump actions/upload-artifact from 5.0.0 to 6.0.0 | #367 |
| Bump build from 1.3.0 to 1.4.0 | #372 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.15.0 | #355 |
| Bump dev version | #356 |
| Update missing or outdated files | #361 |
| Add ``ansys/actions/check-actions-security`` action and related fixes | #362 |
| Make ``ansys/actions/check-actions-security`` dependent on dependabot checks | #368 |
| Update license header | #371 |
| Standardize timeout values across calls to ``requests`` methods and enable definition via env variable | #376 |
| Add dependabot cooldown settings | #377 |

### Miscellaneous

| | |
|---|---|
| Fixed tests | #369 |

## 6.2 0.15.0 - November 12, 2025

### Added

| | |
|---|---|
| Updated the project browse dialog box to native windows dialog box | #343 |
| Add synopsys theme | #352 |

### Dependencies

| | |
|---|---|
| Bump ansys/actions from 10.1.1 to 10.1.2 | #337 |
| Bump ansys/actions from 10.1.2 to 10.1.4 | #339 |
| Bump peter-evans/create-or-update-comment from 4.0.0 to 5.0.0 | #340 |
| Update pydantic requirement from <2.12,>=2.0 to >=2.0,<2.13 | #346 |
| Bump actions/upload-artifact from 4.6.2 to 5.0.0 | #349 |
| Bump actions/download-artifact from 5.0.0 to 6.0.0 | #350 |
| Bump ansys/actions from 10.1.4 to 10.1.5 | #351 |

### Documentation

| | |
|---|---|
| Update ``CONTRIBUTORS.md`` with the latest contributors | #345 |

### Fixed

| | |
|---|---|
| Synopsys theme fix | #353 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.14.3 | #336 |
| Remove usage of vtk-osmesa package | #354 |

## 6.3 0.14.3 - September 26, 2025

### Fixed

| | |
|---|---|
| Fixed change of pyaedt on desktop project_list which is now a property | #334 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.14.2 | #333 |
| Remove caching from wheelhouse | #335 |

## 6.4 0.14.2 - September 25, 2025

### Fixed

| | |
|---|---|
| Change from one project to another | #332 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.14.1 | #331 |

## 6.5 0.14.1 - September 24, 2025

### Dependencies

| | |
|---|---|
| Bump twine from 6.1.0 to 6.2.0 | #323 |
| Update pytest-cov requirement from <6.3,>=4.0.0 to >=4.0.0,<7.1 | #327 |
| Bump ansys/actions from 10.0.20 to 10.1.1 | #329 |

### Documentation

| | |
|---|---|
| Update `CONTRIBUTORS.md` with the latest contributors | #324 |

### Fixed

| | |
|---|---|
| Deprecated pyaedt argument | #330 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.14.0 | #321 |
| Bump 0.15.dev0 | #322 |
| Update README.rst | #325 |

## 6.6 0.14.0 - September 10, 2025

### Dependencies

| | |
|---|---|
| Update flit-core requirement from <3.11,>=3.2 to >=3.2,<4 | #229 |
| Bump codecov/codecov-action from 5.4.3 to 5.5.0 | #311 |
| Bump actions/setup-python from 5.6.0 to 6.0.0 | #314 |
| Bump ansys/actions from 10.0.14 to 10.0.20 | #316 |
| Bump pypa/gh-action-pypi-publish from 1.12.4 to 1.13.0 | #317 |
| Bump actions/labeler from 5.0.0 to 6.0.1 | #318 |
| Bump codecov/codecov-action from 5.5.0 to 5.5.1 | #319 |

### Fixed

| | |
|---|---|
| Update documentation sphinx dependency | #299 |
| Fix pyside version in tests and doc | #307 |
| Pydantic deprecation and CI warning spotted in CI logs | #309 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.13.3 | #306 |
| Strengthen workflows job dependencies | #313 |
| Update SECURITY.md | #320 |

### Test

| | |
|---|---|
| Fix flaky test using geometry thread | #308 |
| Improve menu testing | #312 |

## 6.7 0.13.3 - August 21, 2025

### Dependencies

| | |
|---|---|
| Bump actions/checkout from 4.2.2 to 5.0.0 | #304 |

### Maintenance

| | |
|---|---|
| Update CHANGELOG for v0.13.2 | #303 |
| Revert pyside6 6.9.0 | #305 |

## 6.8 0.13.2 - August 14, 2025

### Dependencies

| | |
|---|---|
| Bump ansys/actions from 10.0.12 to 10.0.14 | #300 |
| Bump actions/download-artifact from 4.3.0 to 5.0.0 | #301 |
| Bump build from 1.2.2.post1 to 1.3.0 | #302 |

### Maintenance

| | |
|---|---|
| Update changelog for v0.13.1 | #294 |
| Pin vtk-osmesa version | #296 |
| Use aedt 2025r2 | #297 |

## 6.9 0.13.1 - July 19, 2025

### Dependencies

| | |
|---|---|
| Update pytest-qt requirement from <4.5,>=4.0.0 to >=4.0.0,<4.6 | #291 |
| Bump ansys/actions from 10.0.11 to 10.0.12 | #292 |

### Documentation

| | |
|---|---|
| Update `contributors.md` with the latest contributors | #293 |

### Maintenance

| | |
|---|---|
| Update changelog for v0.13.0 | #288 |
| Update v0.14.dev0 | #289 |
| Add safety check to all dependencies | #290 |

## 6.10 0.13.0 - July 07, 2025

### Added

| | |
|---|---|
| Add last example tests | #281 |

### Dependencies

| | |
|---|---|
| Update pytest requirement from <8.4,>=7.4.0 to >=7.4.0,<8.5 | #274 |
| Bump pyside6 from 6.9.0 to 6.9.1 | #275 |
| Update pytest-cov requirement from <6.2,>=4.0.0 to >=4.0.0,<6.3 | #277 |
| Update numpydoc requirement from <1.9,>=1.5.0 to >=1.5.0,<1.10 | #287 |

### Documentation

| | |
|---|---|
| Added deepwiki badge | #286 |

### Maintenance

| | |
|---|---|
| Update changelog for v0.12.6 | #273 |
| Cleanup and updates | #280 |
| Add vulnerability check and refactor the code accordingly | #285 |

### Miscellaneous

| | |
|---|---|
| Improve example and test ui | #276 |

## 6.11 0.12.6 - June 13, 2025

### Dependencies

| | |
|---|---|
| Bump ansys/actions from 9 to 10 | #272 |

### Maintenance

| | |
|---|---|
| update CHANGELOG for v0.12.5 | #271 |

## 6.12 0.12.5 - June 06, 2025

### Documentation

| | |
|---|---|
| add doc section for distribution | #269 |
| Update distributing.rst | #270 |

### Maintenance

| | |
|---|---|
| update CHANGELOG for v0.12.4 | #268 |

## 6.13 0.12.4 - June 02, 2025

### Documentation

| | |
|---|---|
| Update `CONTRIBUTORS.md` with the latest contributors | #266 |

### Fixed

| | |
|---|---|
| Widget misaligment | #267 |

### Maintenance

| | |
|---|---|
| update CHANGELOG for v0.12.3 | #265 |

## 6.14 0.12.3 - May 30, 2025

### Added

| | |
|---|---|
| Auto resolution | #264 |

### Maintenance

| | |
|---|---|
| update CHANGELOG for v0.12.2 | #262 |
| Add changelog upper case | #263 |

## 6.15 0.12.2 - May 26, 2025

### Added

| | |
|---|---|
| Add specific application if passed | #260 |
| Add ON/OFF in toggle | #261 |

### Maintenance

| | |
|---|---|
| update CHANGELOG for v0.12.1 | #257 |

## 6.16 0.12.1 - May 20, 2025

### Added

| | |
|---|---|
| Add set_visible_button for left menu | #256 |

### Maintenance

| | |
|---|---|
| update CHANGELOG for v0.12.0 | #252 |
| Update v0.13.dev0 | #253 |

## 6.17 0.12.0 - May 10, 2025

### Maintenance

| | |
|---|---|
| Update Python 3.12 | #248 |

CHAPTER

SEVEN

# DISTRIBUTE

Distributing your application in a distributable format means packaging it in a way that makes it easy to install, run and share. A distributable format bundles everything needed (dependencies, configurations, executables) into a convenient package. Packaging helps making your toolkit easier to distribute and more likely to run consistently across systems.

## 7.1 How to distribute in Windows

The aim is to package the toolkit into a Windows installer (EXE file) using NSIS (Nullsoft Scriptable Install System).

### 7.1.1 Pre-requisites and setup

1. Install chocolatey (Windows package manager) Chocolatey lets you install tools like NSIS easily. For example, you can install NSIS by: - Opening a PowerShell terminal as Administrator. - Running the following command

```
Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::
↪SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex␣
↪((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/
↪install.ps1'))
```

Visit the Chocolatey website for more information on the installation process.

2. Add NSIS to your PATH in Windows environment variables Usually the NSIS is located at `C:\Program Files (x86)\NSIS`.

3. Install NSIS using chocolatey

```
choco install nsis -y
```

4. Install PyInstaller PyInstaller bundles a Python application and all its dependencies into a single package. The user can run the packaged app without installing a Python interpreter or any modules.

This step requires the toolkit TOML file to have a section called `freeze` in `[project.optional-dependencies]`. An example can be found in Antenna Wizard TOML file.

Open the toolkit project in your IDE of choice, activate your virtual environment and run the following command:

```
pip install .[freeze]
```

5. Extract the toolkit version This step requires you to have the extract_version.py python script.

```
python installer/extract_version.py
```

6. Create the standalone executable This step requires you to have a *frozen.spec* file. An example of such file can be found here. This file plays a key role in how your toolkit is turned into a standalone executable. It defines the instructions PyInstaller uses to package your toolkit into a single executable file.

```
pyinstaller frozen.spec
```

7. Create a standalone installer program This step requires you to have a *setup.nsi* file. An example of such file can be found here. The `setup.nsi` file is the NSIS script (a plain text file) that describes how to build the installer, that is what files to include, where to install them, shortcuts to create, etc., and it compiles this script into a Windows installer executable. In simple terms `setup.nsi` contains instructions and by running:

```
makensis setup.nsi
```

NSIS turns these instructions into a standalone installer program.

CHAPTER

EIGHT

# WHAT IS THIS LIBRARY?

This library is a common library for AEDT toolkits. It brings numerous advantages like enhancing efficiency, consistency, and collaboration in the creation and development of AEDT toolkits.

This library provides methods and best practices to develop AEDT toolkits. These are the main advantages of using this library:

## 8.1 Standardization

A common toolkit framework establishes standardized guidelines, conventions, and practices for toolkit development. This ensures that all developers follow a consistent structure, making it easier for them to understand and contribute to different toolkits.

## 8.2 Interoperability

With a common framework, different AEDT toolkits become more interoperable. Developers can create tools that seamlessly integrate with existing toolkits, promoting a modular and extensible ecosystem. This interoperability is crucial for users who may need functionalities from multiple toolkits in their workflows.

## 8.3 Reuse code

A common toolkit framework encourages the reuse of code components across different toolkits. Developers can leverage existing modules, functions, and libraries, saving time and effort in creating similar functionalities from scratch. This promotes a more efficient development process and reduces redundancy.

## 8.4 Maintain

Standardized frameworks make it easier to maintain and update AEDT toolkits. When changes or improvements are required, developers can follow a unified set of procedures, ensuring that updates are applied consistently across all toolkits. This helps in avoiding compatibility issues and streamlining the maintenance process.

## 8.5 Collaboration

A common toolkit framework fosters collaboration among developers by providing a shared set of tools and practices. It becomes easier for multiple developers to work on different aspects of the toolkit simultaneously, as they all adhere to the same framework. This collaborative environment can lead to faster development cycles and a more robust set of tools.

## 8.6 Documentation and training

A standardized framework comes with consistent documentation and training resources. This makes it easier for developers to understand the structure of the toolkit, its functionalities, and best practices.

## 8.7 Quality assurance

A common toolkit framework enables better quality assurance processes. Standardized testing methodologies and practices can be implemented, ensuring that each toolkit adheres to the same quality standards. This results in more reliable and stable toolkits for end-users.

CHAPTER

NINE

# INDICES AND TABLES

- genindex
- modindex
- search