Pyedb Explorer User Guide

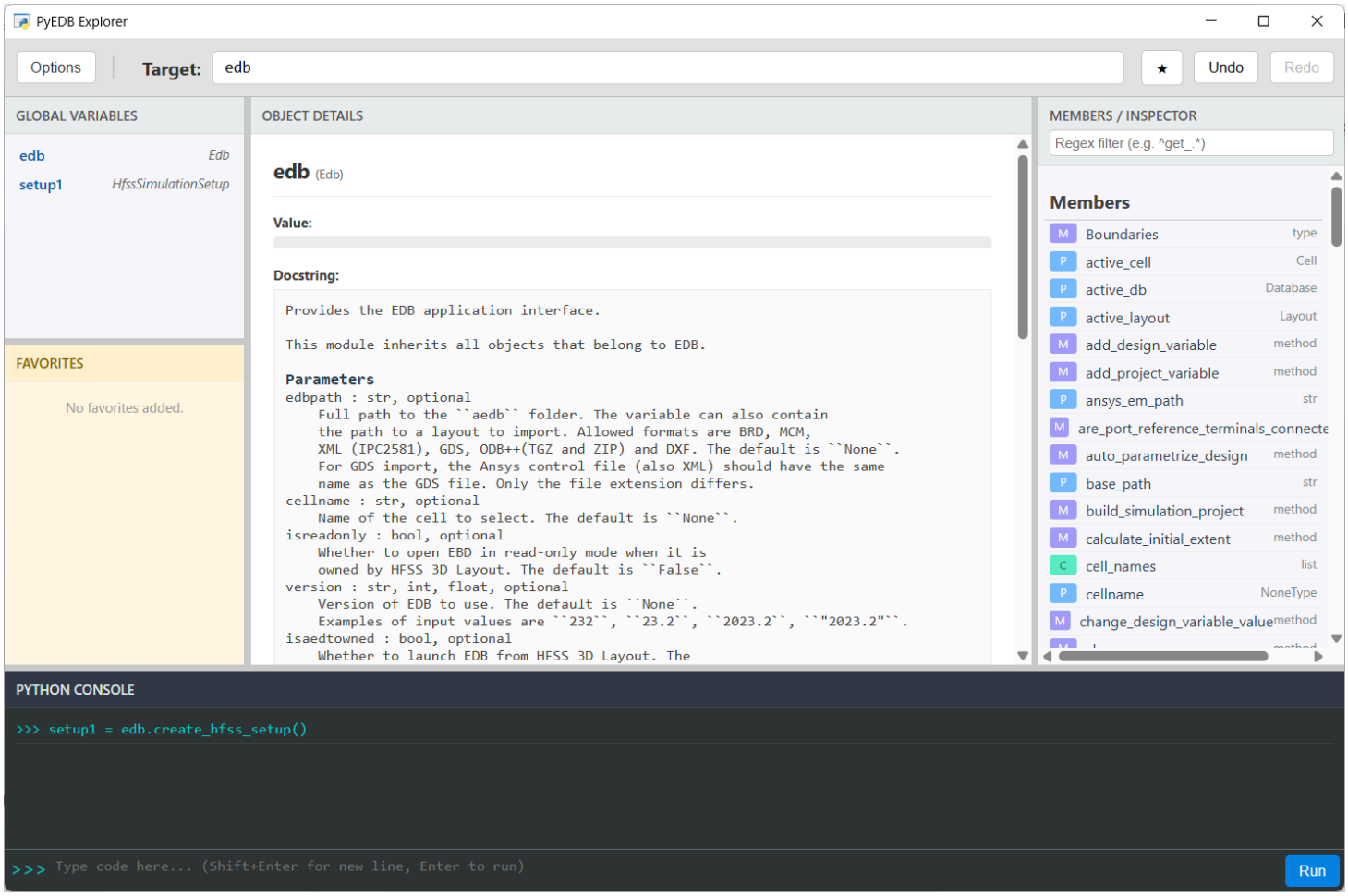
Pyedb is a Python API library for editing EDB files. It can load a `.aedb` file and perform queries and edits, configure ports, and run HFSS 3D Layout simulations. Core capabilities include:

- Load a `.aedb` file and show its structure.
- Query and edit component properties.
- Configure ports and connections.
- Create HFSS or SIwave simulations.

While Pyedb is powerful, its rich object graph and extensive API can make the learning curve steep. Developers often need to rely on `dir()` and `help()` to explore APIs, which is cumbersome for beginners.

To lower the barrier and improve usability, we built Pyedb Explorer - a PyWebView-based GUI that lets you visually explore EDB objects and their properties, with convenient actions to quickly understand relationships and documentation. This greatly improves simulation automation development efficiency and user experience.

Pyedb Explorer Features



When Pyedb Explorer starts, it loads a built-in sample `.aedb` file and shows its top-level `edb` object:

```
from pyedb import Edb
edb = Edb('pcb.aedb')
```

Click the `edb` object to see its full path in the Target pane, with the middle pane showing the object, its type, and value. Below is the docstring, and the right pane lists the object's properties and methods.

- M: Object methods.
- P: Property names and their types.
- C: Collection type, such as a list or dict.

Click a property to view its value and description. If that property is itself an object, you can continue drilling into it to explore its properties and methods. This makes the EDB object structure and relationships easy to grasp.

Pyedb Explorer also provides a Regex Filter search so you can enter keywords to search for properties or methods and quickly locate information - especially helpful when navigating large EDB

files.

Example: selecting the `stackup` property shows it is a `Stackup` object. The Target pane shows the full path `edb.stackup`, the middle pane shows detailed info and description, and the right pane lists the `stackup` object's properties and methods. You can continue this pattern to dig deeper into the EDB hierarchy.

Selecting `signal_layers` shows it is an `OrderedDict`. In addition to listing its properties and methods, the right pane's Collection Items section lists all keys. Clicking key `1_Top` shows the corresponding `Layer` details in the middle pane, and the right pane lists the `StackupLayerEdbClass` properties and methods. This makes it easy to browse collection-type objects.

Use the star on the right of the Target pane to add the current object to the Favorites list on the left for quick access.

The Undo/Redo buttons next to Target let you jump back and forth between recently browsed objects, making it easier to compare items.

The PYTHON CONSOLE at the bottom is an interactive Python prompt. Enter Python commands directly to operate on EDB objects and see results immediately - handy for quick tests and validation.

For example, the docstring for `edb.create_hfss_setup` is:

```
Create an HFSS simulation setup from a template.

Parameters
name : str, optional
    Setup name.

Returns
:class:`legacy.database.edb_data.hfss_simulation_setup_data.HfssSimulationSetup`

Examples
>>> from pyedb import Edb
>>> edbapp = Edb()
>>> setup1 = edbapp.create_hfss_setup("setup1")
>>> setup1.hfss_port_settings.max_delta_z0 = 0.5
```

To see what properties and methods the returned `setup1` object has, use the Python Console. Enter:

```
setup1 = edbapp.create_hfss_setup("setup1")
```

The `setup1` object is created and added to the left Global Variables list. Selecting it shows the object in Target, its description in the middle pane, and all properties and methods in the right pane - helpful for understanding and testing the object's structure and behavior.

If you no longer need `setup1`, enter:

```
del(setup1)
```

If you do not need the Python Console, disable it in Options for a cleaner interface.

Advanced Tip: Load a Custom `.aedb`

Pyedb Explorer builds objects from a real `.aedb` and uses `dir()` / `help()` to query properties and methods dynamically. Some objects may not expose their docstrings correctly in the default sample file. A simple workaround is to load a customer-provided `.aedb`. For example, the default `pcb.aedb` has no DCIR voltage source, so related properties and methods are absent. If the customer supplies a `.aedb` that includes a DCIR source, those attributes and methods become available. Use the Load Custom .aedb option to load such a file and explore more components with their full documentation.