PyEDB

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

Jun 28, 2024

CONTENTS

PyEDB is a Python library that interacts directly with the PyEDB-Core API to make scripting simpler.

Getting started Learn more about PyEDB and how to install and use it. Also view important version, interface, and troubleshooting information.

Getting started User guide Understand how to use PyEDB by looking at some simple tutorials.

User guide API reference Understand PyEDB API endpoints, their capabilities, and how to interact with them programmatically.

API reference Examples Explore examples that show how to use PyEDB to perform many different types of simulations.

examples Contribute Learn how to contribute to the PyEDB codebase or documentation.

getting_started/contributing

CHAPTER

ONE

GETTING STARTED

1.1 About PyEDB

PyEDB is part of the larger PyAnsys effort to facilitate the use of Ansys technologies directly from Python. It is intended to consolidate and extend all existing functionalities around scripting for the Ansys Electronics Database (EDB) to allow reuse of existing code, sharing of best practices, and increased collaboration.

PyEDB includes functionality for interacting with these Ansys Electronics Desktop (AEDT) products:

- EDB
- HFSS 3D Layout
- Icepak

1.1.1 What is EDB?

EDB provides a proprietary database file format (AEDB) for efficient and fast layout design handling and processing for building ready-to-solve projects. EDB addresses signal integrity (SI), power integrity (PI-DC), and electro-thermal workflows. You can import an AEDB file into AEDT to modify the layout, assign materials, and define ports, simulations, and constraints. You can then launch any of the Ansys electromagnetic simulators: HFSS, HFSS 3D Layout, Icepak, Maxwell, Q3D, and SIwave.

EDB runs as a standalone API, which means that you dont need to open a user interface (UI). Because EDB opens the `aedb` folder for directly querying and manipulating layout design in memory, it provides the fastest and most efficient way to handle a large and complex layout.

You can also parse an AEDB file from a command line in batch in an Ansys electromagnetic simulator like HFSS or SIwave. Thus, you can deploy completely non-graphical flows, from layout translation through simulation results.

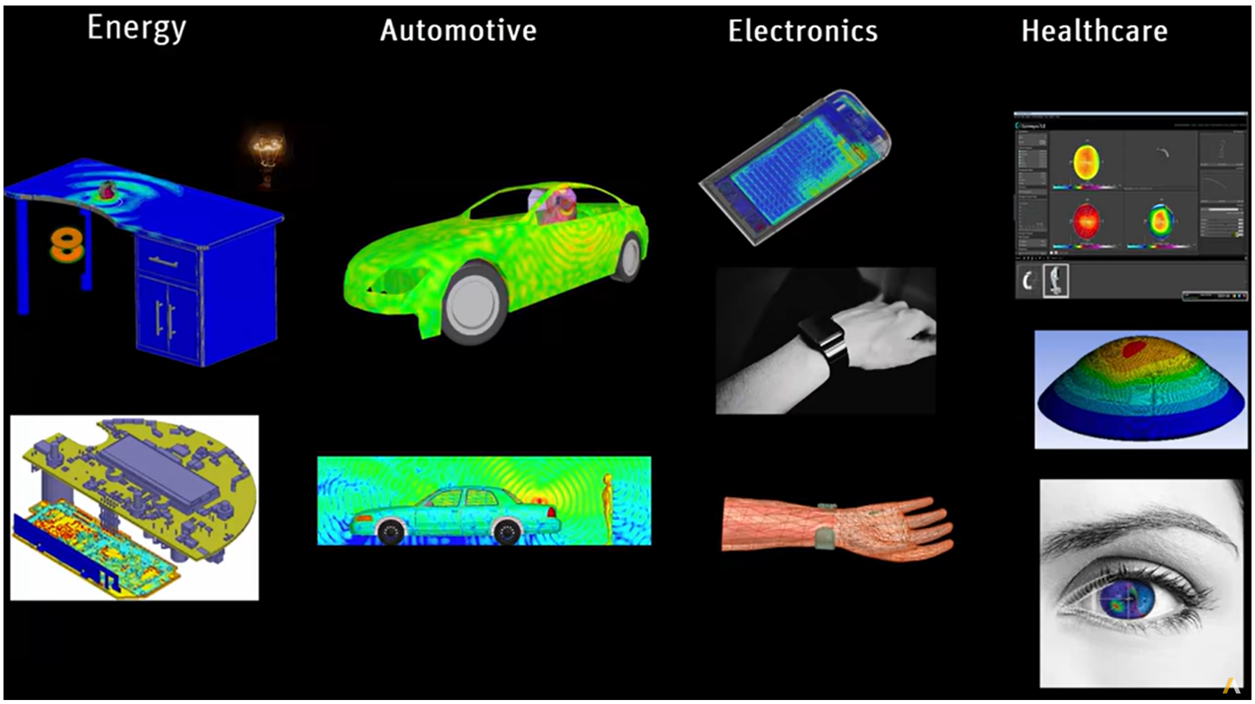
Additionally, you can use PyAEDT to import an AEDB file into AEDT to view a project, combine 3D designs, or perform simulation postprocessing. EDB also supports 3D component models.

1.1.2 Why use PyEDB?

PyEDB interacts with the PyEDB-Core API to make scripting simpler. It provides application-oriented, high-level methods and properties. The PyEDB APIs `Edb` class and methods simplify operations while reusing information as much as possible across the API.

Because PyEDB runs in memory, it does not require a user interface. Its API is extremely efficient at handling and editing large and complex layout designs. PyEDB is the best choice for addressing layout design automation. Its headless architecture also makes it well suited on both Windows and Linux.

PyEDB loads and saves AEDB files, which can natively be read by AEDT and Ansys SIwave to visualize and edit projects, run simulations, or perform postprocessing. AEDB files are project self-contained, meaning that ready-to-solve projects can be written with PyEDB. Therefore Ansys solvers can directly load AEDB files graphically or in batch non-graphically to support submission for job scheduling on a cluster.



For more information, see Ansys Electronics on the Ansys website.

1.2 Installation

PyEDB consolidates and extends all existing capital around scripting for EDB, allowing reuse of existing code, sharing of best practices, and collaboration.

PyEDB has been tested on HFSS, Icepak, and SIWave.

1.2.1 Requirements

To use PyEDB, you must have a licensed copy of AEDT 2023 R2 or later.

PyEDB also supports the AEDT Student version 2023 R2 or later. For more information, see the Ansys Electronics Desktop Student - Free Software Download page on the Ansys website.

Any additional runtime dependencies are listed in the following installation topics.

1.2.2 Install from a Python file

The AEDT installation already provides a Python interpreter that you can use to run PyEDB. In a virtual environment, you can run PyEDB using CPython 3.9 through 3.11. Note that AEDT 2024 R1 installs CPython 3.10.

You can install PyEDB offline using a wheelhouse, which is a ZIP file containing all the needed packages. The Releases page of the PyEDB repository provides an **Assets** ares with the PyEDB wheelhouses for various Python releases on different operating system.

After downloading the wheelhouse for your Python release and operating system, run the script from the Python terminal, providing the full path to the ZIP file as an argument.

1.2.3 Install on CPython from PyPI

You can install PyEDB on CPython 3.8 through 3.11 from PyPI, the Python Package Index, with this command:

```
pip install pyedb
```

1.2.4 Linux support

PyEDB works with CPython 3.8 through 3.10 on Linux in AEDT 2022 R2 and later. However, you must set up the following environment variables:

```
export ANSYSEM_ROOT222=/path/to/AedtRoot/AnsysEM/v222/Linux64
export LD_LIBRARY_PATH=$ANSYSEM_ROOT222/common/mono/Linux64/lib64:$ANSYSEM_ROOT222/
↪Delcross:$LD_LIBRARY_PATH
```

1.2.5 Install offline from a wheelhouse

Using a wheelhouse can be helpful if you work for a company that restricts access to external networks. A wheelhouse is a ZIP file that contains all dependencies for package and allows full installation without a need to download additional files. Having a single file eases the security review of the package content and allows for easy sharing with others who need to install it.

On the Releases page of the PyEDB repository, the **Assets** area shows the wheelhouses that are available. After downloading the wheelhouse for your setup, extract the files to a folder and run the command for installing PyEDB and all of its dependencies from your Python terminal, providing the full path to the ZIP file as an argument.

```
pip install --no-cache-dir --no-index --find-links=/path/to/pyedb/wheelhouse pyedb
```

For example, on Windows with Python 3.8, install PyEDB and all its dependencies from a wheelhouse with code like this:

```
pip install --no-cache-dir --no-index --find-links=file:///<path_to_wheelhouse>/PyEDB-v
↪<release_version>-wheelhouse-Windows-3.8 pyedb
```

1.2.6 Update PyEDB to the latest version

After installing PyEDB, upgrade it to the latest version with this command:

```
pip install -U pyedb
```

1.3 Quick code

To help you begin using PyEDB, you can view or download the PyEDB API cheat sheet. This one-page reference provides syntax rules and commands for using the PyEDB API:

- View the PyEDB API cheat sheet.
- Download the PyEDB API cheat sheet.

1.3.1 Load an AEDB file into memory

This code shows how to use PyEDB to load an existing AEDB file into memory:

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")
```

1.3.2 Connect to EDB from a Python IDE

PyEDB works both inside AEDT and as a standalone app. PyEDB also provides advanced error management. The following code examples provide a brief example of how PyEDB works.

Explicit PyEDB declaration and error management

```
# Start EDB

from pyedb import Edb

edb_file = pyedb.layout_examples.ANSYS - HSD_V1.aedb
edb = Edb(edbversion="2024.1", edbpath=edb_file)
```

Variables

```
from pyedb import Edb

edb_file = pyedb.layout_examples.ANSYS - HSD_V1.aedb
edb = Edb(edbversion="2024.1", edbpath=edb_file)
edb["dim"] = "1mm"  # design variable
edb["$dim"] = "1mm"  # project variable
```

1.4 Versions and interfaces

PyEDB attempts to maintain compatibility with legacy versions of EDB while allowing for support of faster and better interfaces with the latest versions of EDB.

Currently, there is only one interface PyEDB can use to connect to EDB.

1.4.1 gRPC interface

The gRPC interface is under development and should be available soon.

1.4.2 Legacy interface

PyEDB currently connects to EDB using the native C# interface for the EDB API. You do not need to set the `PYEDB_USE_DOTNET` environment variable to `0` to use the legacy interface because it is the default value. Once the gRPC interface is available, to use it, simply set the `PYEDB_USE_DOTNET` environment variable to 1.

```
# Set gRPC interface (future implementation)
import os

os.environ["PYEDB_USE_DOTNET"] = "1"

# Set DotNet interface (actual implementation)
import os
```

(continues on next page)

(continued from previous page)

```
os.environ["PYEDB_USE_DOTNET"] = "0"
```

1.5 Troubleshooting

This section first explains how to create PyEDB issues and post EDB discussions. It then describes how to troubleshoot some common issues related to installing and using PyEDB.

1.5.1 Issues and discussions

On the PyEDB Issues page, you can create issues to report bugs and request new features.

On the PyEDB Discussions page or the Discussions page on the Ansys Developer portal, you can post questions, share ideas, and get community feedback.

To reach the project support team, email pyansys.core@ansys.com.

1.5.2 Installation troubleshooting

Error installing Python or Conda

Some companies do not allow installation of a Python interpreter. In this case, you can use the Python interpreter available in the AEDT installation.

Note: Python 3.10 is available in AEDT 2023 R2 and later.

Here is the path to the Python 3.10 interpreter for the 2023 R1 installation:

```
"path\to\AnsysEM\v231\commonfiles\CPython\3_10\winx64\Release\python"
```

1.5.3 Error installing PyEDB using `pip`

According to Installing Python modules in the official Python documentation, pip, the preferred installer program, is included by default with Python binary installers. If you have issues using `pip`, check these areas for possible issues:

- **Proxy server**: If your company uses a proxy server, you may have to update proxy settings at the command line. For more information, see the Using a Proxy Server in the `pip` documentation.
- **Installation permission**: Make sure that you have write access to the directory where the Python interpreter is installed. The use of a virtual environment helps mitigate this issue by placing the Python interpreter and dependencies in a location that is owned by the user.
- **Firewall**: Some corporate firewalls may block `pip`. In this case, you must work with your IT administrator to enable it. The proxy server settings (described earlier) allow you to explicitly define the ports that `pip` is to use.

If downloads from PyPI, the Python Package Index, are not allowed, you can use a wheelhouse to install PyEDB. For more information, see install_pyedb_from_wheelhouse.

Run PyEDB with gRPC

gRPC is a modern open source, high-performance RPC (remote procedure call) framework that can run in any environment and supports client/server remote calls. Starting from 2024 R1, the EDB-Core API with a gRPC interface is available as Beta. During the Beta phase, both .NET and gRPC interfaces are set to be maintained. Once gRPC is officially released, it is planned for gRPC to become the default usage in PyEDB, with .NET being set up as an legacy.

Table 1: *gRPC compatibility:*

< 2022 R2	2024 R1	> 2024 R1
Only `Python.NET`	`Python.NET`: *Default*	gRPC: *Default*

CHAPTER

TWO

USER GUIDE

This section shows you how to use PyEDB. PyEDB loads EDB in memory, meaning non-graphically.

Load a layout   Learn how to load a layout (AEDB file) in EDB.

Load a layout file   Run layout queries   Learn how to run EDB layout queries.

Run layout queries   Build simulation projects   Learn how to build various types of simulation projects.

Build simulation projects   Create sources   Learn how to create and set up current sources and ports.

Create sources   Set up simulations   This section provides details about how to create a setup in HFSS or SIwave.

Set up simulations   Stackup   This section provides in-depth information on how to modify the edb stackup.

Work with a layer stackup   Padstacks   This section provides in-depth information on how to modify the pad-stacks definitions and instances.

Work with a padstack   Components   This section provides in-depth information on how to play with EDB components.

Working with a component   Parametrization   This section provides example on how to modify your layout.

use_design_variables

2.1 Load a layout file

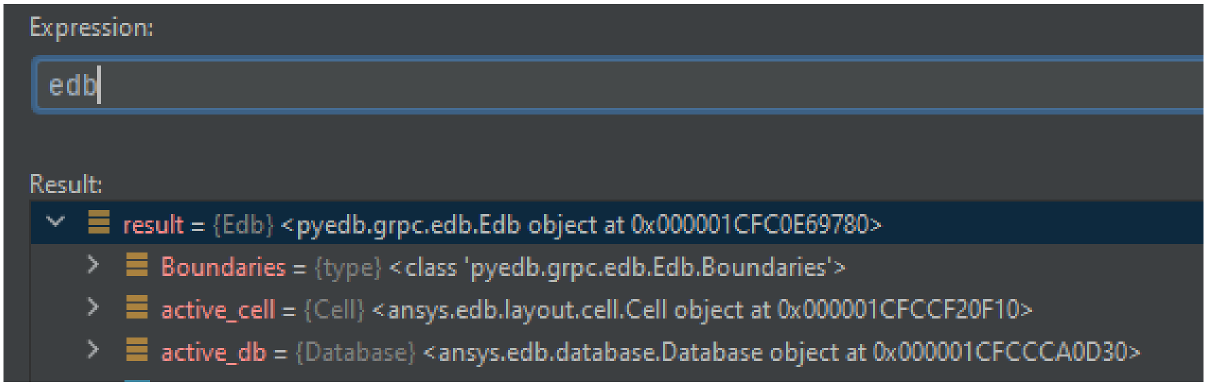
Load a layout file   Learn how to load an AEDB layout file in EDB.

Load a layout

2.1.1 Load a layout

Although you can use PyEDB to build an entire layout from scratch, most of the time you load an layout in an existing AEDB file. This page shows how to load a layout in EDB and start manipulating objects.

```python
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")
```



2.2 Run layout queries

Get layout statistics Learn how to run a query for getting layout statistics.

Get layout statistics Get layout size Learn how to run a query for getting the layout size.

Get layout size

2.2.1 Get layout statistics

PyEDB allows you to query a layout for statistics. This page shows how to perform these tasks:

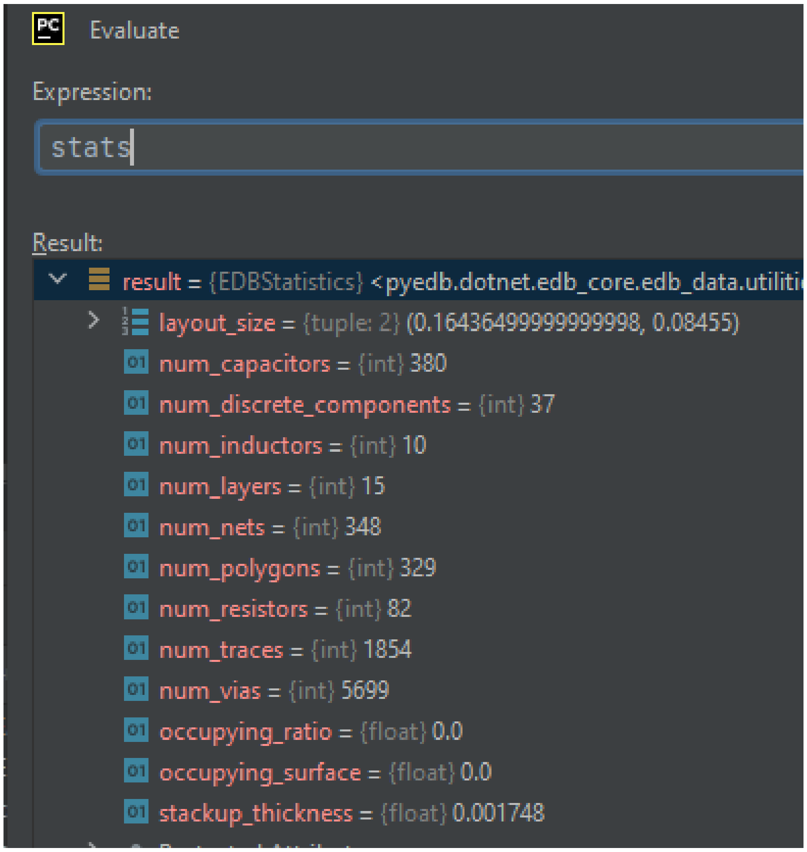
- Load a layout.
- Get statistics.
- Get nets and plot them in Matplotlib.
- Get all components and then get pins from components connected to a given net.

Load a layout

```python
# import EDB and load a layout
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")
```

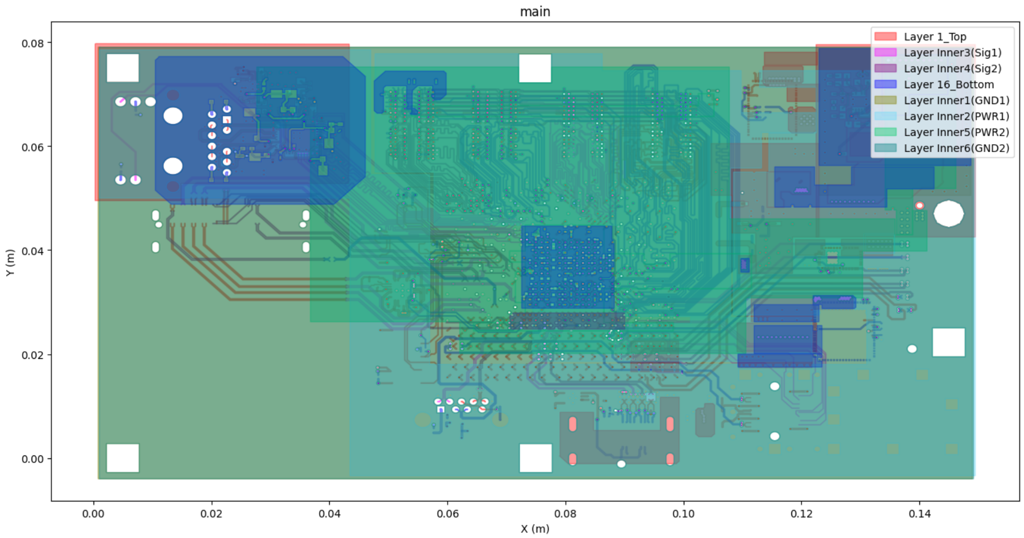
Get statistics

```python
stats = edbapp.get_statistics()
```



Get nets and plot them in Matplotlib

```python
# net list
edbapp.nets.netlist
# power nets
nets.power
# Plot nets in Matplotlib
edbapp.nets.plot(None)
```



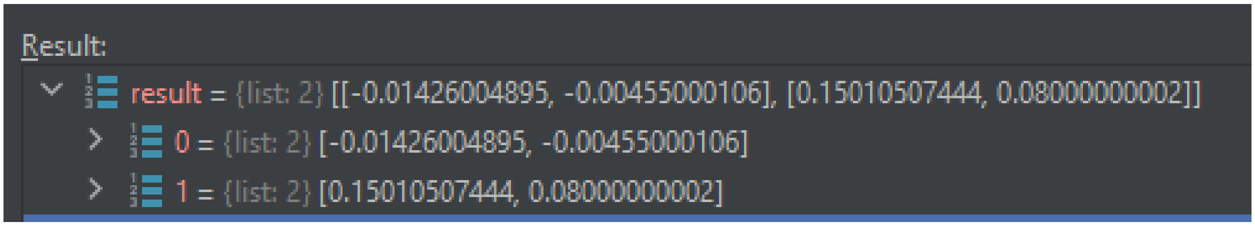
Get all components and then pins from components connected to a net

```
# Get all components
nets = edbapp.components.instances
# Get pins from components connected to a given net
u9_gnd_pins = [
    pin for pin in list(edbapp.components["U9"].pins.values()) if pin.net_name == "GND"
]
```

2.2.2 Get layout size

This tutorial shows how to retrieve the layout size by getting the bounding box.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

edbapp.get_bounding_box()
```



2.3 Build simulation projects

Clip a design Learn how to clip a design based on net selection to reduce computer resources and speed up the simulation.

Clip a design Build a signal integrity project Learn build an signal integrity project.

Build a signal integrity project

2.3.1 Clip a design

Most of the time, only a specific part of a layout needs to be simulated. Thus, you want to clip the design to reduce computer resources and speed up the simulation.

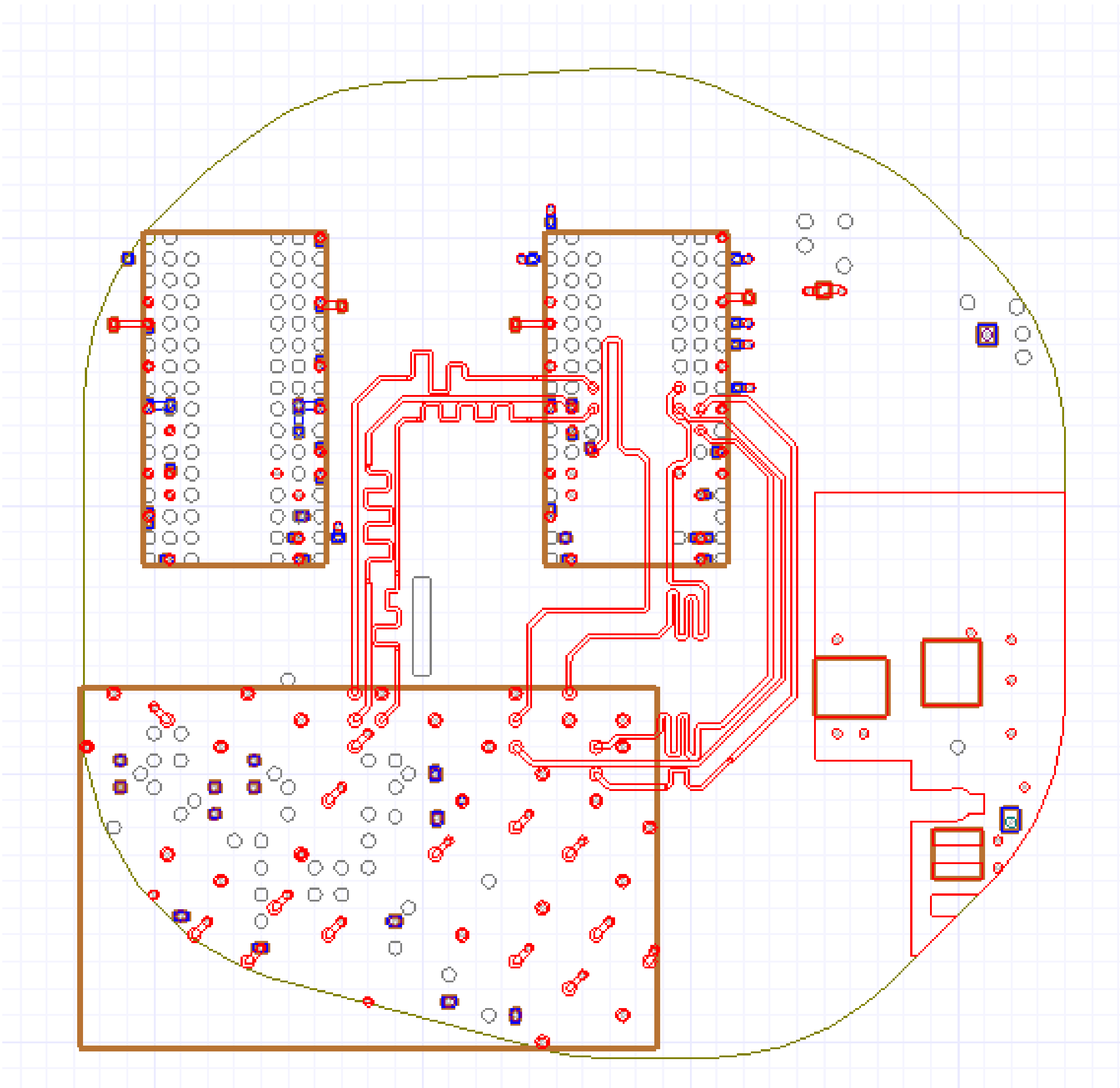
This page shows how to clip a design based on net selection.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads


# Ansys release version
ansys_version = "2024.1"

# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# select signal nets to evaluate the extent for clipping the layout
signal_nets = [
    "DDR4_DQ0",
    "DDR4_DQ1",
    "DDR4_DQ2",
    "DDR4_DQ3",
    "DDR4_DQ4",
    "DDR4_DQ5",
    "DDR4_DQ6",
    "DDR4_DQ7",
]
```

(continues on next page)

(continued from previous page)

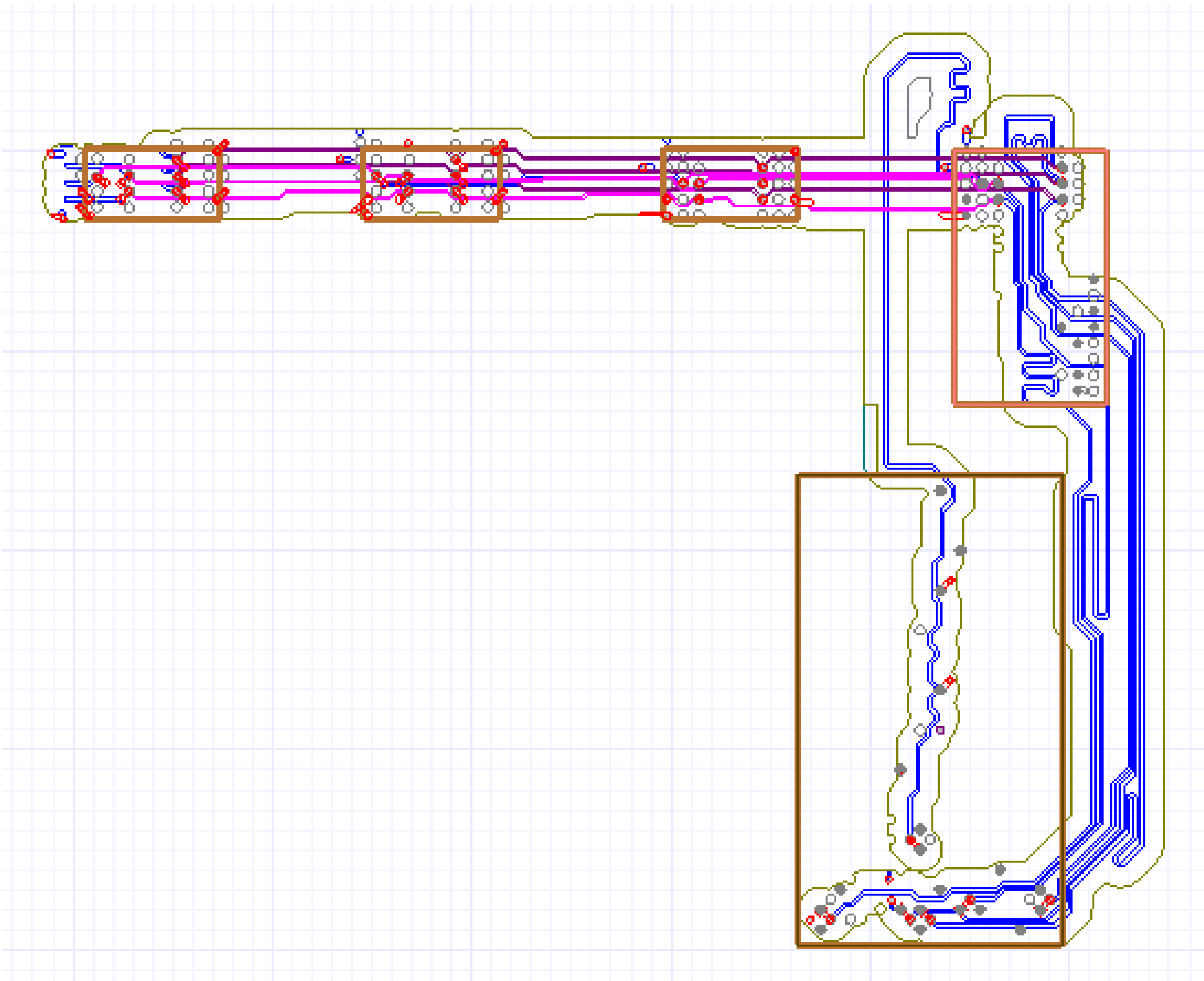
```
# At least one reference net must be included. Reference nets are included in the design␣
↪but clipped.
reference_nets = ["GND"]
# Define the expansion factor, which gives the distance for evaluating the cutout extent.
↪ This code defines a cutout.
expansion = 0.01  # 1cm in this case
# process cutout
edbapp.cutout(
    signal_list=signal_nets, reference_list=reference_nets, expansion_size=expansion
)
# save and close project
edbapp.save_edb()
edbapp.close_edb()
```



2.3.2 Build a signal integrity project

This page shows how to build an signal integrity project.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# Ansys release version
ansys_version = "2024.1"

# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

sim_setup = edbapp.new_simulation_configuration()
sim_setup.signal_nets = [
    "DDR4_A0",
    "DDR4_A1",
    "DDR4_A2",
    "DDR4_A3",
    "DDR4_A4",
    "DDR4_A5",
]
sim_setup.power_nets = ["GND"]
sim_setup.do_cutout_subdesign = True
sim_setup.components = ["U1", "U15"]
sim_setup.use_default_coax_port_radial_extension = False
sim_setup.cutout_subdesign_expansion = 0.001
sim_setup.start_freq = 0
sim_setup.stop_freq = 20e9
sim_setup.step_freq = 10e6
edbapp.build_simulation_project(sim_setup)
edbapp.close()
```



2.4 Create sources

Create a circuit port Learn how to retrieve pins and create a circuit port on a component.

Create a circuit port Create a coaxial port Learn how to create an HFSS coaxial port on a component.

Create a coaxial port Create current and voltage sources Learn how to create current and voltage sources on a component.

Create current and voltage sources Create an edge port Learn how to create an edge port on a polygon and trace.

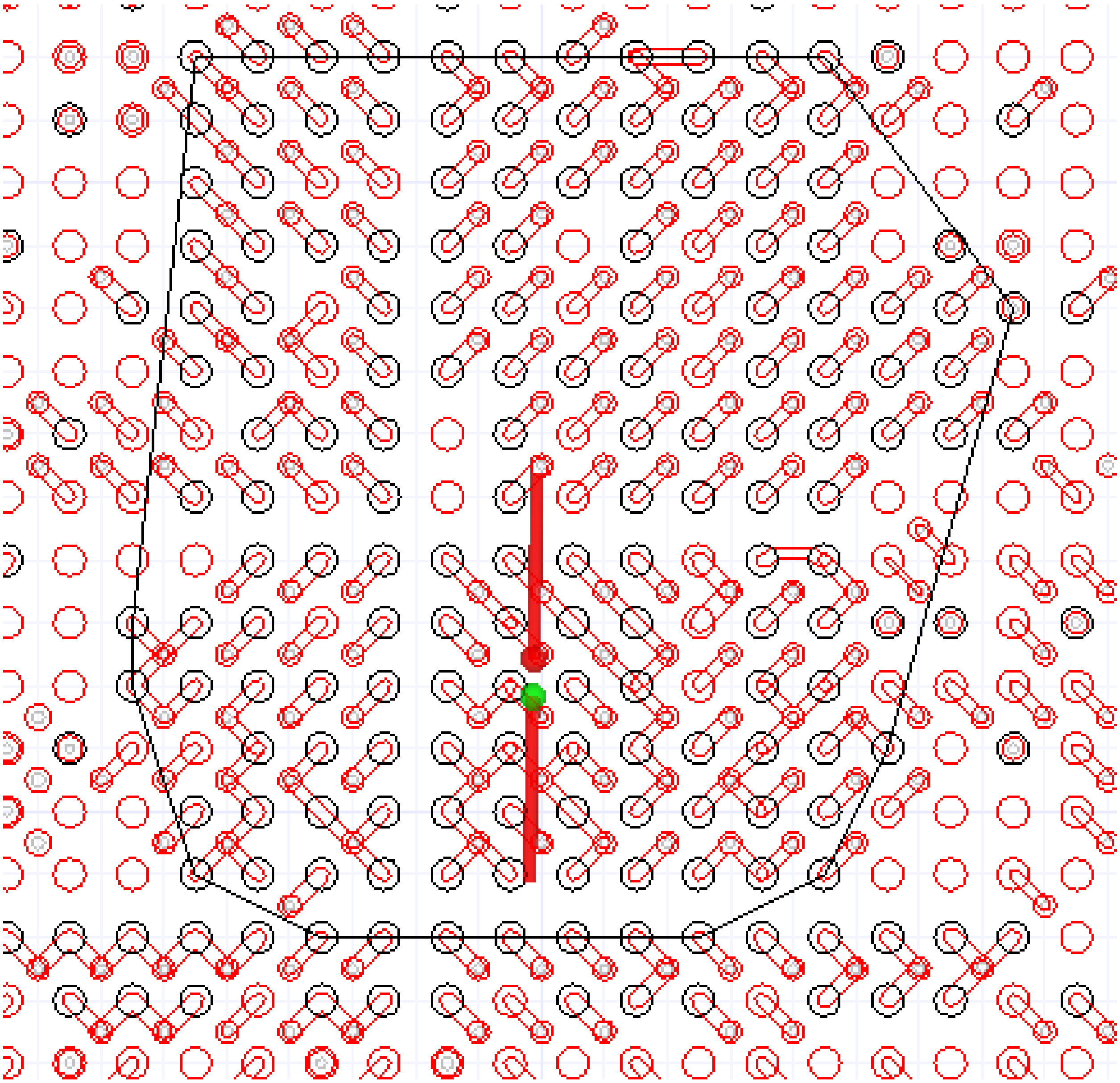
Create an edge port Create a port between a pin and layer Learn how to create a port between a pin and a layer.

Create port between a pin and layer

2.4.1 Create a circuit port

This page shows how to retrieve pins and create a circuit port on a component.

```python
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

edbapp.siwave.create_circuit_port_on_net("U1", "1V0", "U1", "GND", 50, "test")
edbapp.components.get_pin_from_component("U1")

# create pin groups
edbapp.siwave.create_pin_group_on_net("U1", "1V0", "PG_V1P0_S0")
# create port on pin group
edbapp.siwave.create_circuit_port_on_pin_group(
    "PG_V1P0_S0", "PinGroup_2", impedance=50, name="test_port"
)
# rename port with property setter
edbapp.excitations["test_port"].name = "test_rename"
# retrieve port
created_port = (port for port in list(edbapp.excitations) if port == "test_rename")
edbapp.save_edb()
edbapp.close_edb()
```



2.4.2 Create a coaxial port

This page shows how to create an HFSS coaxial port on a component.

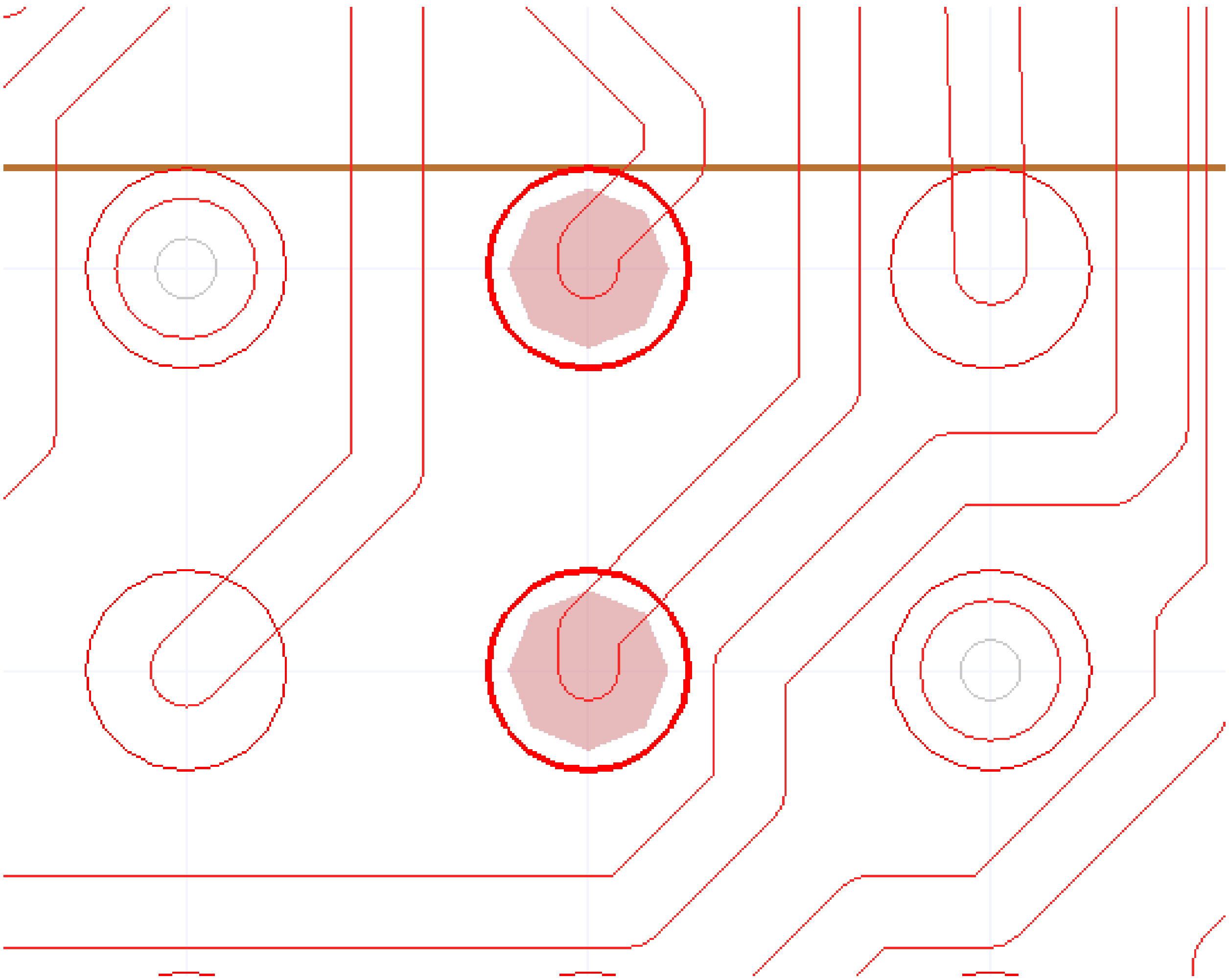
```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# Ansys release version
ansys_version = "2024.1"

```

(continues on next page)

(continued from previous page)

```
# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

edbapp.hfss.create_coax_port_on_component("U1", ["DDR4_DQS0_P", "DDR4_DQS0_N"])
edbapp.save_edb()
edbapp.close_edb()
```

The preceding code creates a coaxial port on nets `DDR4_DSQ0_P` and `DDR4_DSQ0_N` from component `U1`:



2.4.3 Create current and voltage sources

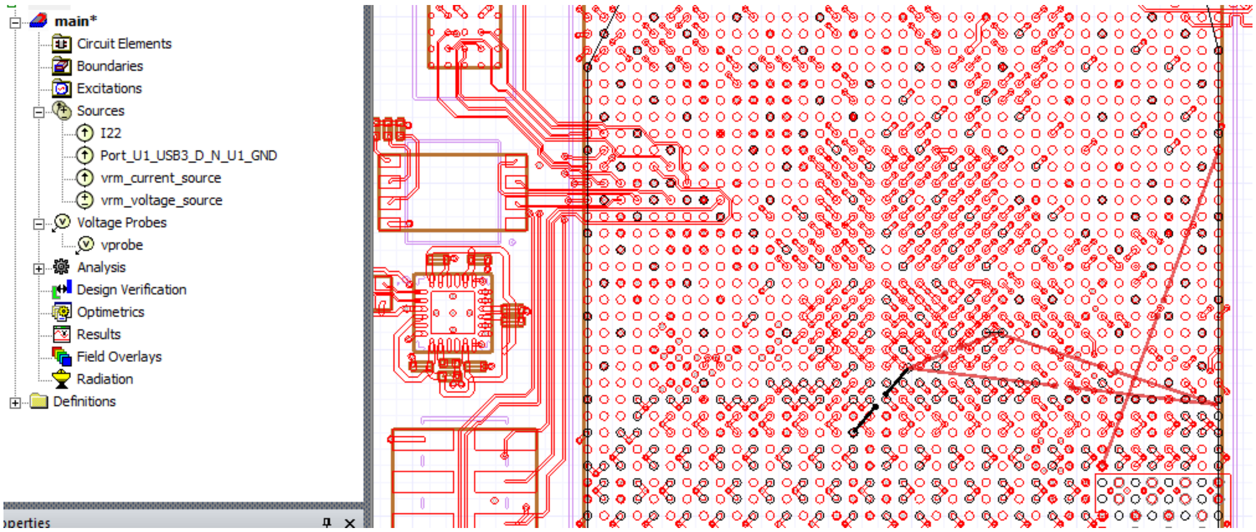
This page shows how to create current and voltage sources on a component.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# create simple current source on ``U1`` component between ``USB3_D_N`` and ``GND`` nets
edbapp.siwave.create_current_source_on_net("U1", "USB3_D_N", "U1", "GND", 0.1, 0) != ""

# retrieve pins from ``U1`` component
pins = edbapp.components.get_pin_from_component("U1")

# create current source on specific pins
edbapp.siwave.create_current_source_on_pin(pins[301], pins[10], 0.1, 0, "I22")

# create pin group on ``GND`` net from ``U1`` component
edbapp.siwave.create_pin_group_on_net(
    reference_designator="U1", net_name="GND", group_name="gnd"
)

# creat pin group on specific pins
edbapp.siwave.create_pin_group(
    reference_designator="U1", pin_numbers=["A27", "A28"], group_name="vrm_pos"
)

# create current source on pin group
edbapp.siwave.create_current_source_on_pin_group(
    pos_pin_group_name="vrm_pos", neg_pin_group_name="gnd", name="vrm_current_source"
)

#  create voltage source
edbapp.siwave.create_pin_group(
    reference_designator="U1", pin_numbers=["R23", "P23"], group_name="sink_pos"
)
edbapp.siwave.create_voltage_source_on_pin_group(
    "sink_pos", "gnd", name="vrm_voltage_source"
)

# create voltage probe
edbapp.siwave.create_pin_group(
    reference_designator="U1", pin_numbers=["A27", "A28"], group_name="vp_pos"
)
edbapp.siwave.create_pin_group(
    reference_designator="U1", pin_numbers=["R23", "P23"], group_name="vp_neg"
)
```

(continues on next page)

(continued from previous page)

```
edbapp.siwave.create_voltage_probe_on_pin_group("vprobe", "vp_pos", "vp_neg")
edbapp.save_edb()
edbapp.close_edb()
```



2.4.4 Create an edge port

This page shows how to create an edge port on a polygon and trace.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# Ansys release version
ansys_version = "2024.1"

# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/edb_edge_ports.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# retrieve polygon list
poly_list = [
    poly for poly in edbapp.layout.primitives if int(poly.GetPrimitiveType()) == 2
]

# select specific polygons
port_poly = [poly for poly in poly_list if poly.GetId() == 17][0]
ref_poly = [poly for poly in poly_list if poly.GetId() == 19][0]
```

(continues on next page)

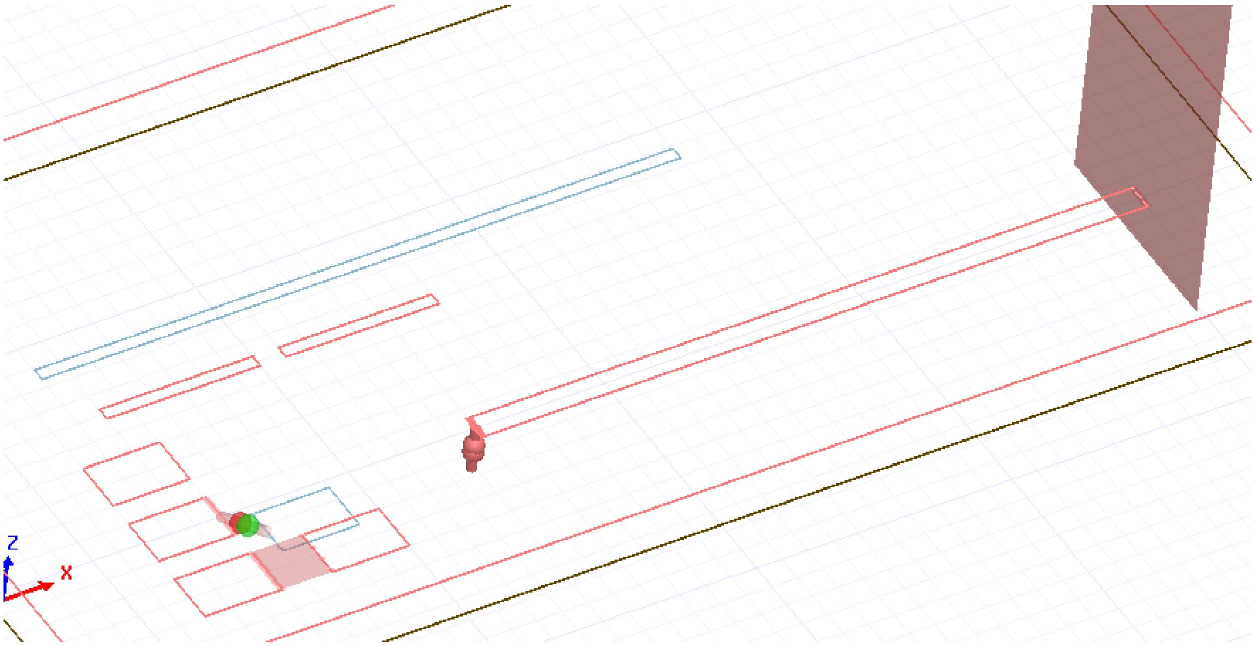
(continued from previous page)

```
#  define port location
port_location = [-65e-3, -13e-3]
ref_location = [-63e-3, -13e-3]

# create edge port
edbapp.hfss.create_edge_port_on_polygon(
    polygon=port_poly,
    reference_polygon=ref_poly,
    terminal_point=port_location,
    reference_point=ref_location,
)

# select specific polygon
port_poly = [poly for poly in poly_list if poly.GetId() == 23][0]
ref_poly = [poly for poly in poly_list if poly.GetId() == 22][0]

# define port location
port_location = [-65e-3, -10e-3]
ref_location = [-65e-3, -10e-3]

# create port on polygon
edbapp.hfss.create_edge_port_on_polygon(
    polygon=port_poly,
    reference_polygon=ref_poly,
    terminal_point=port_location,
    reference_point=ref_location,
)

# select polygon
port_poly = [poly for poly in poly_list if poly.GetId() == 25][0]

# define port location
port_location = [-65e-3, -7e-3]

# create edge port with defining reference layer
edbapp.hfss.create_edge_port_on_polygon(
    polygon=port_poly, terminal_point=port_location, reference_layer="gnd"
)

# create trace
sig = edbapp.modeler.create_trace(
    [["-55mm", "-10mm"], ["-29mm", "-10mm"]], "TOP", "1mm", "SIG", "Flat", "Flat"
)

# create wave port at the end of the trace
sig.create_edge_port("pcb_port_1", "end", "Wave", None, 8, 8)

# create gap port at the beginning of the trace
sig.create_edge_port("pcb_port_2", "start", "gap")

# retrieve existing port
gap_port = edbapp.ports["pcb_port_2"]
```

(continues on next page)

(continued from previous page)

```
# rename port
gap_port.name = "gap_port"

# change gap to circuit port
gap_port.is_circuit_port = True

edbapp.save_edb()
edbapp.close_edb()
```



2.4.5 Create port between a pin and layer

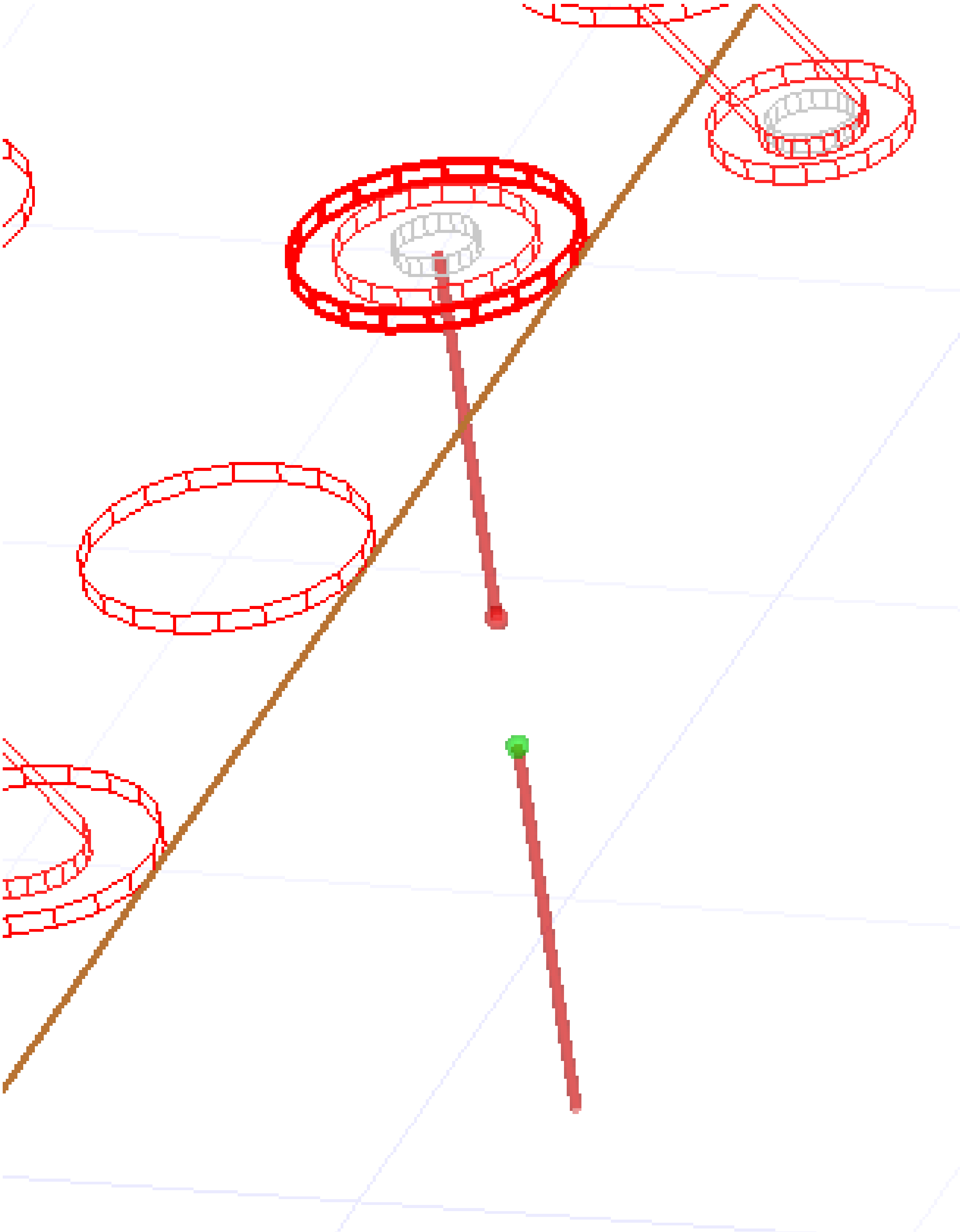
This page shows how to create a port between a pin and a layer.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# Ansys release version
ansys_version = "2024.1"

# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")
```

(continues on next page)

(continued from previous page)

```
edbapp.siwave.create_port_between_pin_and_layer(
    component_name="U1", pins_name="A27", layer_name="16_Bottom", reference_net="GND"
)
U7 = edbapp.components["U7"]
_, pin_group = edbapp.siwave.create_pin_group_on_net(
    reference_designator="U7", net_name="GND", group_name="U7_GND"
)
U7.pins["F7"].create_port(reference=pin_group)
edbapp.save_edb()
edbapp.close_edb()
```



2.5 Set up simulations

Set up a SIwave analysis   Learn how to create and set up a SIwave SWZ analysis.

Set up a SIwave analysis   Set up an HFSS simulation   Learn how to create and set up an HFSS simulation.

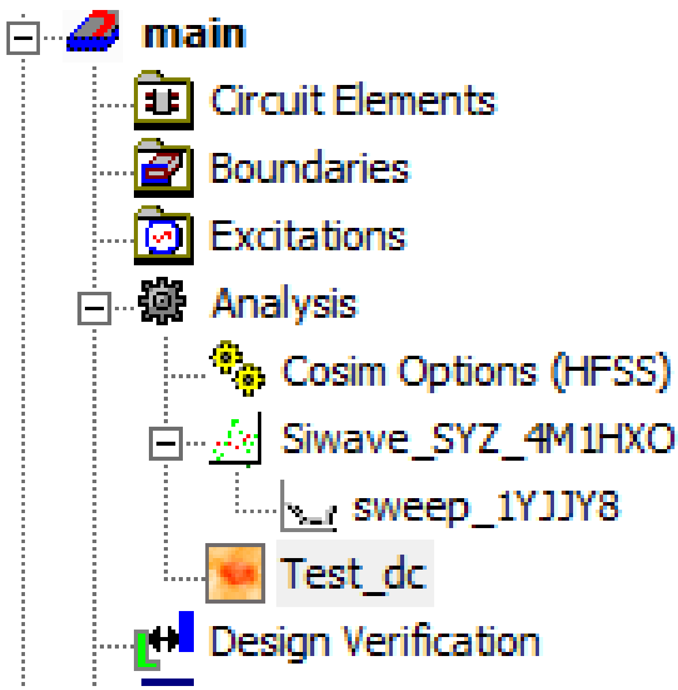
Set up an HFSS simulation   Define an HFSS extent   Learn how to define an HFSS extent.

Define an HFSS extent

2.5.1 Set up a SIwave analysis

This page shows how to create and set up a SIwave SYZ analysis.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# Add SIwave SYZ analysis
edbapp.siwave.add_siwave_syz_analysis(
    start_freq="=GHz", stop_freq="10GHz", step_freq="10MHz"
)

# Add DC analysis
edbapp.siwave.add_siwave_dc_analysis(name="Test_dc")
edbapp.save()
edbapp.close()
```



2.5.2 Set up an HFSS simulation

This page shows how to set up an HFSS simulation.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# Ansys release version
ansys_version = "2024.1"

# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# create HFSS simulation setup
setup1 = edbapp.create_hfss_setup("setup1")

# set solution as single frequenvcy
setup1.set_solution_single_frequency()

# set multi-frequencies solution
setup1.set_solution_multi_frequencies()

# set broadband solution
```

(continues on next page)

(continued from previous page)

```
setup1.set_solution_broadband(low_frequency="1GHz", high_frequency="10GHz")

# enable low-frequency accuracy
setup1.hfss_solver_settings.enhanced_low_freq_accuracy = True

# set solution basis order
setup1.hfss_solver_settings.order_basis = "first"

# set relative residual
setup1.hfss_solver_settings.relative_residual = 0.0002

# enable shell elements usage
setup1.hfss_solver_settings.use_shell_elements = True

# retrieve HFSS solver settings
hfss_solver_settings = edbapp.setups["setup1"].hfss_solver_settings

# add adaptive settings
setup1.adaptive_settings.add_adaptive_frequency_data("5GHz", 8, "0.01")

# add broadband adaptive settings
setup1.adaptive_settings.adapt_type = "kBroadband"

# specify maximum number of adaptive passes
setup1.adaptive_settings.max_refine_per_pass = 20

# specify minimum number of adaptive passes
setup1.adaptive_settings.min_passes = 2

# enable save fields
setup1.adaptive_settings.save_fields = True

# enable save radiate fields only
setup1.adaptive_settings.save_rad_field_only = True

# enable defeature based on absolute length
setup1.defeature_settings.defeature_abs_length = "1um"

# enable defeature based on aspect ratio
setup1.defeature_settings.defeature_ratio = 1e-5

# set healing options
setup1.defeature_settings.healing_option = 0

# set model type
setup1.defeature_settings.model_type = 1

# enable removal of floating geometries
setup1.defeature_settings.remove_floating_geometry = True

# void defeaturing criteria
setup1.defeature_settings.small_void_area = 0.1
```

(continues on next page)

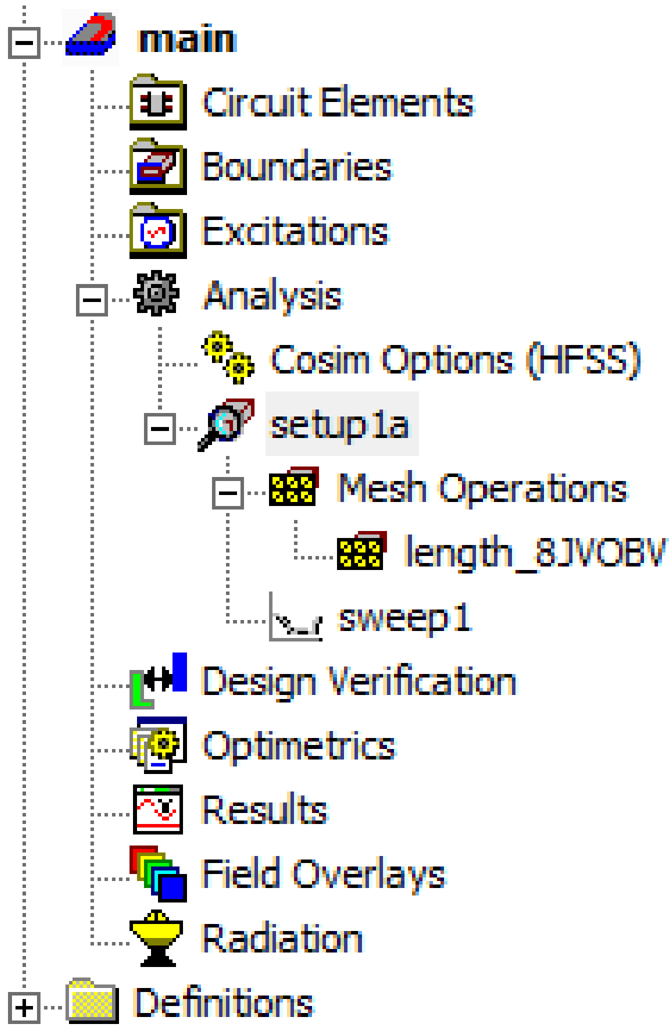
(continued from previous page)

```
# enable polygon union
setup1.defeature_settings.union_polygons = False

#  enable defeaturing
setup1.defeature_settings.use_defeature = False

# enable absolute length defeaturing
setup1.defeature_settings.use_defeature_abs_length = True

via_settings = setup1.via_settings
via_settings.via_density = 1
via_settings.via_material = "pec"
via_settings.via_num_sides = 8
via_settings.via_style = "kNum25DViaStyle"

# specify advanced mesh settings
advanced_mesh_settings = setup1.advanced_mesh_settings
advanced_mesh_settings.layer_snap_tol = "1e-6"
advanced_mesh_settings.mesh_display_attributes = "#0000001"
advanced_mesh_settings.replace_3d_triangles = False

# specify curve approximation settings
curve_approx_settings = setup1.curve_approx_settings
curve_approx_settings.arc_angle = "15deg"
curve_approx_settings.arc_to_chord_error = "0.1"
curve_approx_settings.max_arc_points = 12
curve_approx_settings.start_azimuth = "1"
curve_approx_settings.use_arc_to_chord_error = True

# specify DC settings
dcr_settings = setup1.dcr_settings
dcr_settings.conduction_max_passes = 11
dcr_settings.conduction_min_converged_passes = 2
dcr_settings.conduction_min_passes = 2
dcr_settings.conduction_per_error = 2.0
dcr_settings.conduction_per_refine = 33.0

# specify port settings
hfss_port_settings = setup1.hfss_port_settings
hfss_port_settings.max_delta_z0 = 0.5
hfss_port_settings.max_triangles_wave_port = 1000
hfss_port_settings.min_triangles_wave_port = 200
hfss_port_settings.set_triangles_wave_port = True

# add frequency sweep
setup1.add_frequency_sweep(
    "sweep1",
    frequency_sweep=[
        ["linear count", "0", "1kHz", 1],
        ["log scale", "1kHz", "0.1GHz", 10],
        ["linear scale", "0.1GHz", "10GHz", "0.1GHz"],
```

(continues on next page)

(continued from previous page)

```
    ],
)
sweep1 = setup1.frequency_sweeps["sweep1"]
sweep1.adaptive_sampling = True

# change setup name
edbapp.setups["setup1"].name = "setup1a"

# add length-based mesh operation
mop = edbapp.setups["setup1a"].add_length_mesh_operation(
    {"GND": ["1_Top", "16_Bottom"]}, "m1"
)
mop.name = "m2"
mop.max_elements = 2000
mop.restrict_max_elements = False
mop.restrict_length = False
mop.max_length = "2mm"

# add skin-depth mesh operation
mop = edbapp.setups["setup1a"].add_skin_depth_mesh_operation(
    {"GND": ["1_Top", "16_Bottom"]}
)
mop.skin_depth = "5um"
mop.surface_triangle_length = "2mm"
mop.number_of_layer_elements = "3"
edbapp.save()
edbapp.close()
```



2.5.3 Define an HFSS extent

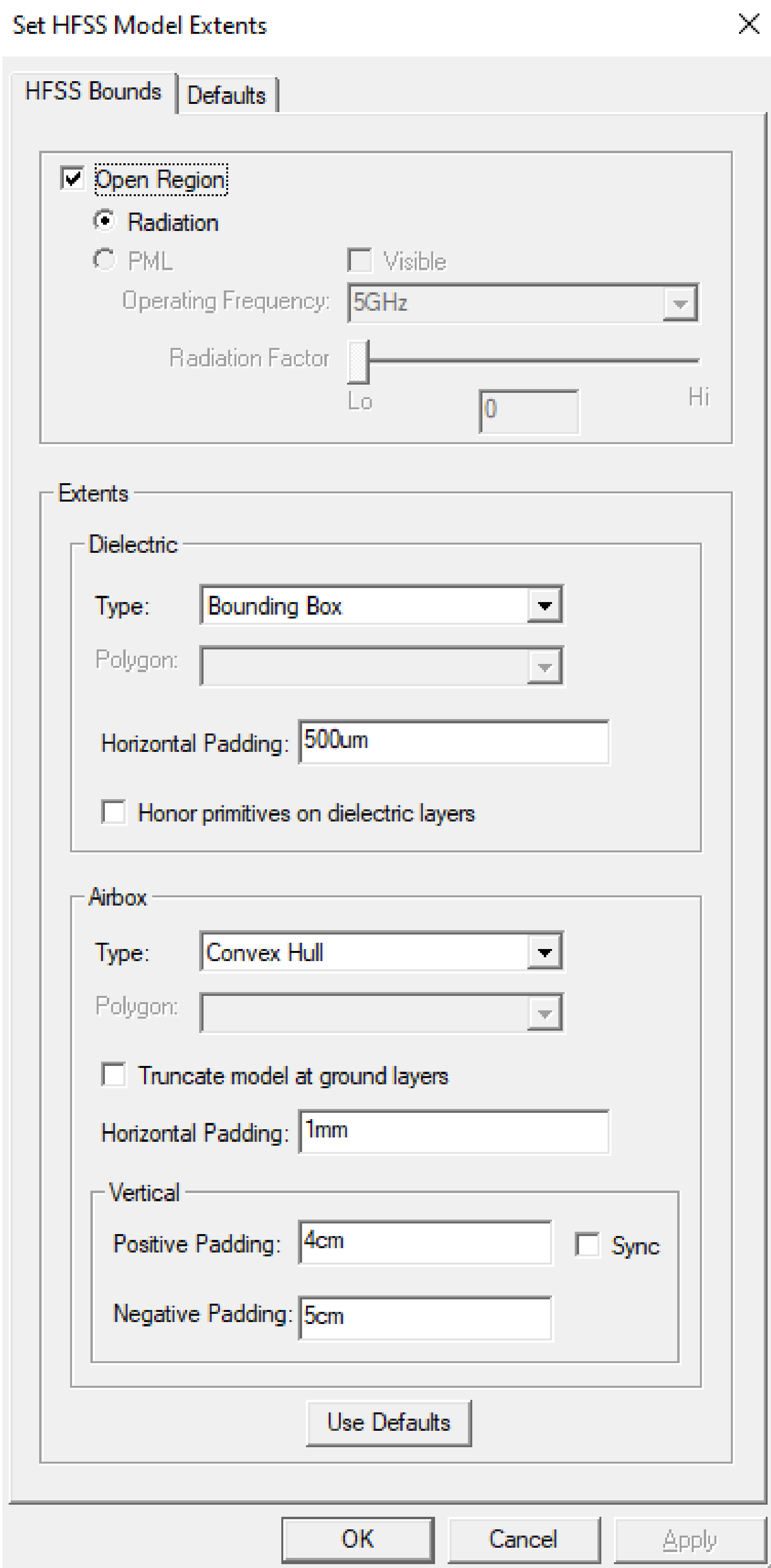
This page shows how to define an HFSS extent using the `SimulationConfiguration` class.

```
from pyedb import Edb

# create EDB
edb = Edb()

# add stackup layers
edb.stackup.add_layer(layer_name="GND", fillMaterial="AIR", thickness="30um")
edb.stackup.add_layer(layer_name="FR4", base_layer="gnd", thickness="250um")
edb.stackup.add_layer(layer_name="SIGNAL", base_layer="FR4", thickness="30um")

# create trace
```

(continues on next page)

(continued from previous page)

```
edb.modeler.create_trace(
    layer_name="SIGNAL", width=0.02, net_name="net1", path_list=[[-1e3, 0, 1e-3, 0]]
)

# create primitive rectangle
edb.modeler.create_rectangle(
    layer_name="GND",
    representation_type="CenterWidthHeight",
    center_point=["0mm", "0mm"],
    width="4mm",
    height="4mm",
    net_name="GND",
)

# create ``SimulationConfiguration`` object
sim_setup = edb.new_simulation_configuration()


#  define air box settings
sim_setup.use_dielectric_extent_multiple = False
sim_setup.use_airbox_horizontal_extent_multiple = False
sim_setup.use_airbox_negative_vertical_extent_multiple = False
sim_setup.use_airbox_positive_vertical_extent_multiple = False
sim_setup.dielectric_extent = 0.0005
sim_setup.airbox_horizontal_extent = 0.001
sim_setup.airbox_negative_vertical_extent = 0.05
sim_setup.airbox_positive_vertical_extent = 0.04

# disable frequency sweep creation
sim_setup.add_frequency_sweep = False

# include only selected nets
sim_setup.include_only_selected_nets = True

# disable cutout
sim_setup.do_cutout_subdesign = False

# disable port generation
sim_setup.generate_excitations = False

# build project
edb.build_simulation_project(sim_setup)
edb.save()
edb.close()
```


Set HFSS Model Extents
HFSS Bounds
Defaults
Open Region
Radiation
PML
Visible
Operating Frequency:
5GHz
Radiation Factor
Lo
0
Hi
Extents
Dielectric
Type:
Bounding Box
Polygon:
Horizontal Padding:
500um
Honor primitives on dielectric layers
Airbox
Type:
Convex Hull
Polygon:
Truncate model at ground layers
Horizontal Padding:
1mm
Vertical
Positive Padding:
4cm
Sync
Negative Padding:
5cm
Use Defaults
OK
Cancel
Apply

2.6 Work with a layer stackup

Edit a layer in a layer stackup   Learn how to edit a layer in the current layer stackup.

Edit a layer stackup in a layout stackup

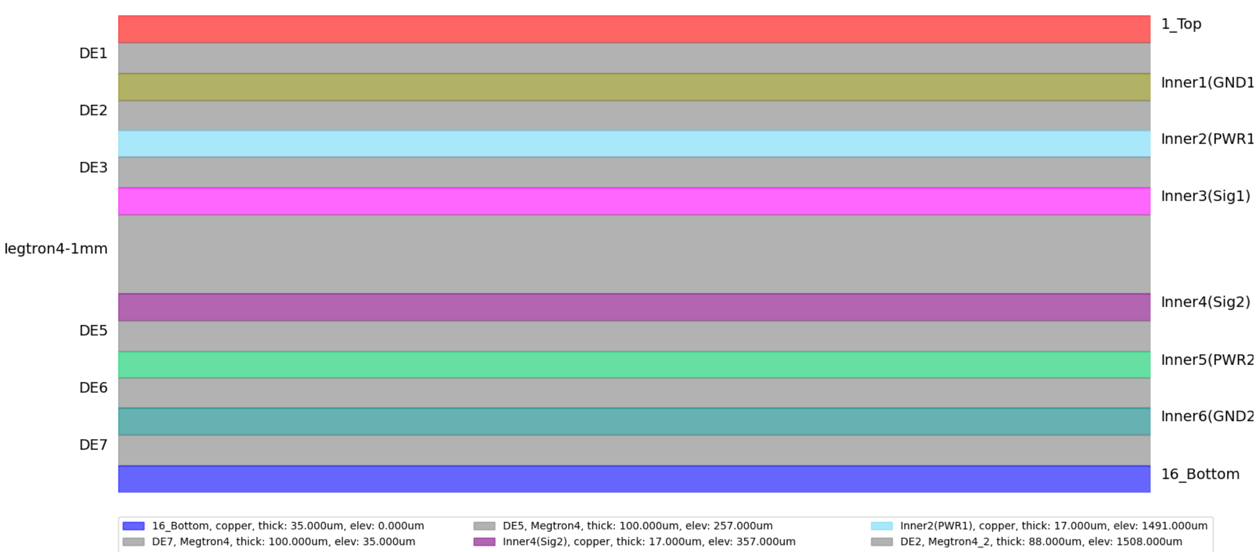
Add a layer in a layout stackup   Learn how to add a layer in the current layout stackup.

Add a layer in a layout stackup

2.6.1 Edit a layer stackup in a layout stackup

This page shows how edit a layer in the current layer stackup.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# plot layer stackup in Matplotlib
edbapp.stackup.plot()
# change top layer thickness to 40um
edbapp.stackup.signal_layers["1_Top"].thickness = 40e-6
print(f"Top layer thickness update {edbapp.stackup.signal_layers['1_Top'].thickness}")
edbapp.save()
edbapp.close()
```




```
# retrieve list of signal layer names
signal_layers = list(edbapp.stackup.signal_layers.keys())
```

(continues on next page)

(continued from previous page)

```
# select top layer
top_layer = edbapp.stackup.signal_layers[signal_layers[0]]

# get stackup total thickness
layout_stats = edbapp.get_statistics()
layout_stats.stackup_thickness

# set thickness of all signal layers to ``20um``
for layer_name, layer in edbapp.stackup.signal_layers.items():
    layer.thickness = "20um"

edbapp.materials.add_material(
    name="MyMaterial", permittivity=4.35, dielectric_loss_tangent=2e-4
)
edbapp.materials.add_material(name="MyMetal", conductivity=1e7)
for layer in list(edbapp.stackup.dielectric_layers.values()):
    layer.material = "MyMaterial"
for layer in list(edbapp.stackup.signal_layers.values()):
    layer.material = "MyMetal"
edbapp.materials.add_material(
    name="SolderMask", permittivity=3.8, dielectric_loss_tangent=1e-3
)
edbapp.stackup.add_layer(
    layer_name="Solder_mask",
    base_layer="1_Top",
    thickness="200um",
    material="SolderMask",
)
```

2.6.2 Add a layer in a layout stackup

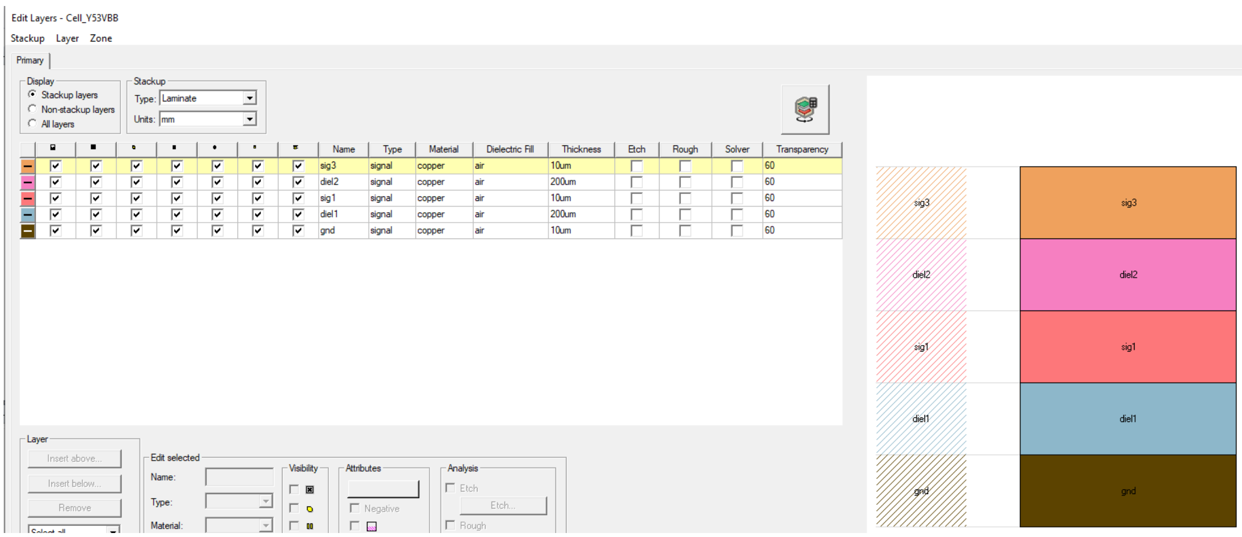
This page shows how to add a layer in the current layer stackup.

```
from pyedb import Edb

edb = Edb(edbversion=desktop_version)
edb.stackup.add_layer(layer_name="gnd", fillMaterial="AIR", thickness="10um")
edb.stackup.add_layer(
    layer_name="diel1", fillMaterial="AIR", thickness="200um", base_layer="gnd"
)
edb.stackup.add_layer(
    layer_name="sig1", fillMaterial="AIR", thickness="10um", base_layer="diel1"
)
edb.stackup.add_layer(
    layer_name="diel2", fillMaterial="AIR", thickness="200um", base_layer="sig1"
)
edb.stackup.add_layer(
```

(continues on next page)

(continued from previous page)

```
    layer_name="sig3", fillMaterial="AIR", thickness="10um", base_layer="diel2"
)
edb.close()
```



2.7 Work with a padstack

Edit a padstack definition   Learn how to edit a padstack definition, setting all anti-pad values to a fixed value.

Edit a padstack definition   Create a padstack instance   Learn how to create a padstack instance.

Create a padstack instance

2.7.1 Edit a padstack definition

This page shows how to edit a padstack definition, setting all anti-pad values to a fixed value.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# Ansys release version
ansys_version = "2024.1"

# download and copy the layout file from examples
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

# load EDB
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# sets all anti-pads value to zero
```

(continues on next page)

(continued from previous page)

```
edbapp.padstacks.set_all_antipad_value(0.0)
edbapp.close()
```

2.7.2 Create a padstack instance

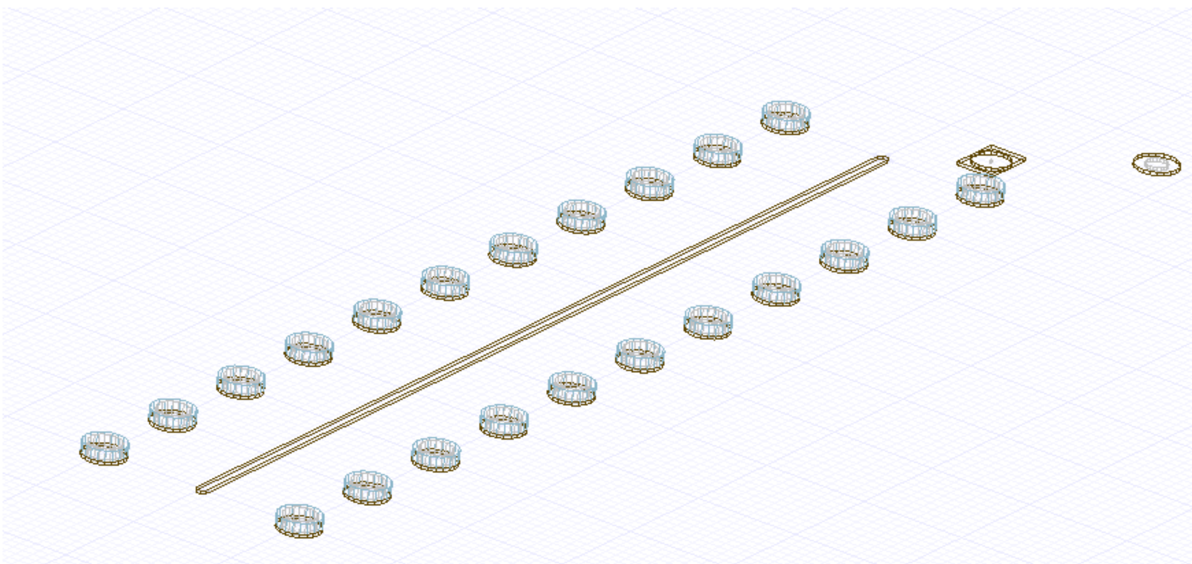
This page shows how to create a padstack instance.

```
from pyedb import Edb


edb = Edb(edbversion=desktop_version)
edb.stackup.add_layer(layer_name="1_Top", fillMaterial="AIR", thickness="30um")
edb.stackup.add_layer(
    layer_name="contact", fillMaterial="AIR", thickness="100um", base_layer="1_Top"
)

edb.padstacks.create(
    pad_shape="Rectangle",
    padstackname="pad",
    x_size="350um",
    y_size="500um",
    holediam=0,
)
pad_instance1 = edb.padstacks.place(
    position=["-0.65mm", "-0.665mm"], definition_name="pad"
)
pad_instance1.start_layer = "1_Top"
pad_instance1.stop_layer = "1_Top"

edb.padstacks.create(
    pad_shape="Circle", padstackname="pad2", paddiam="350um", holediam="15um"
)
pad_instance2 = edb.padstacks.place(
    position=["-0.65mm", "-0.665mm"], definition_name="pad2"
)

pad_instance2.start_layer = "1_Top"
pad_instance2.stop_layer = "1_Top"

edb.padstacks.create(
    pad_shape="Circle",
    padstackname="test2",
    paddiam="400um",
    holediam="200um",
    antipad_shape="Rectangle",
    anti_pad_x_size="700um",
    anti_pad_y_size="800um",
    start_layer="1_Top",
```

(continues on next page)

(continued from previous page)

```
    stop_layer="1_Top",
)

pad_instance3 = edb.padstacks.place(
    position=["-1.65mm", "-1.665mm"], definition_name="test2"
)
pad_instance3.dcir_equipotential_region = True
pad_instance3.dcir_equipotential_region = False

trace = edb.modeler.create_trace(
    [[0, 0], [0, 10e-3]], "1_Top", "0.1mm", "trace_with_via_fence"
)
edb.padstacks.create_padstack("via_0")
trace.create_via_fence("1mm", "1mm", "via_0")
edb.save()
edb.close()
```



2.8 Working with a component

Create a resistor boundary on pins   Learn how to create a resistor boundary on pins.

Create a resistor boundary on pins   Create an RLC component between pins   Learn how to create an RLC component between pins.

Create an RLC component between pins   Create an RLC boundary on pins   Learn how to create an RLC boundary on pins.

Create an RLC boundary on pins

2.8.1 Create a resistor boundary on pins

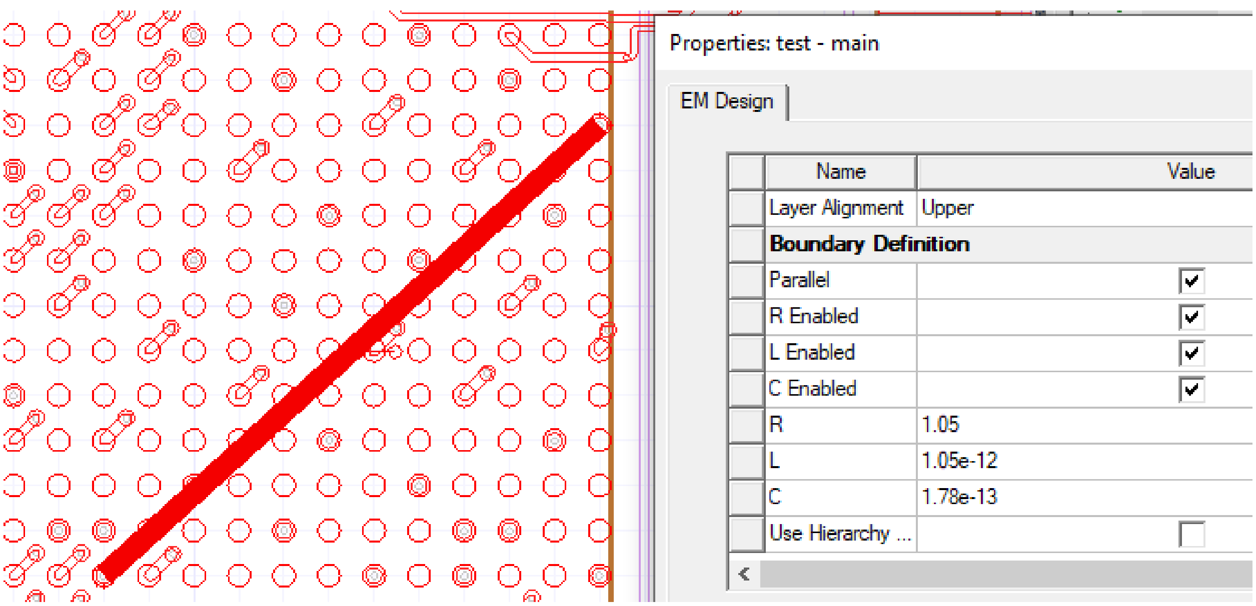
This page shows how to create a resistor boundary on pins:

```python
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

pins = edbapp.components.get_pin_from_component("U1")
resistor = edbapp.siwave.create_resistor_on_pin(pins[302], pins[10], 40, "RST4000")
edbapp.save_edb()
edbapp.close_edb()
```

2.8.2 Create an RLC component between pins

This page shows how to create an RLC component between pins:

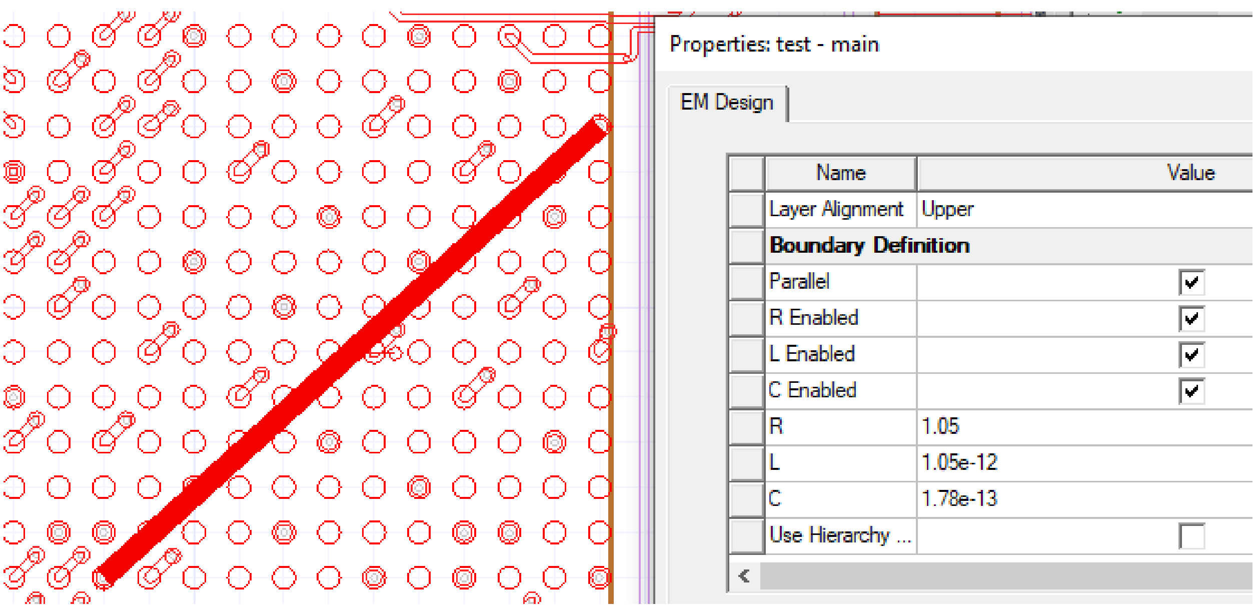
```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# download and open EDB project
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# retrieving pins from component U1 and net 1V0
pins = edbapp.components.get_pin_from_component("U1", "1V0")

# retrieving reference pins from component U1 and net GND
ref_pins = edbapp.components.get_pin_from_component("U1", "GND")

# create RLC network between 2 pins
edbapp.components.create(
    [pins[0], ref_pins[0]], "test_0rlc", r_value=1.67, l_value=1e-13, c_value=1e-11
)
edbapp.save_edb()
edbapp.close_edb()
```



2.8.3 Create an RLC boundary on pins

This page shows how to create an RLC boundary on pins.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads

# download and open EDB project
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")

# retrieve pins from ``U1`` component and ``1V0`` net
pins = edbapp.components.get_pin_from_component("U1", "1V0")

# reference pins
ref_pins = edbapp.components.get_pin_from_component("U1", "GND")

# create rlc boundary
edbapp.hfss.create_rlc_boundary_on_pins(
    pins[0], ref_pins[0], rvalue=1.05, lvalue=1.05e-12, cvalue=1.78e-13
)

# close EDB
edbapp.save_edb()
edbapp.close_edb()
```



CHAPTER

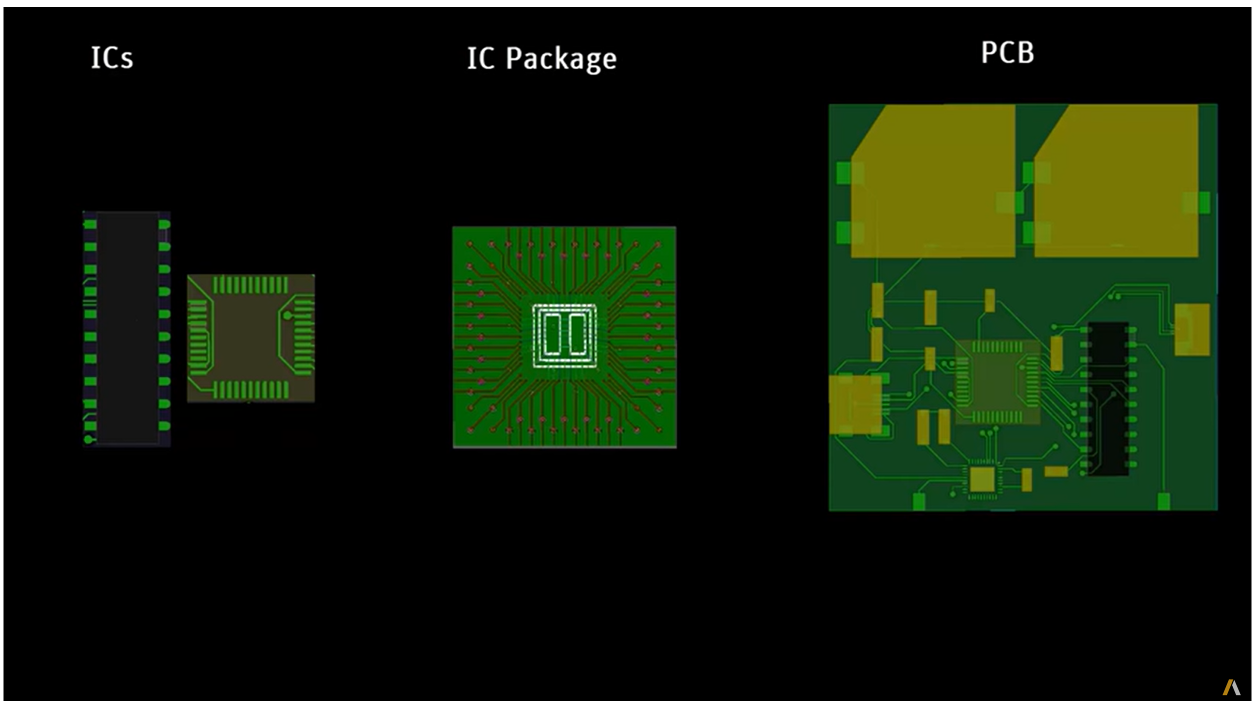
THREE

API REFERENCE

This section describes EDB functions, classes, and methods for EDB apps and modules. Use the search feature or click links to view API documentation.

The PyEDB API includes classes for apps and modules. You must initialize the `Edb` class to get access to all modules and methods. All other classes and methods are inherited into the `Edb` class.

If EDB is launched within the `HfssdLayout` class, EDB is accessible in read-only mode.

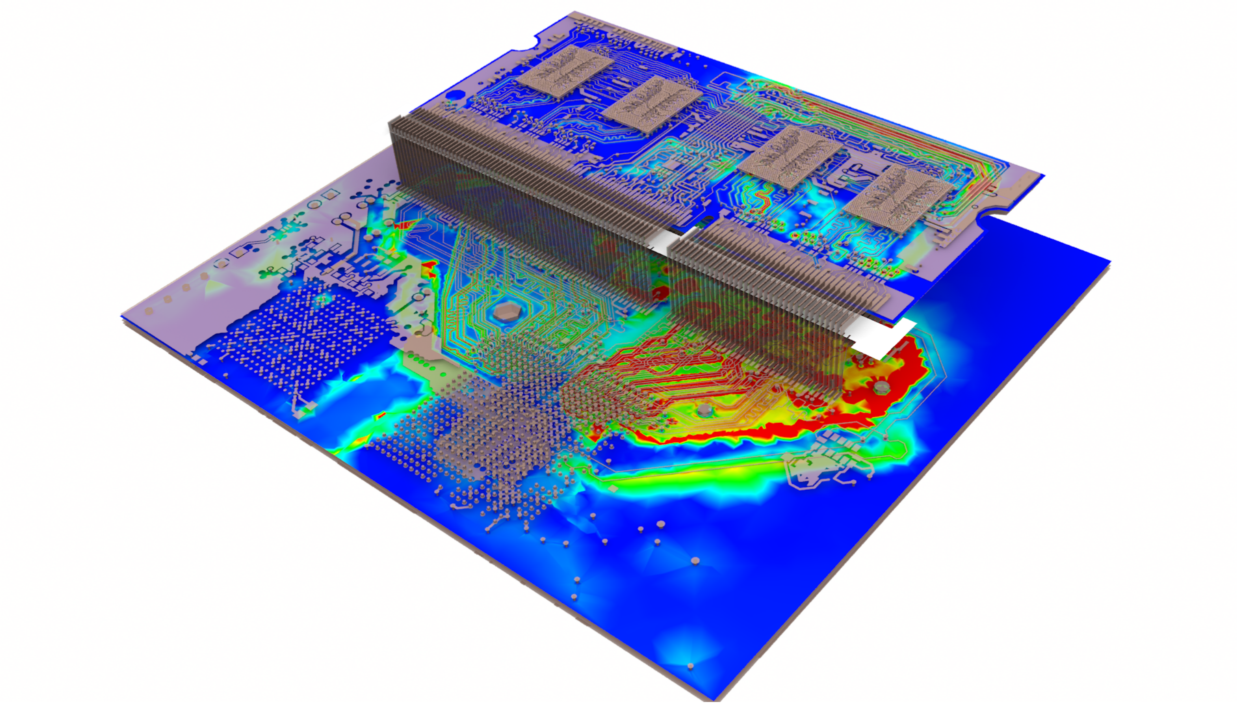


Example

```python
from pyedb import Edb

edb = Edb("my_project.aedb", edbversion="2023.1")
edb.core_components.components["R1"].r_value = 40
edb.close_edb()
```

3.1 EDB manager

An AEDB database is a folder that contains the database representing any part of a PCB. It can be opened and edited using the `Edb` class.



pyedb.dotnet.edb_core.edb_data.variables.Variable(...)	Manages EDB methods for variable accessible from *Edb.Utility.VariableServer* property.
pyedb.dotnet.edb_core.edb_data.edbvalue.EdbValue(edb_obj)	Class defining Edb Value properties.

3.1.1 pyedb.dotnet.edb_core.edb_data.variables.Variable

class pyedb.dotnet.edb_core.edb_data.variables.**Variable**(*pedb*, *name*)

Manages EDB methods for variable accessible from *Edb.Utility.VariableServer* property.

__init__(*pedb*, *name*)

Methods

delete()	Delete this variable.

Attributes

`description`	Get the description of this variable.
`is_parameter`	Determine whether this variable is a parameter.
`name`	Get the name of this variable.
`value`	Get the value of this variable.
`value_object`	Get/Set the value of this variable.
`value_string`	Get/Set the value of this variable.

3.1.2 pyedb.dotnet.edb_core.edb_data.edbvalue.EdbValue

class `pyedb.dotnet.edb_core.edb_data.edbvalue.`**`EdbValue`**(*edb_obj*)

Class defining Edb Value properties.

`__init__`(*edb_obj*)

Attributes

`name`	Variable name.
`tofloat`	Returns the float number of the variable.
`tostring`	Returns the string of the variable.
`value`	Variable Value Object.

```
from pyedb import Edb

# this call returns the Edb class initialized on 2023 R1
edb = Edb(myedb, edbversion="2023.1")

...
```

3.1.3 EDB modules

This section lists the core EDB modules for reading and writing information to AEDB files.

hfss.EdbHfss	Manages EDB method to configure Hfss setup accessible from *Edb.hfss* property.
siwave.EdbSiwave	Manages EDB methods related to Siwave Setup accessible from *Edb.siwave* property.
materials.Materials	Manages EDB methods for material management accessible from *Edb.materials* property.
layout_validation.LayoutValidation	Manages all layout validation capabilities

pyedb.dotnet.edb_core.hfss.EdbHfss

class pyedb.dotnet.edb_core.hfss.**EdbHfss**(*p_edb*)

Manages EDB method to configure Hfss setup accessible from *Edb.hfss* property.

Examples

```
>>> from pyedb import Edb
>>> edbapp = Edb("myaedbfolder")
>>> edb_hfss = edb_3dedbapp.hfss
```

__init__(*p_edb*)

Methods

configure_hfss_analysis_setup([simulation_set	Configure HFSS analysis setup.
configure_hfss_extents([simulation_setup])	Configure the HFSS extent box.
create_bundle_wave_port(primitives_id, ...)	Create a bundle wave port.
create_circuit_port_on_net(...[, ...])	Create a circuit port on a NET.
create_circuit_port_on_pin(pos_pin, neg_pin)	Create Circuit Port on Pin.
create_coax_port_on_component(ref_des_list, ...)	Create a coaxial port on a component or component list on a net or net list.
create_current_source_on_net(...[, ...])	Create a current source.
create_current_source_on_pin(pos_pin, neg_pin)	Create a current source.
create_differential_wave_port(...[, ...])	Create a differential wave port.
create_edge_port_horizontal(prim_id, ...[, ...])	Create a horizontal edge port.
create_edge_port_on_polygon([polygon, ...])	Create lumped port between two edges from two different polygons.
create_edge_port_vertical(prim_id, point_on_edge)	Create a vertical edge port.
create_hfss_ports_on_padstack(pinpos[, portname])	Create an HFSS port on a padstack.
create_lumped_port_on_net([nets, ...])	Create an edge port on nets.
create_resistor_on_pin(pos_pin, neg_pin[, ...])	Create a Resistor boundary between two given pins.
create_rlc_boundary_on_pins([positive_pin, ...])	Create hfss rlc boundary on pins.
create_vertical_circuit_port_on_clipped_t:	Create an edge port on clipped signal traces.
create_voltage_source_on_net(...[, ...])	Create a voltage source.
create_voltage_source_on_pin(pos_pin, neg_pin)	Create a voltage source.
create_wave_port(prim_id, point_on_edge[, ...])	Create a wave port.
get_layout_bounding_box([layout, ...])	Evaluate the layout bounding box.
get_ports_number()	Return the total number of excitation ports in a layout.
get_trace_width_for_traces_with_ports()	Retrieve the trace width for traces with ports.
layout_defeaturing([simulation_setup])	Defeature the layout by reducing the number of points for polygons based on surface deviation criteria.
set_coax_port_attributes([simulation_setup])	Set coaxial port attribute with forcing default impedance to 50 Ohms and adjusting the coaxial extent radius.
trim_component_reference_size([...])	Trim the common component reference to the minimally acceptable size.

Attributes

excitations	Get all excitations.
hfss_extent_info	HFSS extent information.
probes	Get all probes.
sources	Get all sources.

pyedb.dotnet.edb_core.siwave.EdbSiwave

class pyedb.dotnet.edb_core.siwave.**EdbSiwave**(*p_edb*)

Manages EDB methods related to Siwave Setup accessible from *Edb.siwave* property.

Parameters

edb_class
[pyedb.edb.Edb] Inherited parent object.

Examples

```
>>> from pyedb import Edb
>>> edbapp = Edb("myaedbfolder", edbversion="2021.2")
>>> edb_siwave = edbapp.siwave
```

__init__(*p_edb*)

Methods

add_siwave_dc_analysis([name])	Add a Siwave DC analysis in EDB.
add_siwave_syz_analysis([accuracy_level, ...])	Add a SIwave AC analysis to EDB.
configure_siw_analysis_setup([...])	Configure Siwave analysis setup.
create_circuit_port_on_net(...[, ...])	Create a circuit port on a NET.
create_circuit_port_on_pin(pos_pin, neg_pin)	Create a circuit port on a pin.
create_circuit_port_on_pin_group(...[, ...])	Create a port between two pin groups.
create_current_source_on_net(...[, ...])	Create a current source.
create_current_source_on_pin(pos_pin, neg_pin)	Create a current source.
create_current_source_on_pin_group(...[, ...])	Create current source between two pin groups.
create_dc_terminal(component_name, net_name)	Create a dc terminal.
create_exec_file([add_dc, add_ac, add_syz, ...])	Create an executable file.
create_impedance_crosstalk_scan([scan_type])	Create Siwave crosstalk scan object
create_pin_group(reference_designator, ...)	Create pin group on the component.
create_pin_group_on_net(...[, group_name])	Create pin group on component by net name.
create_pin_group_terminal(source)	Create a pin group terminal.
create_port_between_pin_and_layer([...])	Create circuit port between pin and a reference layer.
create_resistor_on_pin(pos_pin, neg_pin[, ...])	Create a Resistor boundary between two given pins..
create_rlc_component(pins[, component_name, ...])	Create physical Rlc component.
create_voltage_probe_on_pin_group(...[, ...])	Create voltage probe between two pin groups.
create_voltage_source_on_net(...[, ...])	Create a voltage source.
create_voltage_source_on_pin(pos_pin, neg_pin)	Create a voltage source.
create_voltage_source_on_pin_group(...[, ...])	Create voltage source between two pin groups.
create_vrm_module([name, is_active, ...])	Create a voltage regulator module.
place_voltage_probe(name, positive_net_name, ...)	Place a voltage probe between two points.

Attributes

excitations	Get all excitations.
icepak_component_file	Icepak component file path.
icepak_use_minimal_comp_defaults	Icepak default setting.
pin_groups	All Layout Pin groups.
probes	Get all probes.
sources	Get all sources.
voltage_regulator_modules	Get all voltage regulator modules

pyedb.dotnet.edb_core.materials.Materials

class pyedb.dotnet.edb_core.materials.**Materials**(*edb: Edb*)

Manages EDB methods for material management accessible from *Edb.materials* property.

__init__(*edb: Edb*)

Methods

add_conductor_material(name, conductivity, ...)	Add a new conductor material.
add_debye_material(name, permittivity_low, ...)	Add a dielectric with the Debye model.
add_dielectric_material(name, permittivity, ...)	Add a new dielectric material in library.
add_djordjevicsarkar_dielectric(name, ...[, ...])	Add a dielectric using the Djordjevic-Sarkar model.
add_material(name, **kwargs)	Add a new material.
add_multipole_debye_material(name, ...)	Add a dielectric with the Multipole Debye model.
delete_material(material_name)	Remove a material from the database.
duplicate(material_name, new_material_name)	Duplicate a material from the database.
iterate_materials_in_amat([amat_file])	Iterate over material description in an AMAT file.
load_amat(amat_file)	Load materials from an AMAT file.
load_material(material)	Load material.
material_property_to_id(property_name)	Convert a material property name to a material property ID.
read_materials(amat_file)	Read materials from an AMAT file.
read_syslib_material(material_name)	Read a specific material from syslib AMAT file.
update_material(material_name, input_dict)	Update material attributes.

Attributes

materials	Get materials.
syslib	Get the project sys library.

pyedb.dotnet.edb_core.layout_validation.LayoutValidation

class pyedb.dotnet.edb_core.layout_validation.**LayoutValidation**(*pedb*)

Manages all layout validation capabilities

__init__(*pedb*)

Methods

dc_shorts([net_list, fix])	Find DC shorts on layout.
disjoint_nets([net_list, ...])	Find and fix disjoint nets from a given netlist.
fix_self_intersections([net_list])	Find and fix self intersections from a given netlist.
illegal_net_names([fix])	Find and fix illegal net names.
illegal_rlc_values([fix])	Find and fix RLC illegal values.

EdbValue	Class defining Edb Value properties.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

# this call returns the EdbHfss Class
comp = edb.hfss

# this call returns the Components Class
comp = edb.components

# this call returns the EdbSiwave Class
comp = edb.siwave

# this call returns the EdbPadstacks Class
comp = edb.padstacks

# this call returns the Stackup Class
comp = edb.stackup

# this call returns the Materials Class
comp = edb.materials

# this call returns the EdbNets Class
comp = edb.nets

...
```

3.2 Stackup & layers

These classes are the containers of the layer and stackup manager of the EDB API.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

# this call returns the EDBLayers class
layer = edb.stackup.stackup_layers
```

(continues on next page)

(continued from previous page)

```
# this call returns the EDBLayer class
layer = edb.stackup["TOP"]
...
```

Stackup	Manages EDB methods for stackup accessible from *Edb.stackup* property.

3.2.1 pyedb.dotnet.edb_core.stackup.Stackup

class pyedb.dotnet.edb_core.stackup.**Stackup**(*pedb*, *edb_object=None*)

Manages EDB methods for stackup accessible from *Edb.stackup* property.

__init__(*pedb*, *edb_object=None*)

Methods

add_document_layer(name[, layer_type])	Add a document layer.
add_layer(layer_name[, base_layer, method, ...])	Insert a layer into stackup.
add_layer_above(name, base_layer_name[, ...])	Add a layer above a layer.
add_layer_below(name, base_layer_name[, ...])	Add a layer below a layer.
add_layer_bottom(name[, layer_type])	Add a layer on bottom of the stackup.
add_layer_top(name[, layer_type])	Add a layer on top of the stackup.
add_outline_layer([outline_name])	Add an outline layer named "Outline" if it is not present.
adjust_solder_dielectrics()	Adjust the stack-up by adding or modifying dielectric layers that contains Solder Balls.
create_symmetric_stackup(layer_count[, ...])	Create a symmetric stackup.
export(fpath[, file_format, ...])	Export stackup definition to a CSV or JSON file.
flip_design()	Flip the current design of a layout.
get_layout_thickness()	Return the layout thickness.
limits([only_metals])	Retrieve stackup limits.
load(file_path[, rename])	Import stackup from a file.
place_a3dcomp_3d_placement(a3dcomp_path[, ...])	Place a 3D Component into current layout.
place_in_layout(edb[, angle, offset_x, ...])	Place current Cell into another cell using layer placement method.
place_in_layout_3d_placement(edb[, angle, ...])	Place current Cell into another cell using 3d placement method.
place_instance(component_edb[, angle, ...])	Place current Cell into another cell using 3d placement method.
plot([save_plot, size, plot_definitions, ...])	Plot current stackup and, optionally, overlap padstack definitions.
refresh_layer_collection()	Refresh layer collection from Edb.
remove_layer(name)	Remove a layer from stackup.
residual_copper_area_per_layer()	Report residual copper area per layer in percentage.
set_layer_clone(layer_clone)	
update_layout()	Set layer collection into edb.

Attributes

all_layers	
dielectric_layers	Dielectric layers.
layer_types	Layer types.
layers	Retrieve the dictionary of layers.
layers_by_id	Retrieve the list of layers with their ids.
mode	Stackup mode.
non_stackup_layers	Retrieve the dictionary of signal layers.
num_layers	Retrieve the stackup layer number.
signal_layers	Retrieve the dictionary of signal layers.
stackup_layers	Retrieve the dictionary of signal and dielectric layers.
thickness	Retrieve Stackup thickness.

LayerEdbClass	Manages Edb Layers.

3.2.2 pyedb.dotnet.edb_core.edb_data.layer_data.LayerEdbClass

class pyedb.dotnet.edb_core.edb_data.layer_data.**LayerEdbClass**(*pedb*, *edb_object=None*, *name=''*, *layer_type='undefined'*, ***kwargs*)

Manages Edb Layers. Replaces EDBLayer.

__init__(*pedb*, *edb_object=None*, *name=''*, *layer_type='undefined'*, ***kwargs*)

Methods

update(**kwargs)	

Attributes

color	Color of the layer.
fill_material	The layer's fill material.
id	
is_stackup_layer	Determine whether this layer is a stackup layer.
is_via_layer	Determine whether this layer is a via layer.
name	Retrieve name of the layer.
transparency	Retrieve transparency of the layer.
type	Retrieve type of the layer.

3.3 Modeler & primitives

These classes are the containers of primitives and all relative methods. Primitives are planes, lines, rectangles, and circles.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

top_layer_obj = edb.modeler.create_rectangle(
    "TOP", net_name="gnd", lower_left_point=plane_lw_pt, upper_right_point=plane_up_pt
)

...
```

EdbLayout	Manages EDB methods for primitives management accessible from *Edb.modeler* property.

3.3.1 pyedb.dotnet.edb_core.layout.EdbLayout

class pyedb.dotnet.edb_core.layout.**EdbLayout**(*p_edb*)

Manages EDB methods for primitives management accessible from *Edb.modeler* property.

Examples

```
>>> from pyedb import Edb
>>> edbapp = Edb("myaedbfolder", edbversion="2021.2")
>>> edb_layout = edbapp.modeler
```

__init__(*p_edb*)

Methods

add_void(shape, void_shape)	Add a void into a shape.
create_circle(layer_name, x, y, radius[, ...])	Create a circle on a specified layer.
create_polygon(main_shape, layer_name[, ...])	Create a polygon based on a list of points and voids.
create_rectangle(layer_name[, net_name, ...])	Create rectangle.
create_trace(path_list, layer_name[, width, ...])	Create a trace based on a list of points.
defeature_polygon(poly[, tolerance])	Defeature the polygon based on the maximum surface deviation criteria.
delete_primitives(net_names)	Delete primitives by net names.
fix_circle_void_for_clipping()	Fix issues when circle void are clipped due to a bug in EDB.
get_layout_statistics([evaluate_area, net_list])	Return EDBStatistics object from a layout.
get_polygon_bounding_box(polygon)	Retrieve a polygon bounding box.
get_polygon_points(polygon)	Retrieve polygon points.
get_polygons_by_layer(layer_name[, net_list])	Retrieve polygons by a layer.
get_primitive(primitive_id)	Retrieve primitive from give id.
get_primitive_by_layer_and_point([point, ...])	Return primitive given coordinate point [x, y], layer name and nets.
get_primitives([net_name, layer_name, ...])	Get primitives by conditions.
parametrize_polygon(polygon, selection_polygon)	Parametrize pieces of a polygon based on another polygon.
parametrize_trace_width(nets_name[, ...])	Parametrize a Trace on specific layer or all stackup.
shape_to_polygon_data(shape)	Convert a shape to polygon data.
unite_polygons_on_layer([layer_name, ...])	Try to unite all Polygons on specified layer.

Attributes

circles	Circles.
db	Db object.
layers	Dictionary of layers.
paths	Paths.
polygons	Polygons.
polygons_by_layer	Primitives with layer names as keys.
primitives	Primitives.
primitives_by_layer	Primitives with layer names as keys.
primitives_by_net	Primitives with net names as keys.
rectangles	Rectangles.

3.3.2 Primitives properties

These classes are the containers of data management for primitives and arcs.

EDBPrimitives	Manages EDB functionalities for a primitives.
EDBArcs	Manages EDB Arc Data functionalities.

pyedb.dotnet.edb_core.edb_data.primitives_data.EDBPrimitives

class pyedb.dotnet.edb_core.edb_data.primitives_data.**EDBPrimitives**(*raw_primitive*, *core_app*)

Manages EDB functionalities for a primitives. It Inherits EDB Object properties.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb_prim = edb.modeler.primitives[0]
>>> edb_prim.is_void # Class Property
>>> edb_prim.IsVoid() # EDB Object Property
```

__init__(*raw_primitive*, *core_app*)

Methods

area([include_voids])	Return the total area.
convert_to_polygon()	Convert path to polygon.
delete()	Delete this primitive.
get_closest_arc_midpoint(point)	Get the closest arc midpoint of the primitive to the input data.
get_closest_point(point)	Get the closest point of the primitive to the input data.
get_connected_object_id_set()	Produce a list of all geometries physically connected to a given layout object.
get_connected_objects()	Get connected objects.
intersect(primitives)	Intersect active primitive with one or more primitives.
intersection_type(primitive)	Get intersection type between actual primitive and another primitive or polygon data.
is_arc(point)	Either if a point is an arc or not.
is_intersecting(primitive)	Check if actual primitive and another primitive or polygon data intesects.
subtract(primitives)	Subtract active primitive with one or more primitives.
unite(primitives)	Unite active primitive with one or more primitives.

Attributes

arcs	Get the Primitive Arc Data.
bbox	Return the primitive bounding box points.
bounding_box	Bounding box.
center	Return the primitive bounding box center coordinate.
component	Component connected to this object.
id	Primitive ID.
is_negative	Determine whether this primitive is negative.
is_null	Flag indicating if this object is null.
is_void	Either if the primitive is a void or not.
layer	Get the primitive edb layer object.
layer_name	Get the primitive layer name.
longest_arc	Get the longest arc.
name	Name of the definition.
net	Net Object.
net_name	Get the primitive net name.
shortest_arc	Get the longest arc.
type	Return the type of the primitive.
voids	list of Primitive: List of void primitive objects inside the primitive.

pyedb.dotnet.edb_core.edb_data.primitives_data.EDBArcs

class pyedb.dotnet.edb_core.edb_data.primitives_data.**EDBArcs**(*app*, *arc*)

Manages EDB Arc Data functionalities. It Inherits EDB primitives arcs properties.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> prim_arcs = edb.modeler.primitives[0].arcs
>>> prim_arcs.center # arc center
>>> prim_arcs.points # arc point list
>>> prim_arcs.mid_point # arc mid point
```

__init__(*app*, *arc*)

Attributes

center	Arc center.
end	Get the coordinates of the ending point.
height	Get the height of the arc.
is_ccw	Test whether arc is counter clockwise.
is_point	Either if it is a point or not.
is_segment	Either if it is a straight segment or not.
length	Arc length.
mid_point	Arc mid point.
points	Return the list of points with arcs converted to segments.
points_raw	Return a list of Edb points.
radius	Arc radius.
start	Get the coordinates of the starting point.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

polygon = edbapp.modeler.polygons[0]
polygon.is_void
poly2 = polygon.clone()

...
```

3.4 Components

This section contains API references for component management. The main component object is called directly from main application using the property `components`.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

pins = edb.components.get_pin_from_component("U2A5")

...
```

Components	Manages EDB components and related method accessible from *Edb.components* property.

3.4.1 pyedb.dotnet.edb_core.components.Components

class pyedb.dotnet.edb_core.components.**Components**(*p_edb*)

Manages EDB components and related method accessible from *Edb.components* property.

Parameters

edb_class
[pyedb.dotnet.edb.Edb]

Examples

```
>>> from pyedb import Edb
>>> edbapp = Edb("myaedbfolder")
>>> edbapp.components
```

__init__(*p_edb*)

Methods

add_port_on_rlc_component([component, ...])	Deactivate RLC component and replace it with a circuit port.
add_rlc_boundary([component, circuit_type])	Add RLC gap boundary on component and replace it with a circuit port.
create(pins[, component_name, ...])	Create a component from pins.
create_pingroup_from_pins(pins[, group_name])	Create a pin group on a component.
create_port_on_component(component, net_list)	Create ports on a component.
create_port_on_pins(refdes, pins, reference_pins)	Create circuit port between pins and reference ones.
create_rlc_component(pins[, component_name, ...])	Create physical Rlc component.
create_source_on_component([sources])	Create voltage, current source, or resistor on component.
deactivate_rlc_component([component, ...])	Deactivate RLC component with a possibility to convert it to a circuit port.
delete(component_name)	Delete a component.
delete_single_pin_rlc([deactivate_only])	Delete all RLC components with a single pin.
disable_rlc_component(component_name)	Disable a RLC component.
export_bom(bom_file[, delimiter])	Export Bom file from layout.
export_definition(file_path)	Export component definitions to json file.
find_by_reference_designator(...)	Find a component.
get_aedt_pin_name(pin)	Retrieve the pin name that is shown in AEDT.
get_component_by_name(name)	Retrieve a component by name.
get_component_net_connection_info(refdes)	Retrieve net connection information.
get_component_placement_vector(...[, flipped])	Get the placement vector between 2 components.
get_components_from_nets([netlist])	Retrieve components from a net list.
get_nets_from_pin_list(PinList)	Retrieve nets with one or more pins.

continues on next page

Table 1 – continued from previous page

`get_pin_from_component`(component[, netName, ...])	Retrieve the pins of a component.
`get_pin_position`(pin)	Retrieve the pin position in meters.
`get_pins`(reference_designator[, net_name, ...])	Get component pins.
`get_pins_name_from_net`(net_name[, pin_list])	Retrieve pins belonging to a net.
`get_rats`()	Retrieve a list of dictionaries of the reference designator, pin names, and net names.
`get_solder_ball_height`(cmp)	Get component solder ball height.
`get_through_resistor_list`([threshold])	Retrieve through resistors.
`import_bom`(bom_file[, delimiter, ...])	Load external BOM file.
`import_definition`(file_path)	Import component definition from json file.
`refresh_components`()	Refresh the component dictionary.
`replace_rlc_by_gap_boundaries`([component])	Replace RLC component by RLC gap boundaries.
`set_component_model`(componentname[, ...])	Assign a Spice or Touchstone model to a component.
`set_component_rlc`(componentname[, ...])	Update values for an RLC component.
`set_solder_ball`([component, sball_diam, ...])	Set cylindrical solder balls on a given component.
`short_component_pins`(component_name[, ...])	Short pins of component with a trace.
`update_rlc_from_bom`(bom_file[, delimiter, ...])	Update the EDC core component values (RLCs) with values coming from a BOM file.

Attributes

`ICs`	Integrated circuits.
`IOs`	Circuit inupts and outputs.
`Others`	Other core components.
`capacitors`	Capacitors.
`components_by_partname`	Components by part name.
`definitions`	Retrieve component definition list.
`inductors`	Inductors.
`instances`	All Cell components objects.
`nport_comp_definition`	Retrieve Nport component definition list.
`resistors`	Resistors.

3.4.2 Instances and definitions

These classes are the containers of data management for components reference designator and definitions.

EDBComponent	Manages EDB functionalities for components.

pyedb.dotnet.edb_core.cell.hierarchy.component.EDBComponent

class pyedb.dotnet.edb_core.cell.hierarchy.component.**EDBComponent**(*pedb*, *edb_object*)

Manages EDB functionalities for components.

Parameters

parent
[*pyedb.dotnet.edb_core.components.Components*] Components object.

component
[object] Edb Component Object

__init__(*pedb*, *edb_object*)

Methods

assign_rlc_model([res, ind, cap, is_parallel])	Assign RLC to this component.
assign_s_param_model(file_path[, name, ...])	Assign S-parameter to this component.
assign_spice_model(file_path[, name, ...])	Assign Spice model to this component.
create_clearance_on_component([...])	Create a Clearance on Soldermask layer by drawing a rectangle.
create_package_def([name, component_part_name])	Create a package definition and assign it to the component.
delete()	Delete this primitive.
use_s_parameter_model(name[, reference_net])	Use S-parameter model on the component.

Attributes

bounding_box	Component's bounding box.
cap_value	Capacitance Value.
center	Compute the component center.
component	Component connected to this object.
component_def	Component definition.
component_instance	Edb component instance.
component_property	ComponentProperty object.
enabled	Get or Set the component to active mode.
id	Primitive ID.
ind_value	Inductance Value.
is_enabled	Flag indicating if the current object is enabled.
is_null	Flag indicating if the current object exists.
is_parallel_rlc	Define if model is Parallel or Series.
is_top_mounted	Check if a component is mounted on top or bottom of the layout.
layout_instance	EDB layout instance object.
location	XY Coordinates.
lower_elevation	Lower elevation of the placement layer.
model	Component model.
model_type	Retrieve assigned model type.
name	Name of the definition.
net	Net Object.

continues on next page

Table 2 – continued from previous page

netlist_model	Get assigned netlist model properties.
nets	Nets of Component.
numpins	Number of Pins of Component.
package_def	Package definition.
part_name	Component part name.
partname	Component part name.
pinlist	Pins of the component.
pins	EDBPadstackInstance of Component.
placement_layer	Placement layer.
refdes	Reference Designator Name.
res_value	Resistance value.
rlc_values	Get component rlc values.
rotation	Compute the component rotation in radian.
s_param_model	Get assigned S-parameter model properties.
solder_ball_diameter	Solder ball diameter.
solder_ball_height	Solder ball height if available.
solder_ball_placement	Solder ball placement if available..
solder_ball_shape	Solder ball shape.
spice_model	Get assigned Spice model properties.
top_bottom_association	Top/bottom association of the placement layer.
type	Component type.
upper_elevation	Upper elevation of the placement layer.
value	Retrieve discrete component value.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

comp = edb.components["C1"]

comp.is_enabled = True
part = edb.components.definitions["AAA111"]
...
```

3.5 Nets

This section contains API references for net management. The main component object is called directly from main application using the property `nets`.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

edb.nets.plot(None, None)

...
```

EdbNets	Manages EDB methods for nets management accessible from *Edb.nets* property.

3.5.1 pyedb.dotnet.edb_core.nets.EdbNets

class pyedb.dotnet.edb_core.nets.**EdbNets**(*p_edb*)

Manages EDB methods for nets management accessible from *Edb.nets* property.

Examples

```
>>> from pyedb import Edb
>>> edbapp = Edb("myaedbfolder", edbversion="2021.2")
>>> edb_nets = edbapp.nets
```

__init__(*p_edb*)

Methods

`classify_nets`([power_nets, signal_nets])	Reassign power/ground or signal nets based on list of nets.
`delete`(netlist)	Delete one or more nets from EDB.
`eligible_power_nets`([threshold])	Return a list of nets calculated by area to be eligible for PWR/Ground net classification.
`find_or_create_net`([net_name, start_with, ...])	Find or create the net with the given name in the layout.
`generate_extended_nets`([resistor_below, ...])	Get extended net and associated components.
`get_dcconnected_net_list`([ground_nets, ...])	Get the nets connected to the direct current through inductors.
`get_net_by_name`(net_name)	Find a net by name.
`get_plot_data`([nets, layers, color_by_net, ...])	Return List of points for Matplotlib 2D Chart.
`get_powertree`(power_net_name, ground_nets)	Retrieve the power tree.
`is_net_in_component`(component_name, net_name)	Check if a net belongs to a component.
`is_power_gound_net`(netname_list)	Determine if one of the nets in a list is power or ground.
`merge_nets_polygons`(net_names_list)	Convert paths from net into polygons, evaluate all connected polygons and perform the merge.
`plot`([nets, layers, color_by_net, ...])	Plot a Net to Matplotlib 2D Chart.

Attributes

`components_by_nets`	Get all component instances grouped by nets.
`db`	Db object.
`netlist`	Return the cell netlist.
`nets`	Nets.
`nets_by_components`	Get all nets for each component instance.
`power`	Power nets.
`signal`	Signal nets.

3.5.2 Net properties

The following class is the container of data management for nets, extended nets and differential pairs.

EDBNetsData	Manages EDB functionalities for a primitives.
EDBNetClassData	Manages EDB functionalities for a primitives.
EDBExtendedNetData	Manages EDB functionalities for a primitives.
EDBDifferentialPairData	Manages EDB functionalities for a primitive.

pyedb.dotnet.edb_core.edb_data.nets_data.EDBNetsData

class pyedb.dotnet.edb_core.edb_data.nets_data.**EDBNetsData**(*raw_net*, *core_app*)

Manages EDB functionalities for a primitives. It Inherits EDB Object properties.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb_net = edb.nets.nets["GND"]
>>> edb_net.name # Class Property
>>> edb_net.name # EDB Object Property
```

__init__(*raw_net*, *core_app*)

Methods

create(layout, name)	Edb Dotnet Api Database *Edb.Net.Create*.
delete()	Edb Dotnet Api Database *Edb.Net.Delete*.
find_by_name(layout, net)	Edb Dotnet Api Database *Edb.Net.FindByName*.
find_dc_short([fix])	Find DC-shorted nets.
get_smallest_trace_width()	Retrieve the smallest trace width from paths.
plot([layers, show_legend, save_plot, ...])	Plot a net to Matplotlib 2D chart.

Attributes

api_class	Return Ansys.Ansoft.Edb class object.
api_object	Return Ansys.Ansoft.Edb object.
components	Return the list of components that touch the net.
extended_net	Get extended net and associated components.
is_null	Edb Dotnet Api Database *Net.IsNull()*.
is_power_ground	Edb Dotnet Api Database *Net.IsPowerGround()* and *Net.SetIsPowerGround()*.
name	Edb Dotnet Api Database *net.name* and *Net.SetName()*.
padstack_instances	Return the list of primitives that belongs to the net.
primitives	Return the list of primitives that belongs to the net.

pyedb.dotnet.edb_core.edb_data.nets_data.EDBNetClassData

class pyedb.dotnet.edb_core.edb_data.nets_data.**EDBNetClassData**(*core_app*, *raw_extended_net=None*)

Manages EDB functionalities for a primitives. It inherits EDB Object properties.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb.net_classes
```

__init__(*core_app*, *raw_extended_net=None*)

Methods

add_net(name)	Add a new net.
api_create(name)	Edb Dotnet Api Database *Edb.NetClass.Create*.
delete()	Edb Dotnet Api Database *Delete*.

Attributes

api_nets	Return Edb Nets object dictionary.
is_null	Edb Dotnet Api Database *NetClass.IsNull()*.
name	Edb Dotnet Api Database *NetClass.name* and *NetClass.SetName()*.
nets	Get nets belong to this net class.

pyedb.dotnet.edb_core.edb_data.nets_data.EDBExtendedNetData

class pyedb.dotnet.edb_core.edb_data.nets_data.**EDBExtendedNetData**(*core_app*, *raw_extended_net=None*)

Manages EDB functionalities for a primitives. It Inherits EDB Object properties.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb_extended_net = edb.nets.extended_nets["GND"]
>>> edb_extended_net.name # Class Property
```

__init__(*core_app*, *raw_extended_net=None*)

Methods

add_net(name)	Add a new net.
api_create(name)	Edb Dotnet Api Database *Edb.ExtendedNet.Create*.
delete()	Edb Dotnet Api Database *Delete*.
find_by_name(layout, net)	Edb Dotnet Api Database *Edb.ExtendedNet.FindByName*.

Attributes

api_class	Return Ansys.Ansoft.Edb class object.
api_nets	Return Edb Nets object dictionary.
components	Dictionary of components.
is_null	Edb Dotnet Api Database *NetClass.IsNull()*.
name	Edb Dotnet Api Database *NetClass.name* and *NetClass.SetName()*.
nets	Nets dictionary.
rlc	Dictionary of RLC components.
serial_rlc	Dictionary of serial RLC components.
shunt_rlc	Dictionary of shunt RLC components.

pyedb.dotnet.edb_core.edb_data.nets_data.EDBDifferentialPairData

class pyedb.dotnet.edb_core.edb_data.nets_data.**EDBDifferentialPairData**(*core_app*, *api_object=None*)

Manages EDB functionalities for a primitive. It inherits EDB object properties.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> diff_pair = edb.differential_pairs["DQ4"]
>>> diff_pair.positive_net
>>> diff_pair.negative_net
```

__init__(*core_app*, *api_object=None*)

Methods

add_net(name)	Add a new net.
api_create(name)	Edb Dotnet Api Database *Edb.DifferentialPair.Create*.
delete()	Edb Dotnet Api Database *Delete*.
find_by_name(layout, net)	Edb Dotnet Api Database *Edb.DifferentialPair.FindByName*.

Attributes

api_class	Return Ansys.Ansoft.Edb class object.
api_negative_net	Edb Api Negative net object.
api_nets	Return Edb Nets object dictionary.
api_positive_net	Edb Api Positive net object.
is_null	Edb Dotnet Api Database *NetClass.IsNull()*.
name	Edb Dotnet Api Database *NetClass.name* and *NetClass.SetName()*.
negative_net	Negative Net.
positive_net	Positive Net.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

edb.nets["M_MA<6>"].delete()
edb.net_classes
edb.differential_pairs
edb.extended_nets



...
```

3.6 vias and padstacks

This section contains API references for padstack management. The main padstack object is called directly from main application using the property `padstacks`.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

edb.padstacks.create_padstack(
    padstackname="SVIA",
    holediam="$via_hole_size",
    antipaddiam="$antipaddiam",
    paddiam="$paddiam",
)



...
```

EdbPadstacks	Manages EDB methods for nets management accessible from *Edb.padstacks* property.

3.6.1 pyedb.dotnet.edb_core.padstack.EdbPadstacks

class pyedb.dotnet.edb_core.padstack.**EdbPadstacks**(*p_edb*)

Manages EDB methods for nets management accessible from *Edb.padstacks* property.

Examples

```
>>> from pyedb import Edb
>>> edbapp = Edb("myaedbfolder", edbversion="2021.2")
>>> edb_padstacks = edbapp.padstacks
```

__init__(*p_edb*)

Methods

check_and_fix_via_plating([...])	Check for minimum via plating ration value, values found below the minimum one are replaced by default plating ratio.
create([padstackname, holediam, paddiam, ...])	Create a padstack.
create_circular_padstack([padstackname, ...])	Create a circular padstack.
create_coax_port(padstackinstance[, ...])	Create HFSS 3Dlayout coaxial lumped port on a pastack Requires to have solder ball defined before calling this method.
delete_padstack_instances(net_names)	Delete padstack instances by net names.
duplicate(target_padstack_name[, ...])	Duplicate a padstack.
get_instances([name, pid, definition_name, ...])	Get padstack instances by conditions.
get_pad_parameters(pin, layername[, pad_type])	Get Padstack Parameters from Pin or Padstack Definition.
get_padstack_instance_by_net_name(net_name)	Get a list of padstack instances by net name.
get_padstack_instances_intersecting_bound	Returns the list of padstack instances ID intersecting a given bounding box and nets.
get_padstack_instances_rtree_index([nets])	Returns padstack instances Rtree index.
get_pinlist_from_component_and_net([refdes, ...])	Retrieve pins given a component's reference designator and net name.
get_reference_pins(positive_pin[, ...])	Search for reference pins using given criteria.
get_via_instance_from_net([net_list])	Get the list for EDB vias from a net name list.
int_to_geometry_type([val])	Convert an integer to an EDB.PadGeometryType.
int_to_pad_type([val])	Convert an integer to an EDB.PadGeometryType.
merge_via_along_lines([net_name, ...])	Replace padstack instances along lines into a single polygon.
place(position, definition_name[, net_name, ...])	Place a via.
remove_pads_from_padstack(padstack_name[, ...])	Remove the Pad from a padstack on a specific layer by setting it as a 0 thickness circle.
set_all_antipad_value(value)	Set all anti-pads from all pad-stack definition to the given value.
set_pad_property(padstack_name[, ...])	Set pad and antipad properties of the padstack.
set_solderball(padstackInst, sballLayer_name)	Set solderball for the given PadstackInstance.

Attributes

db	Db object.
definitions	Padstack definitions.
instances	Dictionary of all padstack instances (vias and pins).
instances_by_name	Dictionary of all padstack instances (vias and pins) by name.
pad_type	Return a PadType Enumerator.
pingroups	All Layout Pin groups.
pins	Dictionary of all pins instances (belonging to component).
vias	Dictionary of all vias instances not belonging to component.

3.6.2 Instances and definitions

These classes are the containers of data management for padstacks instances and padstack definitions.

EDBPadProperties	Manages EDB functionalities for pad properties.
EDBPadstack	Manages EDB functionalities for a padstack.
EDBPadstackInstance	Manages EDB functionalities for a padstack.

pyedb.dotnet.edb_core.edb_data.padstacks_data.EDBPadProperties

class pyedb.dotnet.edb_core.edb_data.padstacks_data.**EDBPadProperties**(*edb_padstack*, *layer_name*, *pad_type*, *p_edb_padstack*)

Manages EDB functionalities for pad properties.

Parameters

edb_padstack

layer_name
[str] Name of the layer.

pad_type
Type of the pad.

pedbpadstack
[str] Inherited AEDT object.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb_pad_properties = edb.padstacks.definitions["MyPad"].pad_by_layer["TOP"]
```

__init__(*edb_padstack*, *layer_name*, *pad_type*, *p_edb_padstack*)

Methods

int_to_geometry_type([val])	Convert an integer to an EDB.PadGeometryType.
int_to_pad_type([val])	Convert an integer to an EDB.PadGeometryType.

Attributes

geometry_type	Geometry type.
offset_x	Offset for the X axis.
offset_y	Offset for the Y axis.
parameters	Get parameters.
parameters_values	Parameters.
parameters_values_string	Parameters value in string format.
polygon_data	Parameters.
rotation	Rotation.
shape	Get the shape of the pad.

pyedb.dotnet.edb_core.edb_data.padstacks_data.EDBPadstack

class pyedb.dotnet.edb_core.edb_data.padstacks_data.**EDBPadstack**(*edb_padstack*, *ppadstack*)

Manages EDB functionalities for a padstack.

Parameters

edb_padstack

ppadstack

[str] Inherited AEDT object.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb_padstack = edb.padstacks.definitions["MyPad"]
```

__init__(*edb_padstack*, *ppadstack*)

Methods

convert_to_3d_microvias([...])	Convert actual padstack instance to microvias 3D Objects with a given aspect ratio.
split_to_microvias()	Convert actual padstack definition to multiple microvias definitions.

Attributes

`hole_diameter`	Hole diameter.
`hole_diameter_string`	Hole diameter in string format.
`hole_finished_size`	Finished hole size.
`hole_offset_x`	Hole offset for the X axis.
`hole_offset_y`	Hole offset for the Y axis.
`hole_parameters`	Hole parameters.
`hole_params`	Via Hole parameters values.
`hole_plating_ratio`	Hole plating ratio.
`hole_plating_thickness`	Hole plating thickness.
`hole_properties`	Hole properties.
`hole_range`	Get hole range value from padstack definition.
`hole_rotation`	Hole rotation.
`hole_type`	Hole type.
`instances`	Definitions Instances.
`material`	Hole material.
`name`	Padstack Definition Name.
`padstack_instances`	Get all the vias that belongs to active Padstack definition.
`via_layers`	Layers.
`via_start_layer`	Starting layer.
`via_stop_layer`	Stopping layer.

pyedb.dotnet.edb_core.edb_data.padstacks_data.EDBPadstackInstance

class pyedb.dotnet.edb_core.edb_data.padstacks_data.**EDBPadstackInstance**(*edb_padstackinstance*, *_pedb*)

Manages EDB functionalities for a padstack.

Parameters

edb_padstackinstance

_pedb
Inherited AEDT object.

Examples

```
>>> from pyedb import Edb
>>> edb = Edb(myedb, edbversion="2021.2")
>>> edb_padstack_instance = edb.padstacks.instances[0]
```

__init__(*edb_padstackinstance*, *_pedb*)

Methods

`create_coax_port`([name, radial_extent_factor])	Create a coax port.
`create_port`([name, reference, is_circuit_port])	Create a port on the padstack.
`create_rectangle_in_pad`(layer_name[, ...])	Create a rectangle inscribed inside a padstack instance pad.
`create_terminal`([name])	Create a padstack instance terminal
`delete`()	Delete this primitive.
`get_connected_object_id_set`()	Produce a list of all geometries physically connected to a given layout object.
`get_connected_objects`()	Get connected objects.
`get_reference_pins`([reference_net, ...])	Search for reference pins using given criteria.
`get_terminal`([name, create_new_terminal])	Get PadstackInstanceTerminal object.
`in_polygon`(polygon_data[, include_partial, ...])	Check if padstack Instance is in given polygon data.
`in_voids`([net_name, layer_name])	Check if this padstack instance is in any void.
`parametrize_position`([prefix])	Parametrize the instance position.
`set_backdrill_bottom`(drill_depth, drill_diameter)	Set backdrill from bottom.
`set_backdrill_top`(drill_depth, drill_diameter)	Set backdrill from top.

Attributes

`aedt_name`	Retrieve the pin name that is shown in AEDT.
`backdrill_bottom`	Backdrill layer from bottom.
`backdrill_top`	Backdrill layer from top.
`bounding_box`	Get bounding box of the padstack instance.
`component`	Component.
`dcir_equipotential_region`	Check whether dcir equipotential region is enabled.
`id`	Primitive ID.
`is_null`	Flag indicating if this object is null.
`is_pin`	Determines whether this padstack instance is a layout pin.
`is_void`	Either if the primitive is a void or not.
`layer`	Get the primitive edb layer object.
`layer_name`	Get the primitive layer name.
`layer_range_names`	List of all layers to which the padstack instance belongs.
`lower_elevation`	Lower elevation of the placement layer.
`metal_volume`	Metal volume of the via hole instance in cubic units (m3).
`name`	Padstack Instance Name.
`net`	Net Object.
`net_name`	Net name.
`object_instance`	Return Ansys.Ansoft.Edb.LayoutInstance.LayoutObjInstance object.
`padstack_definition`	Padstack definition.
`pin`	EDB padstack object.
`pin_number`	Get pin number.
`pingroups`	Pin groups that the pin belongs to.
`placement_layer`	Placement layer.

continues on next page

Table 3 – continued from previous page

position	Padstack instance position.
rotation	Padstack instance rotation.
start_layer	Starting layer.
stop_layer	Stopping layer.
terminal	Terminal.
top_bottom_association	Top/bottom association of the placement layer.
type	Return the type of the primitive.
upper_elevation	Upper elevation of the placement layer.

3.7 Sources and excitations

These classes are the containers of sources methods of the EDB for both HFSS and Siwave.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

# this call returns the EDB excitations dictionary
edb.excitations
...
```

3.8 Simulation configuration

These classes are the containers of simulation configuration constructors for the EDB.

SimulationConfiguration	Provides an ASCII simulation configuration file parser.
SimulationConfigurationDc	Contains all DC analysis settings.
SimulationConfigurationAc	Contains all AC analysis settings.
SimulationConfigurationBatch	Contains all Cutout and Batch analysis settings.

3.8.1 pyedb.dotnet.edb_core.edb_data.simulation_configuration.SimulationConfiguration

class pyedb.dotnet.edb_core.edb_data.simulation_configuration.**SimulationConfiguration**(*filename=None*, *edb=None*)

Provides an ASCII simulation configuration file parser.

This parser supports all types of inputs for setting up and automating any kind of SI or PI simulation with HFSS 3D Layout or Siwave. If fields are omitted, default values are applied. This class can be instantiated directly from Configuration file.

Examples

This class is very convenient to build HFSS and SIwave simulation projects from layout. It is leveraging EDB commands from Pyaedt but with keeping high level parameters making more easy PCB automation flow. SYZ and DC simulation can be addressed with this class.

The class is instantiated from an open edb:

```
>>> from pyedb import Edb
>>> edb = Edb()
>>> sim_setup = edb.new_simulation_configuration()
```

The returned object sim_setup is a SimulationConfiguration object. From this class you can assign a lot of parameters related the project configuration but also solver options. Here is the list of parameters available:

```
>>> from dotnet.generic.constants import SolverType
>>> sim_setup.solver_type = SolverType.Hfss3dLayout
```

Solver type can be selected, HFSS 3D Layout and Siwave are supported.

```
>>> sim_setup.signal_nets = ["net1", "net2"]
```

Set the list of net names you want to include for the simulation. These nets will have excitations ports created if corresponding pins are found on selected component. We usually refer to signal nets but power / reference nets can also be passed into this list if user wants to have ports created on these ones.

```
>>> sim_setup.power_nets = ["gnd", "vcc"]
```

Set the list on power and reference nets. These nets wont have excitation ports created on them and will be clipped during the project build if the cutout option is enabled.

```
>>> sim_setup.components = ["comp1", "comp2"]
```

Set the list of components which will be included in the simulation. These components will have ports created on pins belonging to the net list.

```
>>> sim_setup.do_cutout_subdesign = True
```

When true activates the layout cutout based on net signal net selection and cutout expansion.

```
>>> from dotnet.generic.constants import CutoutSubdesignType
>>> sim_setup.cutout_subdesign_type = CutoutSubdesignType.Conformal
```

Define the type of cutout used for computing the clippingextent polygon. CutoutSubdesignType.Conformal CutoutSubdesignType.BBox are surpported.

```
>>> sim_setup.cutout_subdesign_expansion = "4mm"
```

Define the distance used for computing the extent polygon. Integer or string can be passed. For example 0.001 is in meter so here 1mm. You can also pass the string 1mm for the same result.

```
>>> sim_setup.cutout_subdesign_round_corner = True
```

Boolean to allow using rounded corner for the cutout extent or not.

```
>>> sim_setup.use_default_cutout = False
```

When True use the native edb API command to process the cutout. Using False uses the Pyaedt one which improves the cutout speed.

```
>>> sim_setup.generate_solder_balls = True
```

Boolean to activate the solder ball generation on components. When HFSS solver is selected in combination with this parameter, coaxial ports will be created on solder balls for pins belonging to selected signal nets. If Siwave solver is selected this parameter will be ignored.

```
>>> sim_setup.use_default_coax_port_radial_extension = True
```

When `True` the default coaxial extent is used for the ports (only for HFSS). When the design is having dense solder balls close to each other (like typically package design), the default value might be too large and cause port overlapping, then solver failure. To prevent this issue set this parameter to `False` will use a smaller value.

```
>>> sim_setup.output_aedb = r"C:    emp\my_edb.aedb"
```

Specify the output edb file after building the project. The parameter must be the complete file path. leaving this parameter blank will oervwritte the current open edb.

```
>>>  sim_setup.dielectric_extent = 0.01
```

Gives the dielectric extent after cutout, keeping default value is advised unless for very specific application.

```
>>> sim_setup.airbox_horizontal_extent = "5mm"
```

Provide the air box horizonzal extent values. Unitless float value will be treated as ratio but string value like 5mm is also supported.

```
>>> sim_setup.airbox_negative_vertical_extent = "5mm"
```

Provide the air box negative vertical extent values. Unitless float value will be treated as ratio but string value like 5mm is also supported.

```
>>> sim_setup.airbox_positive_vertical_extent = "5mm"
```

Provide the air box positive vertical extent values. Unitless float value will be treated as ratio but string value like 5mm is also supported.

```
>>> sim_setup.use_radiation_boundary = True
```

When `True` use radiation airbox boundary condition and perfect metal box when set to `False`. Default value is `True`, using enclosed metal box will greatly change simulation results. Setting this parameter as `False` must be used cautiously.

```
>>> sim_setup.do_cutout_subdesign = True
```

`True` activates the cutout with associated parameters. Setting `False` will keep the entire layout. Setting to `False` can impact the simulation run time or even memory failure if HFSS solver is used.

```
>>> sim_setup.do_pin_group = False
```

When circuit ports are used, setting to `True` will force to create pin groups on components having pins belonging to same net. Setting to `False` will generate port on each signal pin with taking the closest reference pin. The last configuration is more often used when users are creating ports on PDN (Power delivery Network) and want to connect all pins individually.

```
>>> from dotnet.generic.constants import SweepType
>>> sim_setup.sweep_type = SweepType.Linear
```

Specify the frequency sweep type, Linear or Log sweep can be defined.

SimulationCOnfiguration also inherit from SimulationConfigurationAc class for High frequency settings.

```
>>> sim_setup.start_freq = "0Hz"
```

Define the start frequency from the sweep.

```
>>> sim_setup.stop_freq = "40GHz"
```

Define the stop frequency from the sweep.

```
>>> sim_setup.step_freq = "10MHz"
```

Define the step frequency from the sweep.

```
>>> sim_setup.decade_count = 100
```

Used when log sweep is defined and specify the number of points per decade.

```
>>> sim_setup.enforce_causality = True
```

Activate the option `Enforce Causality` for the solver, recommended for signal integrity application

```
>>> sim_setup.enforce_passivity = True
```

Activate the option `Enforce Passivity` for the solver, recommended for signal integrity application

```
>>> sim_setup.do_lambda_refinement = True
```

Activate the lambda refinement for the initial mesh (only for HFSS), default value is `True`. Keeping this activated is highly recommended.

```
>>> sim_setup.use_q3d_for_dc = False
```

Enable when `True` the Q3D DC point computation. Only needed when very high accuracy is required for DC point. Can eventually cause extra computation time.

```
>>> sim_setup.sweep_name = "Test_sweep"
```

Define the frequency sweep name.

```
>>> sim_setup.mesh_freq = "10GHz"
```

Define the frequency used for adaptive meshing (available for both HFSS and SIwave).

```
>>> from dotnet.generic.constants import RadiationBoxType
>>> sim_setup.radiation_box = RadiationBoxType.ConvexHull
```

Defined the radiation box type, Conformal, Bounding box and ConvexHull are supported (HFSS only).

```
>>> sim_setup.max_num_passes= 30
```

Default value is 30, specify the maximum number of adaptive passes (only HFSS). Reasonable high value is recommended to force the solver reaching the convergence criteria.

```
>>> sim_setup.max_mag_delta_s = 0.02
```

Define the convergence criteria

```
>>> sim_setup.min_num_passes = 2
```

specify the minimum number of consecutive coberged passes. Setting to 2 is a good practice to avoid converging on local minima.

```
>>> from dotnet.generic.constants import BasisOrder
>>> sim_setup.basis_order =  BasisOrder.Single
```

Select the order basis (HFSS only), Zero, Single, Double and Mixed are supported. For Signal integrity Single or Mixed should be used.

```
>>> sim_setup.minimum_void_surface = 0
```

Only for Siwave, specify the minimum void surface to be meshed. Void with lower surface value will be ignored by meshing.

SimulationConfiguration also inherits from SimulationDc class to handle DC simulation projects.

```
>>> sim_setup.dc_compute_inductance = True
```

`True` activate the DC loop inductance computation (Siwave only), `False` is deactivated.

```
>>> sim_setup.dc_slide_position = 1
```

The provided value must be between 0 and 2 and correspond ti the SIwave DC slide position in GUI. 0 : coarse 1 : medium accuracy 2 : high accuracy

```
>>> sim_setup.dc_plot_jv = True
```

`True` activate the current / voltage plot with Siwave DC solver, `False` deactivate.

```
>>> sim_setup.dc_error_energy = 0.02
```

Fix the DC error convergence criteria. In this example 2% is defined.

```
>>> sim_setup.dc_max_num_pass = 6
```

Provide the maximum number of passes during Siwave DC adaptive meshing.

```
>>> sim_setup.dc_min_num_pass = 1
```

Provide the minimum number of passes during Siwave DC adaptive meshing.

```
>>> sim_setup.dc_mesh_bondwires = True
```

`True` bondwires are meshed, `False` bond wires are ignored during meshing.

```
>>> sim_setup.dc_num_bondwire_sides = 8
```

Gives the number of facets wirebonds are discretized.

```
>>> sim_setup.dc_refine_vias = True
```

`True` meshing refinement on nondwires activated during meshing process. Deactivated when set to `False`.

```
>>> sim_setup.dc_report_show_Active_devices = True
```

Activate when `True` the components showing in the DC report.

```
>>> sim_setup.dc_export_thermal_data = True
```

`True` thermal data are exported for Icepak simulation.

```
>>> sim_setup.dc_full_report_path = r"C:    emp\my_report.html"
```

Provides the file path for the DC report.

```
>>> sim_setup.dc_icepak_temp_file = r"C:    emp\my_file"
```

Provides icepak temporary files location.

```
>>> sim_setup.dc_import_thermal_data = False
```

Import DC thermal data when *True* `

```
>>> sim_setup.dc_per_pin_res_path = r"C:    emp\dc_pin_res_file"
Provides the resistance per pin file path.
```

```
>>> sim_setup.dc_per_pin_use_pin_format = True
```

When `True` activate the pin format.

```
>>> sim_setup.dc_use_loop_res_for_per_pin = True
```

Activate the loop resistance usage per pin when `True`

```
>>> sim_setup.dc_via_report_path = 'C:\temp\via_report_file'
```

Define the via report path file.

```
>>> sim_setup.add_current_source(name="test_isrc",
>>>                                 current_value=1.2,
>>>                                 phase_value=0.0,
>>>                                 impedance=5e7,
>>>                                 positive_node_component="comp1",
>>>                                 positive_node_net="net1",
>>>                                 negative_node_component="comp2",
>>>                                 negative_node_net="net2"
>>>                             )
```

Define a current source.

```
>>> sim_setup.add_dc_ground_source_term(source_name="test_isrc", node_to_ground=1)
```

Define the pin from a source which has to be set to reference for DC simulation.

```
>>> sim_setup.add_voltage_source(name="test_vsrc",
>>>                                 current_value=1.33,
>>>                                 phase_value=0.0,
>>>                                 impedance=1e-6,
>>>                                 positive_node_component="comp1",
>>>                                 positive_node_net="net1",
>>>                                 negative_node_component="comp2",
>>>                                 negative_node_net="net2"
>>>                             )
```

Define a voltage source.

```
>>> sim_setup.add_dc_ground_source_term(source_name="test_vsrc", node_to_ground=1)
```

Define the pin from a source which has to be set to reference for DC simulation.

```
>>> edb.build_simulation_project(sim_setup)
```

Will build and save your project.

__init__(*filename=None*, *edb=None*)

Methods

add_current_source([name, current_value, ...])	Add a current source for the current SimulationConfiguration instance.
add_dc_ground_source_term([source_name, ...])	Add a dc ground source terminal for Siwave.
add_rlc([name, r_value, c_value, l_value, ...])	Add a voltage source for the current SimulationConfiguration instance.
add_voltage_source([name, voltage_value, ...])	Add a voltage source for the current SimulationConfiguration instance.
build_simulation_project()	Build active simulation project.
export_json(output_file)	Export Json file from SimulationConfiguration object.
import_json(input_file)	Import Json file into SimulationConfiguration object instance.

Attributes

`ac_settings`	AC Settings class.
`batch_solve_settings`	Cutout and Batch Settings class.
`dc_settings`	DC Settings class.
`filename`	Retrieve the file name loaded for mapping properties value.
`open_edb_after_build`	Either if open the Edb after the build or not.
`setup_name`	Retrieve setup name for the simulation.
`solver_type`	Retrieve the SolverType class to select the solver to be called during the project build.

3.8.2 pyedb.dotnet.edb_core.edb_data.simulation_configuration.SimulationConfigurationDc

class pyedb.dotnet.edb_core.edb_data.simulation_configuration.**SimulationConfigurationDc**

Contains all DC analysis settings. The class is part of *SimulationConfiguration* class as a property.

__init__()

Attributes

dc_compute_inductance	Return the boolean for computing the inductance with SIwave DC solver.
dc_contact_radius	Retrieve the value for SIwave DC contact radius.
dc_error_energy	Retrieve the value for the DC error energy.
dc_export_thermal_data	Retrieve the value for using external data.
dc_full_report_path	Retrieve the path for the report.
dc_icepak_temp_file	Retrieve the icepak temp file path.
dc_import_thermal_data	Retrieve the value for importing thermal data.
dc_max_init_mesh_edge_length	Retrieve the maximum initial mesh edge value.
dc_max_num_pass	Retrieve the maximum number of adaptive passes.
dc_mesh_bondwires	Retrieve the value for meshing bondwires.
dc_mesh_vias	Retrieve the value for meshing vias.
dc_min_num_pass	Retrieve the minimum number of adaptive passes.
dc_min_plane_area_to_mesh	Retrieve the value of the minimum plane area to be meshed by Siwave for DC solution.
dc_min_void_area_to_mesh	Retrieve the value for the minimum void surface to mesh.
dc_num_bondwire_sides	Retrieve the number of sides used for cylinder discretization.
dc_num_via_sides	Retrieve the number of sides used for cylinder discretization.
dc_per_pin_res_path	Retrieve the file path.
dc_per_pin_use_pin_format	Retrieve the value for using pin format.
dc_percent_local_refinement	Retrieve the value for local mesh refinement.
dc_perform_adaptive_refinement	Retrieve the value for performing adaptive meshing.
dc_plot_jv	Retrieve the value for computing current density and voltage distribution.
dc_refine_bondwires	Retrieve the value for performing bond wire refinement.
dc_refine_vias	Retrieve the value for performing vias refinement.
dc_report_config_file	Retrieve the report configuration file path.
dc_report_show_Active_devices	Retrieve the value for showing active devices.
dc_slide_position	Retrieve the SIwave DC slide position value.
dc_source_terms_to_ground	Retrieve the dictionary of grounded terminals.
dc_use_dc_custom_settings	Retrieve the value for using DC custom settings.
dc_use_loop_res_for_per_pin	Retrieve the value for using the loop resistor per pin.
dc_via_report_path	Retrieve the via report file path.

3.8.3 pyedb.dotnet.edb_core.edb_data.simulation_configuration.SimulationConfigurationAc

class pyedb.dotnet.edb_core.edb_data.simulation_configuration.**SimulationConfigurationAc**

Contains all AC analysis settings. The class is part of *SimulationConfiguration* class as a property.

__init__()

Attributes

adaptive_high_freq	HFSS broadband high frequency adaptive meshing.
adaptive_low_freq	HFSS broadband low frequency adaptive meshing.
adaptive_type	HFSS adaptive type.
arc_angle	Retrieve the value for the HFSS meshing arc angle.
arc_to_chord_error	Retrieve the value of arc to chord error for HFSS meshing.
basis_order	Retrieve the BasisOrder object.
decade_count	Retrieve decade count number for the frequency sweep in case of a log sweep selected.
defeature_abs_length	Retrieve the value of arc to chord for HFSS meshing.
defeature_layout	Retrieve the boolean to activate the layout defeaturing.This method has been developed to simplify polygons with reducing the number of points to simplify the meshing with controlling its surface deviation.
do_lambda_refinement	Retrieve boolean to activate the lambda refinement.
enforce_causality	Retrieve boolean to enforce causality for the frequency sweep.
enforce_passivity	Retrieve boolean to enforce passivity for the frequency sweep.
ignore_non_functional_pads	Boolean to ignore nonfunctional pads with Siwave.
include_inter_plane_coupling	Boolean to activate the inter-plane coupling with Siwave.
max_arc_points	Retrieve the value of the maximum arc points number for the HFSS meshing.
max_mag_delta_s	Retrieve the magnitude of the delta S convergence criteria for the interpolating sweep.
max_num_passes	Retrieve maximum of points for the HFSS adaptive meshing.
max_suf_dev	Retrieve the value for the maximum surface deviation for the layout defeaturing.
mesh_freq	Retrieve the meshing frequency for the HFSS adaptive convergence.
mesh_sizefactor	Retrieve the Mesh Size factor value.
min_num_passes	Retrieve the minimum number of adaptive passes for HFSS convergence.
min_pad_area_to_mesh	Retrieve the value of minimum pad area to be meshed by Siwave.
min_plane_area_to_mesh	Retrieve the minimum plane area to be meshed by Siwave.
min_void_area	Retrieve the value of minimum void area to be considered by Siwave.
minimum_void_surface	Retrieve the minimum void surface to be considered for the layout defeaturing.
passivity_tolerance	Retrieve the value for the passivity tolerance when used.
percentage_error_z0	Retrieve boolean to perform the cutout during the project build.
process_padstack_definitions	Retrieve the boolean for activating the padstack definition processing.

continues on next page

Table 4 – continued from previous page

radiation_box	Retrieve RadiationBoxType object selection defined for the radiation box type.
relative_error	Retrieve relative error used for the interpolating sweep convergence.
return_current_distribution	Boolean to activate the current distribution return with Siwave.
snap_length_threshold	Retrieve the boolean to activate the snapping threshold feature.
start_azimuth	Retrieve the value of the starting azimuth for the HFSS meshing.
start_freq	Starting frequency for the frequency sweep.
step_freq	Retrieve step frequency for the frequency sweep.
stop_freq	Retrieve stop frequency for the frequency sweep.
sweep_interpolating	Retrieve boolean to add a sweep interpolating sweep.
sweep_name	Retrieve frequency sweep name.
sweep_type	Retrieve SweepType object for the frequency sweep.
use_arc_to_chord_error	Retrieve the boolean for activating the arc to chord for HFSS meshing.
use_error_z0	Retrieve value for the error on Z0 for the port.
use_q3d_for_dc	Retrieve boolean to Q3D solver for DC point value computation.
xtalk_threshold	Return the value for Siwave cross talk threshold.

3.8.4 pyedb.dotnet.edb_core.edb_data.simulation_configuration.SimulationConfigurationBatch

class
pyedb.dotnet.edb_core.edb_data.simulation_configuration.**SimulationConfigurationBatch**

Contains all Cutout and Batch analysis settings. The class is part of *SimulationConfiguration* class as a property.

__init__()

Methods

add_source([source])	Add a new source to configuration.

Attributes

add_frequency_sweep	Activate the frequency sweep creation when build project with the class.
airbox_horizontal_extent	Horizontal extent of the airbox for HFSS.
airbox_negative_vertical_extent	Negative vertical extent of the airbox for HFSS.
airbox_positive_vertical_extent	Positive vertical extent of the airbox for HFSS.
coax_solder_ball_diameter	Retrieve the list of solder balls diameter values when the auto evaluated one is overwritten.
components	Retrieve the list component name to be included in the simulation.

continues on next page

Table 5 – continued from previous page

coplanar_instances	Retrieve the list of component to be replaced by circuit ports (obsolete).
cutout_subdesign_expansion	Retrieve expansion factor used for clipping the design.
cutout_subdesign_round_corner	Retrieve boolean to perform the design clipping using round corner for the extent generation.
cutout_subdesign_type	Retrieve the CutoutSubdesignType selection for clipping the design.
dielectric_extent	Retrieve the value of dielectric extent.
do_cutout_subdesign	Retrieve boolean to perform the cutout during the project build.
do_pingroup	Do pingroup on multi-pin component.
etching_factor_instances	Retrieve the list of etching factor with associated layers.
generate_excitations	Activate ports and sources for DC generation when build project with the class.
generate_solder_balls	Retrieve the boolean for applying solder balls.
honor_user_dielectric	Retrieve the boolean to activate the feature "'Honor user dielectric'".
include_only_selected_nets	Include only net selection in the project.
output_aedb	Retrieve the path for the output aedb folder.
power_nets	Retrieve the list of power and reference net names.
signal_layer_etching_instances	Retrieve the list of layers which has layer etching activated.
signal_layers_properties	Retrieve the list of layers to have properties changes.
signal_nets	Retrieve the list of signal net names.
sources	Retrieve the source list.
trim_reference_size	Retrieve the trim reference size when used.
truncate_airbox_at_ground	Retrieve the boolean to truncate hfss air box at ground.
use_airbox_horizontal_extent_multiple	Whether the multiple value is used for the horizontal extent of the air box.
use_airbox_negative_vertical_extent_multi]	Multiple value for the negative extent of the airbox.
use_airbox_positive_vertical_extent_multi]	Whether the multiple value for the positive extent of the airbox is used.
use_default_coax_port_radial_extension	Retrieve the boolean for using the default coaxial port extension value.
use_default_cutout	Whether to use the default EDB cutout.
use_dielectric_extent_multiple	Whether the multiple value of the dielectric extent is used.
use_pyaedt_cutout	Whether the default EDB cutout or a new PyAEDT cutout is used.
use_radiation_boundary	Retrieve the boolean to use radiation boundary with HFSS.

```python
from pyedb import Edb

edbapp = Edb(myedb, edbversion="2023.1")

sim_setup = edbapp.new_simulation_configuration()
sim_setup.solver_type = sim_setup.SOLVER_TYPE.SiwaveSYZ
```

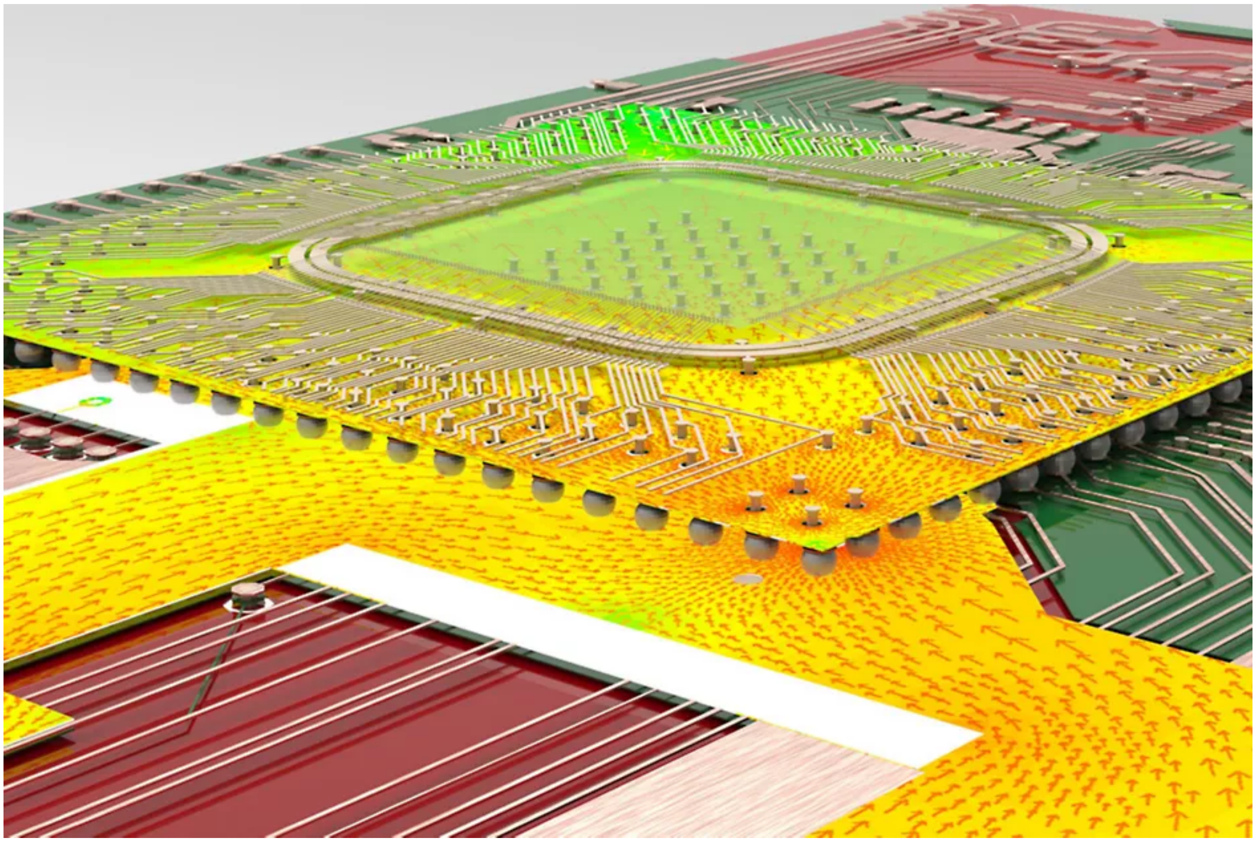
(continues on next page)

(continued from previous page)

```
sim_setup.batch_solve_settings.cutout_subdesign_expansion = 0.01
sim_setup.batch_solve_settings.do_cutout_subdesign = True
sim_setup.use_default_cutout = False
sim_setup.batch_solve_settings.signal_nets = [
    "PCIE0_RX0_P",
    "PCIE0_RX0_N",
    "PCIE0_TX0_P_C",
    "PCIE0_TX0_N_C",
    "PCIE0_TX0_P",
    "PCIE0_TX0_N",
]
sim_setup.batch_solve_settings.components = ["U2A5", "J2L1"]
sim_setup.batch_solve_settings.power_nets = ["GND"]
sim_setup.ac_settings.start_freq = "100Hz"
sim_setup.ac_settings.stop_freq = "6GHz"
sim_setup.ac_settings.step_freq = "10MHz"

sim_setup.export_json(os.path.join(project_path, "configuration.json"))
edbapp.build_simulation_project(sim_setup)

...
```

3.9 Siwave manager

Siwave is a specialized tool for power integrity, signal integrity, and EMI analysis of IC packages and PCB. This tool solves power delivery systems and high-speed channels in electronic devices. It can be accessed from PyEDB in Windows only. All setups can be implemented through EDB API.



Siwave([specified_version])	Initializes SIwave based on the inputs provided and manages SIwave release and closing.

3.9.1 pyedb.siwave.Siwave

class pyedb.siwave.**Siwave**(*specified_version=None*)

Initializes SIwave based on the inputs provided and manages SIwave release and closing.

Parameters

specified_version
[str, int, float, optional] Version of AEDT to use. The default is None, in which case the active setup is used or the latest installed version is used.

__init__(*specified_version=None*)

Methods

close_project([save_project])	Close the project.
export_dc_simulation_report(simulation_name, ...)	Export the Siwave DC simulation report.
export_element_data(simulation_name, file_path)	Export element data.
export_icepak_project(file_path, ...)	Exports an Icepak project for standalone use.
export_siwave_report(simulation_name, file_path)	Export the Siwave report.
open_project([proj_path])	Open a project.
quit_application()	Quit the application.
run_dc_simulation([...])	Run DC simulation.
run_icepak_simulation(...)	Runs an Icepak simulation.
save_project([projectpath, projectName])	Save the project.

Attributes

current_version	Current version of AEDT.
icepak	
lock_file	Lock file.
oproject	Project.
project_file	Project file.
project_name	Project name.
project_path	Project path.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
src_dir	Source directory.
version_keys	Version keys for AEDT.

```python
from pyedb.siwave import Siwave

# this call returns the Edb class initialized on 2023 R1
siwave = Siwave("2024.1")
siwave.open_project("pyproject.siw")
siwave.export_element_data("mydata.txt")
siwave.close_project()
...
```

3.10 Ports

These classes are the containers of ports methods of the EDB for both HFSS and Siwave.

```
from pyedb import Edb

edb = Edb(myedb, edbversion="2023.1")

# this call returns the EDB excitations dictionary
edb.ports
...
```

GapPort	Manages gap port properties.
WavePort	Manages wave port properties.

3.10.1 pyedb.dotnet.edb_core.edb_data.ports.GapPort

class pyedb.dotnet.edb_core.edb_data.ports.**GapPort**(*pedb*, *edb_object*)

Manages gap port properties.

Parameters

pedb
[pyedb.edb.Edb] EDB object from the Edblib library.

edb_object
[Ansys.Ansoft.Edb.Cell.Terminal.EdgeTerminal] Edge terminal instance from EDB.

Examples

This example shows how to access the GapPort class. >>> from pyedb import Edb >>> edb = Edb(myaedb.aedb) >>> gap_port = edb.ports[gap_port]

__init__(*pedb*, *edb_object*)

Methods

couple_ports(port)	Create a bundle wave port.
delete()	Delete this primitive.
get_edge_terminal_reference_primitive()	Check and return a primitive instance that serves Edge ports, wave ports and coupled edge ports that are directly connedted to primitives.
get_pad_edge_terminal_reference_pin([...])	Get the closest pin padstack instances and serves any edge terminal connected to a pad.
get_padstack_terminal_reference_pin([...])	Get a list of pad stacks instances and serves Coax wave ports, pingroup terminals, PadEdge terminals.
get_pin_group_terminal_reference_pin([...])	Return a list of pins and serves terminals connected to pingroups.
get_point_terminal_reference_primitive()	Find and return the primitive reference for the point terminal or the padstack instance.

Attributes

boundary_type	Boundary type.
bounding_box	Bounding box.
component	Component connected to this object.
deembed	Inductance value of the deembed gap port.
do_renormalize	Determine whether port renormalization is enabled.
hfss_type	HFSS port type.
id	Primitive ID.
impedance	Impedance of the port.
is_circuit_port	Whether it is a circuit port.
is_null	Flag indicating if this object is null.
is_reference_terminal	Whether it is a reference terminal.
layer	Get layer of the terminal.
location	Location of the terminal.
magnitude	Magnitude.
name	Port Name.
net	Net Object.
net_name	Net name.
phase	Phase.
ref_terminal	Get reference terminal.
reference_net_name	Net name to which reference_object belongs.
reference_object	This returns the object assigned as reference.
renormalize	Whether renormalize is active.
renormalize_z0	Renormalize Z0 value (real, imag).
terminal_type	Terminal Type.
type	Type of the edb object.

3.10.2 pyedb.dotnet.edb_core.edb_data.ports.WavePort

class pyedb.dotnet.edb_core.edb_data.ports.**WavePort**(*pedb*, *edb_terminal*)

Manages wave port properties.

Parameters

pedb
[pyedb.edb.Edb] EDB object from the Edblib library.

edb_object
[Ansys.Ansoft.Edb.Cell.Terminal.EdgeTerminal] Edge terminal instance from EDB.

Examples

This example shows how to access the WavePort class.

```
>>> from pyedb import Edb
>>> edb = Edb("myaedb.aedb")
>>> exc = edb.ports
```

__init__(*pedb*, *edb_terminal*)

Methods

couple_ports(port)	Create a bundle wave port.
delete()	Delete this primitive.
get_edge_terminal_reference_primitive()	Check and return a primitive instance that serves Edge ports, wave ports and coupled edge ports that are directly connedted to primitives.
get_pad_edge_terminal_reference_pin([...])	Get the closest pin padstack instances and serves any edge terminal connected to a pad.
get_padstack_terminal_reference_pin([...])	Get a list of pad stacks instances and serves Coax wave ports, pingroup terminals, PadEdge terminals.
get_pin_group_terminal_reference_pin([...])	Return a list of pins and serves terminals connected to pingroups.
get_point_terminal_reference_primitive()	Find and return the primitive reference for the point terminal or the padstack instance.

Attributes

boundary_type	Boundary type.
bounding_box	Bounding box.
component	Component connected to this object.
deembed	Whether deembed is active.
deembed_length	Deembed Length.
do_renormalize	Determine whether port renormalization is enabled.
hfss_type	HFSS port type.
horizontal_extent_factor	Horizontal extent factor.
id	Primitive ID.
impedance	Impedance of the port.
is_circuit_port	Whether it is a circuit port.
is_null	Flag indicating if this object is null.
is_reference_terminal	Whether it is a reference terminal.
layer	Get layer of the terminal.
location	Location of the terminal.
name	Port Name.
net	Net Object.
net_name	Net name.
pec_launch_width	Launch width for the printed electronic component (PEC).
ref_terminal	Get reference terminal.
reference_net_name	Net name to which reference_object belongs.
reference_object	This returns the object assigned as reference.
terminal_type	Terminal Type.
type	Type of the edb object.
vertical_extent_factor	Vertical extent factor.

CHAPTER

FOUR

EXAMPLES

End-to-end examples show how you can use PyEDB. If PyEDB is installed on your machine, you can download these examples as Python files or Jupyter notebooks and run them locally.

Note: Some examples require additional Python packages.

4.1 AEDT integration

The following examples illustrate the use of the legacy PyEDB API with PyAEDT.

Note: In the examples, PyAEDT is configured not to display AEDT (*non-graphical=True*).

4.2 Standalone

The following examples illustrate the use of the legacy PyEDB API as a standalone package.

4.2.1 AEDT integration

The following examples illustrate the use of the legacy PyEDB API with PyAEDT.

Note: In the examples, PyAEDT is configured not to display AEDT (*non-graphical=True*).

EDB: 5G linear array antenna

This example shows how you can use HFSS 3D Layout to create and solve a 5G linear array antenna.

Perform required imports

Perform required imports.

```
import os
import tempfile

from pyaedt import Hfss3dLayout

import pyedb
from pyedb.generic.general_methods import generate_unique_name
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False


class Patch:
    def __init__(self, width=0.0, height=0.0, position=0.0):
        self.width = width
        self.height = height
        self.position = position

    @property
    def points(self):
        return [
            [self.position, -self.height / 2],
            [self.position + self.width, -self.height / 2],
            [self.position + self.width, self.height / 2],
            [self.position, self.height / 2],
        ]


class Line:
    def __init__(self, length=0.0, width=0.0, position=0.0):
        self.length = length
        self.width = width
        self.position = position

    @property
    def points(self):
        return [
            [self.position, -self.width / 2],
            [self.position + self.length, -self.width / 2],
            [self.position + self.length, self.width / 2],
            [self.position, self.width / 2],
        ]


class LinearArray:
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, nb_patch=1, array_length=10e-3, array_width=5e-3):
        self.nbpatch = nb_patch
        self.length = array_length
        self.width = array_width

    @property
    def points(self):
        return [
            [-1e-3, -self.width / 2 - 1e-3],
            [self.length + 1e-3, -self.width / 2 - 1e-3],
            [self.length + 1e-3, self.width / 2 + 1e-3],
            [-1e-3, self.width / 2 + 1e-3],
        ]


tmpfold = tempfile.gettempdir()
aedb_path = os.path.join(tmpfold, generate_unique_name("pcb") + ".aedb")
print(aedb_path)
edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\03_5G_
↪antenna_example.py", line 101, in <module>
    edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Add stackup layers

Add the stackup layers.

```
if edb:
    edb.stackup.add_layer("Virt_GND")
    edb.stackup.add_layer("Gap", "Virt_GND", layer_type="dielectric", thickness="0.05mm",
↪ material="Air")
    edb.stackup.add_layer("GND", "Gap")
    edb.stackup.add_layer("Substrat", "GND", layer_type="dielectric", thickness="0.5mm",␣
↪material="Duroid (tm)")
    edb.stackup.add_layer("TOP", "Substrat")
```

Create linear array

Create the first patch of the linear array.

```
first_patch = Patch(width=1.4e-3, height=1.2e-3, position=0.0)
edb.modeler.create_polygon(first_patch.points, "TOP", net_name="Array_antenna")
# First line
first_line = Line(length=2.4e-3, width=0.3e-3, position=first_patch.width)
edb.modeler.create_polygon(first_line.points, "TOP", net_name="Array_antenna")
```

Patch linear array

Patch the linear array.

```
patch = Patch(width=2.29e-3, height=3.3e-3)
line = Line(length=1.9e-3, width=0.2e-3)
linear_array = LinearArray(nb_patch=8, array_width=patch.height)

current_patch = 1
current_position = first_line.position + first_line.length

while current_patch <= linear_array.nbpatch:
    patch.position = current_position
    edb.modeler.create_polygon(patch.points, "TOP", net_name="Array_antenna")
    current_position += patch.width
    if current_patch < linear_array.nbpatch:
        line.position = current_position
        edb.modeler.create_polygon(line.points, "TOP", net_name="Array_antenna")
        current_position += line.length
    current_patch += 1

linear_array.length = current_position
```

Add ground

Add a ground.

```
edb.modeler.create_polygon(linear_array.points, "GND", net_name="GND")
```

Add connector pin

Add a central connector pin.

```
edb.padstacks.create(padstackname="Connector_pin", holediam="100um", paddiam="0",␣
↪antipaddiam="200um")
con_pin = edb.padstacks.place(
    [first_patch.width / 4, 0],
    "Connector_pin",
    net_name="Array_antenna",
```

(continues on next page)

(continued from previous page)

```
    fromlayer="TOP",
    tolayer="GND",
    via_name="coax",
)
```

Add connector ground

Add a connector ground.

```
edb.modeler.create_polygon(first_patch.points, "Virt_GND", net_name="GND")
edb.padstacks.create("gnd_via", "100um", "0", "0")
con_ref1 = edb.padstacks.place(
    [first_patch.points[0][0] + 0.2e-3, first_patch.points[0][1] + 0.2e-3],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
con_ref2 = edb.padstacks.place(
    [first_patch.points[1][0] - 0.2e-3, first_patch.points[1][1] + 0.2e-3],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
con_ref3 = edb.padstacks.place(
    [first_patch.points[2][0] - 0.2e-3, first_patch.points[2][1] - 0.2e-3],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
con_ref4 = edb.padstacks.place(
    [first_patch.points[3][0] + 0.2e-3, first_patch.points[3][1] - 0.2e-3],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
```

Add excitation port

Add an excitation port.

```
edb.padstacks.set_solderball(con_pin, "Virt_GND", isTopPlaced=False, ballDiam=0.1e-3)
port_name = edb.padstacks.create_coax_port(con_pin)
```

Plot geometry

Plot the geometry.

```
edb.nets.plot(None)
```

Save and close Edb instance prior to opening it in Electronics Desktop.

Save EDB.

```
edb.save_edb()
edb.close_edb()
print("EDB saved correctly to {}. You can import in AEDT.".format(aedb_path))
```

Launch HFSS 3D Layout and open EDB

Launch HFSS 3D Layout and open EDB.

```
h3d = Hfss3dLayout(
    projectname=aedb_path, specified_version="2024.1", new_desktop_session=True, non_
↪graphical=non_graphical
)
```

Plot geometry

Plot the geometry. The EDB methods are also accessible from the `Hfss3dlayout` class.

```
h3d.modeler.edb.nets.plot(None)
```

Create setup and sweeps

Getters and setters facilitate the settings on the nested property dictionary. Previously, you had to use these commands:

`setup.props["AdaptiveSettings"]["SingleFrequencyDataList"]["AdaptiveFrequencyData"]["AdaptiveFrequency"] = "20GHz"` `setup.props["AdaptiveSettings"]["SingleFrequencyDataList"]["AdaptiveFrequencyData"]["MaxPasses"] = 4`

You can now use the simpler approach that follows.

```
setup = h3d.create_setup()

setup["AdaptiveFrequency"] = "20GHz"
setup["AdaptiveSettings/SingleFrequencyDataList/AdaptiveFrequencyData/MaxPasses"] = 4
h3d.create_linear_count_sweep(
    setupname=setup.name,
    unit="GHz",
    freqstart=20,
    freqstop=50,
    num_of_freq_points=1001,
```

(continues on next page)

(continued from previous page)

```
    sweepname="sweep1",
    sweep_type="Interpolating",
    interpolation_tol_percent=1,
    interpolation_max_solutions=255,
    save_fields=False,
    use_q3d_for_dc=False,
)
```

Solve setup and create report

Solve the project and create a report.

```
h3d.analyze()
h3d.post.create_report(["db(S({0},{1}))".format(port_name, port_name)])
```

Plot results outside AEDT

Plot results using Matplotlib.

```
solution = h3d.post.get_solution_data(["S({0},{1})".format(port_name, port_name)])
solution.plot()
```

Close AEDT

After the simulation completes, you can close AEDT or release it using the `dotnet.Desktop.release_desktop()` method. All methods provide for saving the project before closing AEDT.

```
h3d.save_project()
h3d.release_desktop()
```

Total running time of the script: (0 minutes 0.328 seconds)

EDB: Layout Components

This example shows how you can use EDB to create a layout component parametrics and use it in HFSS 3D.

Perform required imports

Perform required imports, which includes importing the `Hfss3dlayout` object and initializing it on version 2024 R1.

```
import os
import tempfile

from pyaedt import Hfss

import pyedb
from pyedb.generic.general_methods import generate_unique_name
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False
```

Creating data classes

Data classes are useful to do calculations and store variables. We create 3 Data classes for Patch, Line and Array

```
class Patch:
    def __init__(self, width=0.0, height=0.0, position=0.0):
        self.width = width
        self.height = height
        self.position = position

    @property
    def points(self):
        return [
            [self.position, "-{}/2".format(self.height)],
            ["{} + {}".format(self.position, self.width), "-{}/2".format(self.height)],
            ["{} + {}".format(self.position, self.width), "{}/2".format(self.height)],
            [self.position, "{}/2".format(self.height)],
        ]


class Line:
    def __init__(self, length=0.0, width=0.0, position=0.0):
        self.length = length
        self.width = width
        self.position = position

    @property
    def points(self):
        return [
            [self.position, "-{}/2".format(self.width)],
            ["{} + {}".format(self.position, self.length), "-{}/2".format(self.width)],
            ["{} + {}".format(self.position, self.length), "{}/2".format(self.width)],
            [self.position, "{}/2".format(self.width)],
        ]


class LinearArray:
    def __init__(self, nb_patch=1, array_length=10e-3, array_width=5e-3):
        self.nbpatch = nb_patch
        self.length = array_length
        self.width = array_width

    @property
    def points(self):
        return [
            [-1e-3, "-{}/2-1e-3".format(self.width)],
```

(continues on next page)

(continued from previous page)

```
            ["{}+1e-3".format(self.length), "-{}/2-1e-3".format(self.width)],
            ["{}+1e-3".format(self.length), "{}/2+1e-3".format(self.width)],
            [-1e-3, "{}/2+1e-3".format(self.width)],
        ]
```

Launch EDB

PyEDB.dotnet.Edb allows to open existing Edb project or create a new empty project.

```
tmpfold = tempfile.gettempdir()
aedb_path = os.path.join(tmpfold, generate_unique_name("pcb") + ".aedb")
print(aedb_path)
edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\03_5G_
↪antenna_example_parametrics.py", line 114, in <module>
    edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Add stackup layers

Add the stackup layers.

```
edb.stackup.add_layer("Virt_GND")
edb.stackup.add_layer("Gap", "Virt_GND", layer_type="dielectric", thickness="0.05mm",␣
↪material="Air")
edb.stackup.add_layer("GND", "Gap")
edb.stackup.add_layer("Substrat", "GND", layer_type="dielectric", thickness="0.5mm",␣
↪material="Duroid (tm)")
edb.stackup.add_layer("TOP", "Substrat")
```

Create linear array

Create the first patch of the linear array.

```
edb["w1"] = 1.4e-3
edb["h1"] = 1.2e-3
edb["initial_position"] = 0.0
edb["l1"] = 2.4e-3
edb["trace_w"] = 0.3e-3
first_patch = Patch(width="w1", height="h1", position="initial_position")
edb.modeler.create_polygon(first_patch.points, "TOP", net_name="Array_antenna")
# First line

first_line = Line(length="l1", width="trace_w", position=first_patch.width)
edb.modeler.create_polygon(first_line.points, "TOP", net_name="Array_antenna")
```

Patch linear array

Patch the linear array.

```
edb["w2"] = 2.29e-3
edb["h2"] = 3.3e-3
edb["l2"] = 1.9e-3
edb["trace_w2"] = 0.2e-3

patch = Patch(width="w2", height="h2")
line = Line(length="l2", width="trace_w2")
linear_array = LinearArray(nb_patch=8, array_width=patch.height)

current_patch = 1
current_position = "{} + {}".format(first_line.position, first_line.length)

while current_patch <= linear_array.nbpatch:
    patch.position = current_position
    edb.modeler.create_polygon(patch.points, "TOP", net_name="Array_antenna")
    current_position = "{} + {}".format(current_position, patch.width)
    if current_patch < linear_array.nbpatch:
        line.position = current_position
        edb.modeler.create_polygon(line.points, "TOP", net_name="Array_antenna")
        current_position = "{} + {}".format(current_position, line.length)
    current_patch += 1

linear_array.length = current_position
```

Add ground

Add a ground.

```
edb.modeler.create_polygon(linear_array.points, "GND", net_name="GND")
```

Add connector pin

Add a central connector pin.

```
edb.padstacks.create(padstackname="Connector_pin", holediam="100um", paddiam="0",␣
↪antipaddiam="200um")
con_pin = edb.padstacks.place(
    ["{}/4.0".format(first_patch.width), 0],
    "Connector_pin",
    net_name="Array_antenna",
    fromlayer="TOP",
    tolayer="GND",
    via_name="coax",
)
```

Add connector ground

Add a connector ground.

```
edb.modeler.create_polygon(first_patch.points, "Virt_GND", net_name="GND")
edb.padstacks.create("gnd_via", "100um", "0", "0")
edb["via_spacing"] = 0.2e-3
con_ref1 = edb.padstacks.place(
    [
        "{} + {}".format(first_patch.points[0][0], "via_spacing"),
        "{} + {}".format(first_patch.points[0][1], "via_spacing"),
    ],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
con_ref2 = edb.padstacks.place(
    [
        "{} + {}".format(first_patch.points[1][0], "-via_spacing"),
        "{} + {}".format(first_patch.points[1][1], "via_spacing"),
    ],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
con_ref3 = edb.padstacks.place(
    [
```

(continues on next page)

(continued from previous page)

```
        "{} + {}".format(first_patch.points[2][0], "-via_spacing"),
        "{} + {}".format(first_patch.points[2][1], "-via_spacing"),
    ],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
con_ref4 = edb.padstacks.place(
    [
        "{} + {}".format(first_patch.points[3][0], "via_spacing"),
        "{} + {}".format(first_patch.points[3][1], "-via_spacing"),
    ],
    "gnd_via",
    fromlayer="GND",
    tolayer="Virt_GND",
    net_name="GND",
)
```

Add excitation port

Add an excitation port.

```
edb.padstacks.set_solderball(con_pin, "Virt_GND", isTopPlaced=False, ballDiam=0.1e-3)
port_name = edb.padstacks.create_coax_port(con_pin)
```

Plot geometry

Plot the geometry.

```
edb.nets.plot()
```

Save and close Edb instance prior to opening it in Electronics Desktop.

Save EDB.

```
edb.save_edb()
edb.close_edb()
print("EDB saved correctly to {}. You can import in AEDT.".format(aedb_path))
```

Launch HFSS 3D

Launch HFSS 3D.

```
h3d = Hfss(
    specified_version="2024.1",
    new_desktop_session=True,
    close_on_exit=True,
    solution_type="Terminal",
    non_graphical=non_graphical,
)
```

Add the layout component

Hfss allows user to add Layout components (aedb) or 3D Components into a 3D Design and benefit of different functionalities like parametrization, mesh fusion and others.

```
component = h3d.modeler.insert_layout_component(aedb_path, parameter_mapping=True)
```

Edit Parameters

If a layout component is parametric, parameters can be exposed and changed in HFSS

```
component.parameters

w1_name = "{}_{}".format("w1", h3d.modeler.user_defined_component_names[0])
h3d[w1_name] = 0.0015
```

Boundaries

To run the simulation we need an airbox to which apply radiation boundaries. We dont need to create ports because are embedded in layout component.

```
h3d.modeler.fit_all()


h3d.modeler.create_air_region(130, 400, 1000, 130, 400, 300)
h3d.assign_radiation_boundary_to_objects("Region")
```

Create setup and sweeps

Getters and setters facilitate the settings on the nested property dictionary. - `setup.props['Frequency']="20GHz"`

You can now use the simpler approach that follows.

```
setup = h3d.create_setup()

setup.props["Frequency"] = "20GHz"
```

(continues on next page)

(continued from previous page)

```
setup.props["MaximumPasses"] = 2

sweep1 = setup.add_sweep()
sweep1.props["RangeStart"] = "20GHz"
sweep1.props["RangeEnd"] = "50GHz"
sweep1.update()
```

Solve setup and create report

Solve the project and create a report.

```
h3d.analyze()
```

Plot results outside AEDT

Plot results using Matplotlib.

```
trace = h3d.get_traces_for_plot()
solution = h3d.post.get_solution_data(trace[0])
solution.plot()
```

Plot Far Fields in AEDT

Plot Radiation patterns in AEDT.

```
variations = {}
variations["Freq"] = ["20GHz"]
variations["Theta"] = ["All"]
variations["Phi"] = ["All"]
h3d.insert_infinite_sphere(name="3D")


new_report = h3d.post.reports_by_category.far_field("db(RealizedGainTotal)", h3d.nominal_
↪adaptive, "3D")
new_report.variations = variations
new_report.primary_sweep = "Theta"
new_report.create("Realized2D")
```

Plot Far Fields in AEDT

Plot Radiation patterns in AEDT.

```
new_report.report_type = "3D Polar Plot"
new_report.secondary_sweep = "Phi"
new_report.create("Realized3D")
```

Plot Far Fields outside AEDT

Plot Radiation patterns outside AEDT.

```
solutions_custom = new_report.get_solution_data()
solutions_custom.plot_3d()
```

Plot E Field on nets and layers

Plot E Field on nets and layers in AEDT.

```
h3d.post.create_fieldplot_layers_nets(
    [["TOP", "Array_antenna"]],
    "Mag_E",
    intrinsics={"Freq": "20GHz", "Phase": "0deg"},
    plot_name="E_Layers",
)
```

Close AEDT

After the simulation completes, you can close AEDT or release it using the `dotnet.Desktop.release_desktop()` method. All methods provide for saving the project before closing AEDT.

```
h3d.save_project(os.path.join(tmpfold, "test_layout.aedt"))
h3d.release_desktop()
```

EDB: fully parametrized design

This example shows how you can use HFSS 3D Layout to create and solve a parametric design.

Perform required imports

Perform required imports, which includes importing the `Hfss3dlayout` object and initializing it on version 2023 R2.

```
import os

from pyaedt import Hfss3dLayout

import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False
```

Launch EDB

Launch EDB.

```
aedb_path = os.path.join(generate_unique_folder_name(), generate_unique_name("pcb") + ".
↪aedb")
print(aedb_path)
edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\04_edb_
↪parametrized_design.py", line 59, in <module>
    edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Define parameters

Define the parameters.

```
params = {
    "$ms_width": "0.4mm",
    "$sl_width": "0.2mm",
    "$ms_spacing": "0.2mm",
    "$sl_spacing": "0.1mm",
    "$via_spacing": "0.5mm",
    "$via_diam": "0.3mm",
    "$pad_diam": "0.6mm",
    "$anti_pad_diam": "0.7mm",
    "$pcb_len": "30mm",
    "$pcb_w": "5mm",
    "$x_size": "1.2mm",
    "$y_size": "1mm",
    "$corner_rad": "0.5mm",
}
```

(continues on next page)

(continued from previous page)

```
for par_name in params:
    edb.add_project_variable(par_name, params[par_name])
```

Define stackup layers

Define the stackup layers from bottom to top.

```
layers = [
    {"name": "bottom", "layer_type": "signal", "thickness": "35um", "material": "copper"}
↪,
    {"name": "diel_3", "layer_type": "dielectric", "thickness": "275um", "material":
↪"FR4_epoxy"},
    {"name": "sig_2", "layer_type": "signal", "thickness": "35um", "material": "copper"},
    {"name": "diel_2", "layer_type": "dielectric", "thickness": "275um", "material":
↪"FR4_epoxy"},
    {"name": "sig_1", "layer_type": "signal", "thickness": "35um", "material": "copper"},
    {"name": "diel_1", "layer_type": "dielectric", "thickness": "275um", "material":
↪"FR4_epoxy"},
    {"name": "top", "layer_type": "signal", "thickness": "35um", "material": "copper"},
]

# Create EDB stackup.
# Bottom layer
prev = None
for layer in layers:
    edb.stackup.add_layer(
        layer["name"],
        base_layer=prev,
        layer_type=layer["layer_type"],
        thickness=layer["thickness"],
        material=layer["material"],
    )
    prev = layer["name"]
```

Create padstack for signal via

Create a parametrized padstack for the signal via.

```
signal_via_padstack = "automated_via"
edb.padstacks.create(
    padstackname=signal_via_padstack,
    holediam="$via_diam",
    paddiam="$pad_diam",
    antipaddiam="",
    antipad_shape="Bullet",
    x_size="$x_size",
    y_size="$y_size",
    corner_radius="$corner_rad",
    start_layer=layers[-1]["name"],
```

(continues on next page)

(continued from previous page)

```
    stop_layer=layers[-3]["name"],
)
```

Assign net names

Assign net names. There are only two signal nets.

```
net_p = "p"
net_n = "n"
```

Place signal vias

Place signal vias.

```
edb.padstacks.place(
    position=["$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing)/2"],
    definition_name=signal_via_padstack,
    net_name=net_p,
    via_name="",
    rotation=90.0,
)

edb.padstacks.place(
    position=["2*$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing)/2"],
    definition_name=signal_via_padstack,
    net_name=net_p,
    via_name="",
    rotation=90.0,
)

edb.padstacks.place(
    position=["$pcb_len/3", "-($ms_width+$ms_spacing+$via_spacing)/2"],
    definition_name=signal_via_padstack,
    net_name=net_n,
    via_name="",
    rotation=-90.0,
)

edb.padstacks.place(
    position=["2*$pcb_len/3", "-($ms_width+$ms_spacing+$via_spacing)/2"],
    definition_name=signal_via_padstack,
    net_name=net_n,
    via_name="",
    rotation=-90.0,
)

# ###############################################################################
# Draw parametrized traces
# ~~~~~~~~~~~~~~~~~~~~~~~~
```

(continues on next page)

(continued from previous page)

```
# Draw parametrized traces.
# Trace the width and the routing (Microstrip-Stripline-Microstrip).
# Applies to both p and n nets.

width = ["$ms_width", "$sl_width", "$ms_width"]  # Trace width, n and p
route_layer = [layers[-1]["name"], layers[4]["name"], layers[-1]["name"]]  # Routing␣
↪layer, n and p

# Define points for three traces in the "p" net

points_p = [
    [
        ["0.0", "($ms_width+$ms_spacing)/2"],
        ["$pcb_len/3-2*$via_spacing", "($ms_width+$ms_spacing)/2"],
        ["$pcb_len/3-$via_spacing", "($ms_width+$ms_spacing+$via_spacing)/2"],
        ["$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing)/2"],
    ],
    [
        ["$pcb_len/3", "($ms_width+$sl_spacing+$via_spacing)/2"],
        ["$pcb_len/3+$via_spacing", "($ms_width+$sl_spacing+$via_spacing)/2"],
        ["$pcb_len/3+2*$via_spacing", "($sl_width+$sl_spacing)/2"],
        ["2*$pcb_len/3-2*$via_spacing", "($sl_width+$sl_spacing)/2"],
        ["2*$pcb_len/3-$via_spacing", "($ms_width+$sl_spacing+$via_spacing)/2"],
        ["2*$pcb_len/3", "($ms_width+$sl_spacing+$via_spacing)/2"],
    ],
    [
        ["2*$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing)/2"],
        ["2*$pcb_len/3+$via_spacing", "($ms_width+$ms_spacing+$via_spacing)/2"],
        ["2*$pcb_len/3+2*$via_spacing", "($ms_width+$ms_spacing)/2"],
        ["$pcb_len", "($ms_width+$ms_spacing)/2"],
    ],
]

# Define points for three traces in the "n" net

points_n = [
    [
        ["0.0", "-($ms_width+$ms_spacing)/2"],
        ["$pcb_len/3-2*$via_spacing", "-($ms_width+$ms_spacing)/2"],
        ["$pcb_len/3-$via_spacing", "-($ms_width+$ms_spacing+$via_spacing)/2"],
        ["$pcb_len/3", "-($ms_width+$ms_spacing+$via_spacing)/2"],
    ],
    [
        ["$pcb_len/3", "-($ms_width+$sl_spacing+$via_spacing)/2"],
        ["$pcb_len/3+$via_spacing", "-($ms_width+$sl_spacing+$via_spacing)/2"],
        ["$pcb_len/3+2*$via_spacing", "-($ms_width+$sl_spacing)/2"],
        ["2*$pcb_len/3-2*$via_spacing", "-($ms_width+$sl_spacing)/2"],
        ["2*$pcb_len/3-$via_spacing", "-($ms_width+$sl_spacing+$via_spacing)/2"],
        ["2*$pcb_len/3", "-($ms_width+$sl_spacing+$via_spacing)/2"],
    ],
    [
        ["2*$pcb_len/3", "-($ms_width+$ms_spacing+$via_spacing)/2"],
```

(continues on next page)

(continued from previous page)

```
        ["2*$pcb_len/3 + $via_spacing", "-($ms_width+$ms_spacing+$via_spacing)/2"],
        ["2*$pcb_len/3 + 2*$via_spacing", "-($ms_width+$ms_spacing)/2"],
        ["$pcb_len", "-($ms_width + $ms_spacing)/2"],
    ],
]
# ##############################################################################
# Add traces to EDB
# ~~~~~~~~~~~~~~~~~
# Add traces to EDB.

trace_p = []
trace_n = []
for n in range(len(points_p)):
    trace_p.append(edb.modeler.create_trace(points_p[n], route_layer[n], width[n], net_p,
↪ "Flat", "Flat"))
    trace_n.append(edb.modeler.create_trace(points_n[n], route_layer[n], width[n], net_n,
↪ "Flat", "Flat"))
```

Create wave ports

Create wave ports:

```
edb.hfss.create_differential_wave_port(
    trace_p[0].id,
    ["0.0", "($ms_width+$ms_spacing)/2"],
    trace_n[0].id,
    ["0.0", "-($ms_width+$ms_spacing)/2"],
    "wave_port_1",
)
edb.hfss.create_differential_wave_port(
    trace_p[2].id,
    ["$pcb_len", "($ms_width+$ms_spacing)/2"],
    trace_n[2].id,
    ["$pcb_len", "-($ms_width + $ms_spacing)/2"],
    "wave_port_2",
)
```

Draw ground polygons

Draw ground polygons.

```
gnd_poly = [[0.0, "-$pcb_w/2"], ["$pcb_len", "-$pcb_w/2"], ["$pcb_len", "$pcb_w/2"], [0.
↪0, "$pcb_w/2"]]
gnd_shape = edb.modeler.Shape("polygon", points=gnd_poly)

# Void in ground for traces on the signal routing layer

void_poly = [
    ["$pcb_len/3", "-($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/2-$via_spacing/2
```

(continues on next page)

(continued from previous page)

```
↪"],
    ["$pcb_len/3 + $via_spacing", "-($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/
↪2-$via_spacing/2"],
    ["$pcb_len/3 + 2*$via_spacing", "-($ms_width+$ms_spacing+$via_spacing+$anti_pad_
↪diam)/2"],
    ["2*$pcb_len/3 - 2*$via_spacing", "-($ms_width+$ms_spacing+$via_spacing+$anti_pad_
↪diam)/2"],
    ["2*$pcb_len/3 - $via_spacing", "-($ms_width+$ms_spacing+$via_spacing+$anti_pad_
↪diam)/2-$via_spacing/2"],
    ["2*$pcb_len/3", "-($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/2-$via_
↪spacing/2"],
    ["2*$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/2+$via_spacing/
↪2"],
    ["2*$pcb_len/3 - $via_spacing", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/
↪2+$via_spacing/2"],
    ["2*$pcb_len/3 - 2*$via_spacing", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_
↪diam)/2"],
    ["$pcb_len/3 + 2*$via_spacing", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/
↪2"],
    ["$pcb_len/3 + $via_spacing", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/2+
↪$via_spacing/2"],
    ["$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/2+$via_spacing/2
↪"],
    ["$pcb_len/3", "($ms_width+$ms_spacing+$via_spacing+$anti_pad_diam)/2"],
]

void_shape = edb.modeler.Shape("polygon", points=void_poly)

# Add ground layers

for layer in layers[:-1:2]:
    # add void if the layer is the signal routing layer.
    void = [void_shape] if layer["name"] == route_layer[1] else []

    edb.modeler.create_polygon(main_shape=gnd_shape, layer_name=layer["name"],␣
↪voids=void, net_name="gnd")
```

Plot EDB

Plot EDB.

```
edb.nets.plot(None)
```

Save EDB

Save EDB.

```
edb.save_edb()
edb.close_edb()
```

Open EDB in AEDT

Open EDB in AEDT.

```
h3d = Hfss3dLayout(
    projectname=aedb_path, specified_version="2024.1", non_graphical=non_graphical, new_
↪desktop_session=True
)
```

Add HFSS simulation setup

Add HFSS simulation setup.

```
setup = h3d.create_setup()
setup.props["AdaptiveSettings"]["SingleFrequencyDataList"]["AdaptiveFrequencyData"][
↪"MaxPasses"] = 3

h3d.create_linear_count_sweep(
    setupname=setup.name,
    unit="GHz",
    freqstart=0,
    freqstop=10,
    num_of_freq_points=1001,
    sweepname="sweep1",
    sweep_type="Interpolating",
    interpolation_tol_percent=1,
    interpolation_max_solutions=255,
    save_fields=False,
    use_q3d_for_dc=False,
)
```

Set Differential Pairs.

Define the differential pairs to be used in the postprocessing.

```
h3d.set_differential_pair(diff_name="In", positive_terminal="wave_port_1:T1", negative_
↪terminal="wave_port_1:T2")
h3d.set_differential_pair(diff_name="Out", positive_terminal="wave_port_2:T1", negative_
↪terminal="wave_port_2:T2")
```

Start HFSS solver

Start the HFSS solver by uncommenting the `h3d.analyze()` command.

```
h3d.analyze()
```

Generate Plot

Generate the plot of differential pairs.

```
solutions = h3d.post.get_solution_data(["dB(S(In,In))", "dB(S(In,Out))"], context=
↪"Differential Pairs")
solutions.plot()

h3d.release_desktop()
```

Note that the ground nets are only connected to each other due to the wave ports. The problem with poor grounding can be seen in the S-parameters. Try to modify this script to add ground vias and eliminate the resonance.

Total running time of the script: (0 minutes 0.016 seconds)

EDB: Pin to Pin project

This example shows how you can create a project using a BOM file and configuration files. run anlasyis and get results.

Perform required imports

Perform required imports. Importing the `Hfss3dlayout` object initializes it on version 2023 R2.

```
import os

from pyaedt import Hfss3dLayout

import pyedb
from pyedb.generic.general_methods import generate_unique_folder_name
from pyedb.misc.downloads import download_file
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False
```

Download file

Download the AEDB file and copy it in the temporary folder.

```
project_path = generate_unique_folder_name()
target_aedb = download_file("edb/ANSYS-HSD_V1.aedb", destination=project_path)
print("Project folder will be", target_aedb)
```

```
Project folder will be C:\Users\ansys\AppData\Local\Temp\pyedb_prj_RXD\edb/ANSYS-HSD_V1.
↪aedb
```

Launch EDB

Launch the pyedb.Edb class, using EDB 2023 R2 and SI units.

```
edbapp = pyedb.Edb(target_aedb, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\09_
↪Configuration.py", line 67, in <module>
    edbapp = pyedb.Edb(target_aedb, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Import Definitions

A definitions file is a json containing, for each part name the model associated. Model can be RLC, Sparameter or Spice. Once imported the definition is applied to the board. In this example the json file is stored for convenience in aedb folder and has the following format: {

SParameterModel: {
GRM32_DC0V_25degC_series: ./GRM32_DC0V_25degC_series.s2p

}, SPICEModel: {

GRM32_DC0V_25degC: ./GRM32_DC0V_25degC.mod

}, Definitions: {

CAPC1005X05N: {
Component_type: Capacitor, Model_type: RLC, Res: 1, Ind: 2, Cap: 3, Is_parallel: false

}, CAPC3216X180X55ML20T25: {

Component_type: Capacitor, Model_type: SParameterModel, Model_name: GRM32_DC0V_25degC_series

}, CAPC3216X180X20ML20: {

Component_type: Capacitor, Model_type: SPICEModel, Model_name: GRM32_DC0V_25degC

}

}

}

```
edbapp.components.import_definition(os.path.join(target_aedb, "1_comp_definition.json"))
```

Import BOM

This step imports a BOM file in CSV format. The BOM contains the reference designator, part name, component type, and default value. Components not in the BOM are deactivated. In this example the csv file is stored for convenience in aedb folder.

RefDes	Part name	Type	Value
C380	CAPC1005X55X25LL05T10	Capacitor	11nF

```
edbapp.components.import_bom(
    os.path.join(target_aedb, "0_bom.csv"), refdes_col=0, part_name_col=1, comp_type_
↪col=2, value_col=3
)
```

Check Component Values

Component property allows to access all components instances and their property with getters and setters.

```
comp = edbapp.components["C1"]
comp.model_type, comp.value
```

Check Component Definition

When an s-parameter model is associated to a component it will be available in nport_comp_definition property.

```
edbapp.components.nport_comp_definition
```

Save Edb

```
edbapp.save_edb()
```

Configure Setup

This step allows to define the project. It includes:

- Definition of nets to be included into the cutout,
- Cutout details,
- Components on which to create the ports,
- Simulation settings.

```
sim_setup = edbapp.new_simulation_configuration()
sim_setup.solver_type = sim_setup.SOLVER_TYPE.SiwaveSYZ
sim_setup.batch_solve_settings.cutout_subdesign_expansion = 0.003
sim_setup.batch_solve_settings.do_cutout_subdesign = True
sim_setup.batch_solve_settings.use_pyaedt_cutout = True
sim_setup.ac_settings.max_arc_points = 6
sim_setup.ac_settings.max_num_passes = 5

sim_setup.batch_solve_settings.signal_nets = [
    "PCIe_Gen4_TX2_CAP_P",
    "PCIe_Gen4_TX2_CAP_N",
    "PCIe_Gen4_TX2_P",
    "PCIe_Gen4_TX2_N",
]
sim_setup.batch_solve_settings.components = ["U1", "X1"]
sim_setup.batch_solve_settings.power_nets = ["GND", "GND_DP"]
sim_setup.ac_settings.start_freq = "100Hz"
sim_setup.ac_settings.stop_freq = "6GHz"
sim_setup.ac_settings.step_freq = "10MHz"
```

Run Setup

This step allows to create the cutout and apply all settings.

```
sim_setup.export_json(os.path.join(project_path, "configuration.json"))
edbapp.build_simulation_project(sim_setup)
```

Plot Cutout

Plot cutout once finished.

```
edbapp.nets.plot(None, None)
```

Save and Close EDB

Edb will be saved and closed in order to be opened by Hfss 3D Layout and solved.

```
edbapp.save_edb()
edbapp.close_edb()
```

Open Aedt

Project folder aedb will be opened in AEDT Hfss3DLayout and loaded.

```
h3d = Hfss3dLayout(
    specified_version="2024.1", projectname=target_aedb, non_graphical=non_graphical,␣
↪new_desktop_session=True
)
```

Analyze

Project will be solved.

```
h3d.analyze()
```

Get Results

S Parameter data will be loaded at the end of simulation.

```
solutions = h3d.post.get_solution_data()
```

Plot Results

Plot S Parameter data.

```
solutions.plot(solutions.expressions, "db20")
```

Save and Close AEDT

Hfss3dLayout is saved and closed.

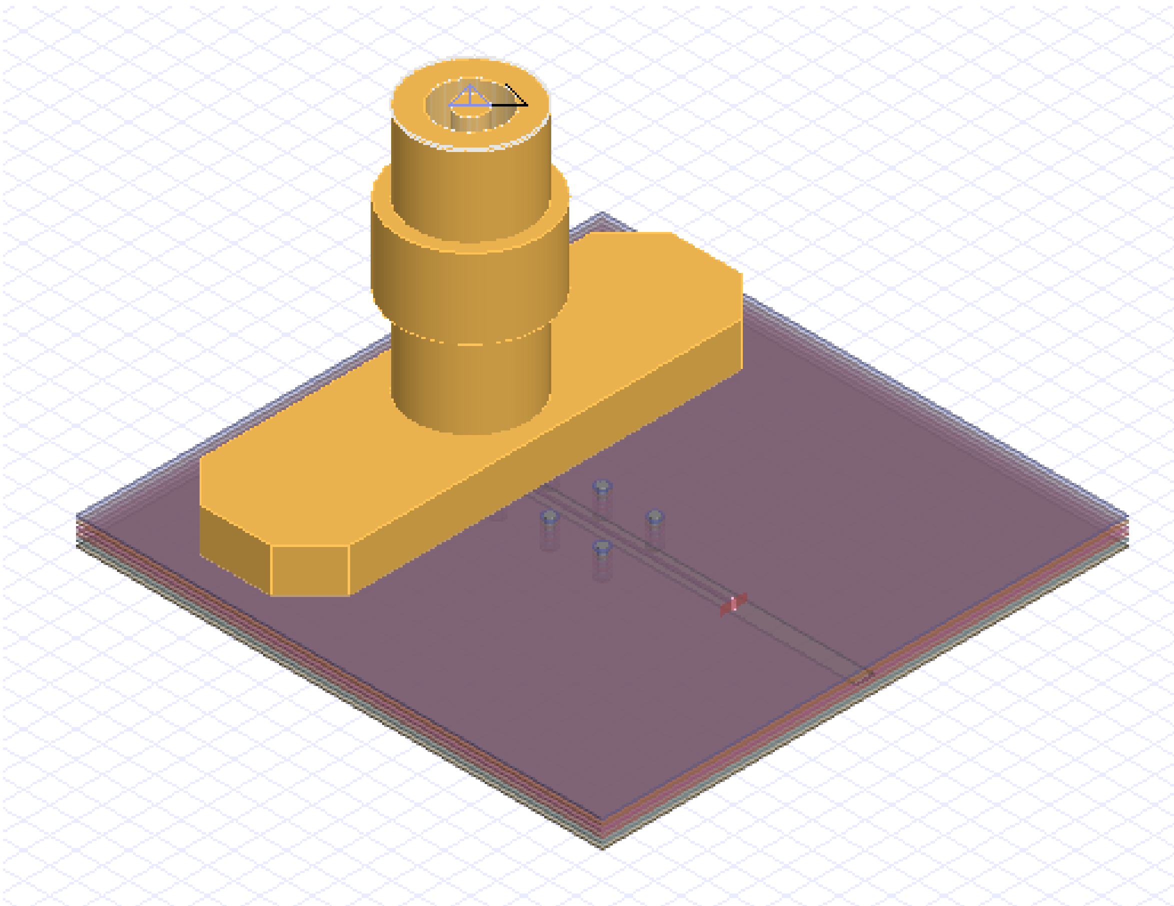
```
h3d.save_project()
h3d.release_desktop()
```

Total running time of the script: (0 minutes 0.876 seconds)

EDB: geometry creation

This example shows how to 1, Create a parameterized PCB layout design. 2, Place 3D component on PCB. 3, Create HFSS setup and frequency sweep with a mesh operation. 4, Create return loss plot

Final expected project



Create parameterized PCB
Initialize an empty EDB layout object on version 2023 R2.

```
import os

import numpy as np
from pyaedt import Hfss3dLayout

# import pyaedt
import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)
from pyedb.misc.downloads import download_file
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False
```

Launch EDB

Launch the `pyedb.Edb` class, using EDB 2023 R2.

```
aedb_path = os.path.join(generate_unique_folder_name(), generate_unique_name("pcb") + ".
↪aedb")
edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
print("EDB is located at {}".format(aedb_path))
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\12_edb_
↪sma_connector_on_board.py", line 73, in <module>
    edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Create FR4 material

```
edb.materials.add_dielectric_material("ANSYS_FR4", 3.5, 0.005)

################
# Create stackup
# ~~~~~~~~~~~~~~
# A stackup can be created by importing from a csv/xml file or adding layer by layer.
#

edb.add_design_variable("$DIEL_T", "0.15mm")
edb.stackup.add_layer("BOT")
edb.stackup.add_layer("D5", "GND", layer_type="dielectric", thickness="$DIEL_T",␣
↪material="ANSYS_FR4")
edb.stackup.add_layer("L5", "Diel", thickness="0.05mm")
edb.stackup.add_layer("D4", "GND", layer_type="dielectric", thickness="$DIEL_T",␣
↪material="ANSYS_FR4")
edb.stackup.add_layer("L4", "Diel", thickness="0.05mm")
edb.stackup.add_layer("D3", "GND", layer_type="dielectric", thickness="$DIEL_T",␣
↪material="ANSYS_FR4")
edb.stackup.add_layer("L3", "Diel", thickness="0.05mm")
edb.stackup.add_layer("D2", "GND", layer_type="dielectric", thickness="$DIEL_T",␣
↪material="ANSYS_FR4")
edb.stackup.add_layer("L2", "Diel", thickness="0.05mm")
edb.stackup.add_layer("D1", "GND", layer_type="dielectric", thickness="$DIEL_T",␣
↪material="ANSYS_FR4")
edb.stackup.add_layer("TOP", "Diel", thickness="0.05mm")
```

Create ground planes

```
edb.add_design_variable("PCB_W", "20mm")
edb.add_design_variable("PCB_L", "20mm")

gnd_dict = {}
for layer_name in edb.stackup.signal_layers.keys():
    gnd_dict[layer_name] = edb.modeler.create_rectangle(layer_name, "GND", [0, "PCB_W/-2
↪"], ["PCB_L", "PCB_W/2"])

###################
# Create signal net
# ~~~~~~~~~~~~~~~~~
# Create signal net on layer 3, and add clearance to the ground plane.

edb.add_design_variable("SIG_L", "10mm")
edb.add_design_variable("SIG_W", "0.1mm")
edb.add_design_variable("SIG_C", "0.3mm")

signal_path = (["5mm", 0], ["SIG_L+5mm", 0])
signal_trace = edb.modeler.create_trace(signal_path, "L3", "SIG_W", "SIG", "Flat", "Flat
↪")
```

(continues on next page)

(continued from previous page)

```
signal_path = (["5mm", 0], ["PCB_L", 0])
clr = edb.modeler.create_trace(signal_path, "L3", "SIG_C*2+SIG_W", "SIG", "Flat", "Flat")
gnd_dict["L3"].add_void(clr)
```

Create signal vias

Create via padstack definition. Place the signal vias.

```
edb.add_design_variable("SG_VIA_D", "1mm")

edb.add_design_variable("$VIA_AP_D", "1.2mm")

edb.padstacks.create("ANSYS_VIA", "0.3mm", "0.5mm", "$VIA_AP_D")

edb.padstacks.place(["5mm", 0], "ANSYS_VIA", "SIG")
```

Create ground vias around signal via

```
for i in np.arange(30, 331, 30):
    px = np.cos(i / 180 * np.pi)
    py = np.sin(i / 180 * np.pi)
    edb.padstacks.place(["{}*{}+5mm".format("SG_VIA_D", px), "{}*{}".format("SG_VIA_D",␣
↪py)], "ANSYS_VIA", "GND")
```

Create ground vias along signal trace

```
for i in np.arange(2e-3, edb.variables["SIG_L"].value - 2e-3, 2e-3):
    edb.padstacks.place(["{}+5mm".format(i), "1mm"], "ANSYS_VIA", "GND")
    edb.padstacks.place(["{}+5mm".format(i), "-1mm"], "ANSYS_VIA", "GND")
```

Create a wave port at the end of the signal trace

```
signal_trace.create_edge_port("port_1", "End", "Wave", horizontal_extent_factor=10)

#################
# Set hfss options
# ~~~~~~~~~~~~~~~~

edb.design_options.antipads_always_on = True
edb.hfss.hfss_extent_info.air_box_horizontal_extent = 0.01
edb.hfss.hfss_extent_info.air_box_positive_vertical_extent = 2
edb.hfss.hfss_extent_info.air_box_negative_vertical_extent = 2

##############
# Create setup
```

(continues on next page)

(continued from previous page)

```
# ~~~~~~~~~~~~

setup = edb.create_hfss_setup("Setup1")
setup.set_solution_single_frequency("5GHz", max_num_passes=2, max_delta_s="0.01")
setup.hfss_solver_settings.order_basis = "first"
```

Add mesh operation to setup

```
edb.setups["Setup1"].add_length_mesh_operation({"SIG": ["L3"]}, "m1", max_length="0.1mm")
```

Add frequency sweep to setup

```
setup.add_frequency_sweep(
    "Sweep1",
    frequency_sweep=[
        ["linear count", "0", "1KHz", 1],
        ["log scale", "1KHz", "0.1GHz", 10],
        ["linear scale", "0.1GHz", "5GHz", "0.1GHz"],
    ],
)
```

Plot EDB

Plot EDB.

```
edb.nets.plot(None)
```

Save and close EDB

```
edb.save_edb()
edb.close_edb()
```

Launch Hfss3dLayout

```
h3d = Hfss3dLayout(aedb_path, specified_version="2024.1", new_desktop_session=True, non_
↪graphical=non_graphical)
```

Place 3D component

```
component3d = download_file(
    "component_3d",
    "SMA_RF_SURFACE_MOUNT.a3dcomp",
)

comp = h3d.modeler.place_3d_component(
    component_path=component3d,
    number_of_terminals=1,
    placement_layer="TOP",
    component_name="my_connector",
    pos_x="5mm",
    pos_y=0.000,
)

##########
# Analysis
# ~~~~~~~~
h3d.analyze(num_cores=4)
```

Create return loss plot

```
h3d.post.create_report("dB(S(port_1, port_1))")
```

Save and close the project

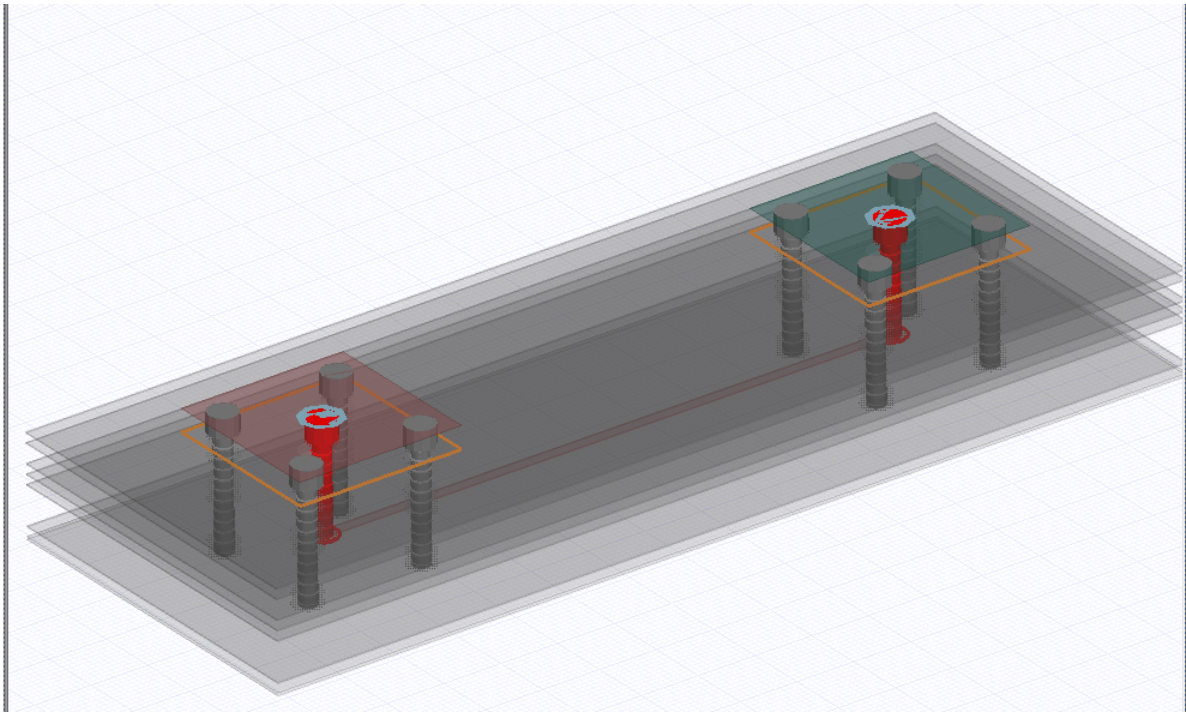
```
h3d.save_project()
print("Project is saved to {}".format(h3d.project_path))
h3d.release_desktop(True, True)
```

Total running time of the script: (0 minutes 0.016 seconds)

EDB: geometry creation

This example shows how to 1, Create a layout layer stackup. 2, Create Padstack definition. 3, Place padstack instances at given location. 4, Create primitives, polygon and trace. 5, Create component from pins. 6, Create HFSS simulation setup and excitation ports.

Final expected project



Create connector component from pad-stack
Initialize an empty EDB layout object on version 2023 R2.

```
import os

from pyaedt import Hfss3dLayout

import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False
```

Launch EDB

Launch the pyedb.Edb class, using EDB 2023 R2.

```
aedb_path = os.path.join(generate_unique_folder_name(), generate_unique_name("component_
↪example") + ".aedb")
edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
print("EDB is located at {}".format(aedb_path))
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\13_edb_
↪create_component.py", line 72, in <module>
    edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Initialize variables

```
layout_count = 12
diel_material_name = "FR4_epoxy"
diel_thickness = "0.15mm"
cond_thickness_outer = "0.05mm"
cond_thickness_inner = "0.017mm"
soldermask_thickness = "0.05mm"
trace_in_layer = "TOP"
trace_out_layer = "L10"
trace_width = "200um"
connector_size = 2e-3
conectors_position = [[0, 0], [10e-3, 0]]

################
# Create stackup
# ~~~~~~~~~~~~~~
edb.stackup.create_symmetric_stackup(
    layer_count=layout_count,
    inner_layer_thickness=cond_thickness_inner,
    outer_layer_thickness=cond_thickness_outer,
    soldermask_thickness=soldermask_thickness,
    dielectric_thickness=diel_thickness,
    dielectric_material=diel_material_name,
)
```

Create ground planes

```
ground_layers = [
    layer_name for layer_name in edb.stackup.signal_layers.keys() if layer_name not in␣
↪[trace_in_layer, trace_out_layer]
]
plane_shape = edb.modeler.Shape("rectangle", pointA=["-3mm", "-3mm"], pointB=["13mm",
↪"3mm"])
for i in ground_layers:
    edb.modeler.create_polygon(plane_shape, i, net_name="VSS")
```

Add design variables

```
edb.add_design_variable("$via_hole_size", "0.3mm")
edb.add_design_variable("$antipaddiam", "0.7mm")
edb.add_design_variable("$paddiam", "0.5mm")
edb.add_design_variable("trace_in_width", "0.2mm", is_parameter=True)
edb.add_design_variable("trace_out_width", "0.1mm", is_parameter=True)
```

Create padstack definition

```
edb.padstacks.create_padstack(
    padstackname="Via", holediam="$via_hole_size", antipaddiam="$antipaddiam", paddiam="
↪$paddiam"
)
```

Create connector 1

```
component1_pins = [
    edb.padstacks.place_padstack(
        conectors_position[0], "Via", net_name="VDD", fromlayer=trace_in_layer,␣
↪tolayer=trace_out_layer
    ),
    edb.padstacks.place_padstack(
        [conectors_position[0][0] - connector_size / 2, conectors_position[0][1] -␣
↪connector_size / 2],
        "Via",
        net_name="VSS",
    ),
    edb.padstacks.place_padstack(
        [conectors_position[0][0] + connector_size / 2, conectors_position[0][1] -␣
↪connector_size / 2],
        "Via",
        net_name="VSS",
    ),
    edb.padstacks.place_padstack(
        [conectors_position[0][0] + connector_size / 2, conectors_position[0][1] +␣
```

(continues on next page)

(continued from previous page)

```
↪connector_size / 2],
            "Via",
            net_name="VSS",
        ),
        edb.padstacks.place_padstack(
            [conectors_position[0][0] - connector_size / 2, conectors_position[0][1] +␣
↪connector_size / 2],
            "Via",
            net_name="VSS",
        ),
]
```

Create connector 2

```
component2_pins = [
    edb.padstacks.place_padstack(
        conectors_position[-1], "Via", net_name="VDD", fromlayer=trace_in_layer,␣
↪tolayer=trace_out_layer
    ),
    edb.padstacks.place_padstack(
        [conectors_position[1][0] - connector_size / 2, conectors_position[1][1] -␣
↪connector_size / 2],
        "Via",
        net_name="VSS",
    ),
    edb.padstacks.place_padstack(
        [conectors_position[1][0] + connector_size / 2, conectors_position[1][1] -␣
↪connector_size / 2],
        "Via",
        net_name="VSS",
    ),
    edb.padstacks.place_padstack(
        [conectors_position[1][0] + connector_size / 2, conectors_position[1][1] +␣
↪connector_size / 2],
        "Via",
        net_name="VSS",
    ),
    edb.padstacks.place_padstack(
        [conectors_position[1][0] - connector_size / 2, conectors_position[1][1] +␣
↪connector_size / 2],
        "Via",
        net_name="VSS",
    ),
]
```

Create layout pins

```
for padstack_instance in list(edb.padstacks.instances.values()):
    padstack_instance.is_pin = True
```

Create component from pins

```
edb.components.create(component1_pins, "connector_1")
edb.components.create(component2_pins, "connector_2")
```

Creating ports and adding simulation setup using SimulationConfiguration class

```
sim_setup = edb.new_simulation_configuration()
sim_setup.solver_type = sim_setup.SOLVER_TYPE.Hfss3dLayout
sim_setup.batch_solve_settings.cutout_subdesign_expansion = 0.01
sim_setup.batch_solve_settings.do_cutout_subdesign = False
sim_setup.batch_solve_settings.signal_nets = ["VDD"]
sim_setup.batch_solve_settings.components = ["connector_1", "connector_2"]
sim_setup.batch_solve_settings.power_nets = ["VSS"]
sim_setup.ac_settings.start_freq = "0GHz"
sim_setup.ac_settings.stop_freq = "5GHz"
sim_setup.ac_settings.step_freq = "1GHz"
edb.build_simulation_project(sim_setup)
```

Plot EDB

Plot EDB.

```
edb.nets.plot(None)
```

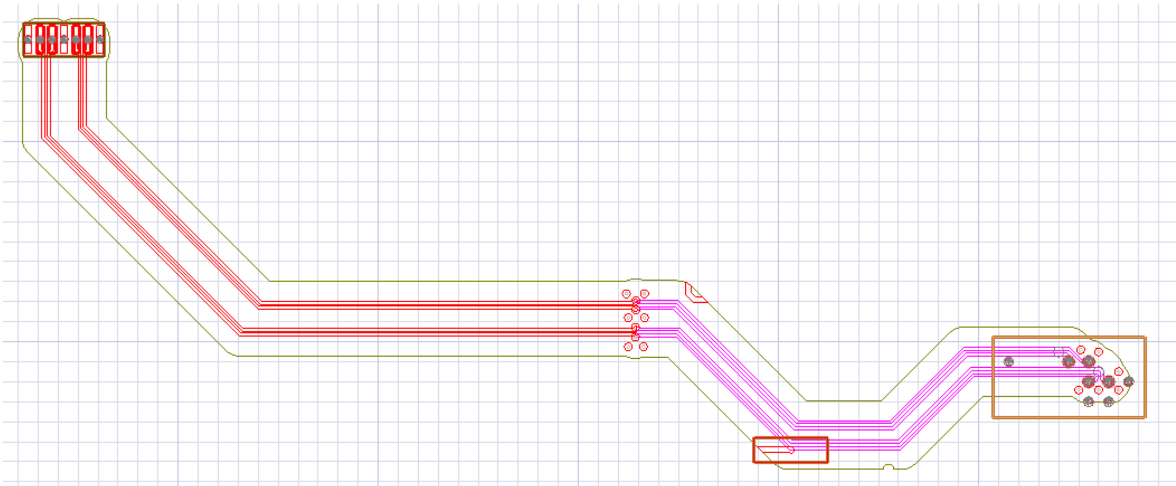
Save EDB and open in AEDT

```
edb.save_edb()
edb.close_edb()
h3d = Hfss3dLayout(
    specified_version="2024.1", projectname=aedb_path, non_graphical=non_graphical, new_
↪desktop_session=True
)
h3d.release_desktop(False, False)
```

EDB: parameterized design

This example shows how to 1, Create an HFSS simulation project using SimulationConfiguration class. 2, Create automatically parametrized design.

Final expected project



Create HFSS simulatio project

Load an existing EDB folder.

```
from pyaedt import Hfss3dLayout

import pyedb
from pyedb.generic.general_methods import generate_unique_folder_name
from pyedb.misc.downloads import download_file

project_path = generate_unique_folder_name()
target_aedb = download_file("edb/ANSYS-HSD_V1.aedb", destination=project_path)
print("Project folder will be", target_aedb)
```

```
Project folder will be C:\Users\ansys\AppData\Local\Temp\pyedb_prj_USA\edb/ANSYS-HSD_V1.
↪aedb
```

Set non-graphical mode

Set non-graphical mode. The default is False.

```
non_graphical = False
```

Launch EDB

Launch the pyedb.Edb class, using EDB 2023 R2.

```
aedt_version = "2024.1"
edb = pyedb.Edb(edbpath=target_aedb, edbversion=aedt_version)
print("EDB is located at {}".format(target_aedb))
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\14_edb_
↪create_parametrized_design.py", line 68, in <module>
    edb = pyedb.Edb(edbpath=target_aedb, edbversion=aedt_version)
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Create SimulationConfiguration object and define simulation parameters

```
simulation_configuration = edb.new_simulation_configuration()
simulation_configuration.signal_nets = ["PCIe_Gen4_RX0_P", "PCIe_Gen4_RX0_N", "PCIe_Gen4_
↪RX1_P", "PCIe_Gen4_RX1_N"]
simulation_configuration.power_nets = ["GND"]
simulation_configuration.components = ["X1", "U1"]
simulation_configuration.do_cutout_subdesign = True
simulation_configuration.start_freq = "OGHz"
simulation_configuration.stop_freq = "20GHz"
simulation_configuration.step_freq = "10MHz"
```

Build simulation project

```
edb.build_simulation_project(simulation_configuration)
```

Generated design parameters

```
edb.auto_parametrize_design(layers=True, materials=True, via_holes=True, pads=True,␣
↪antipads=True, traces=True)
```

Plot EDB

Plot EDB.

```
edb.nets.plot(None)
```

Save EDB and open in AEDT

```
edb.save_edb()
edb.close_edb()

# Uncomment the following line to open the design in HFSS 3D Layout
hfss = Hfss3dLayout(
    projectname=target_aedb, specified_version=aedt_version, non_graphical=non_graphical,
↪ new_desktop_session=True
)
hfss.release_desktop()
```

Total running time of the script: (0 minutes 0.453 seconds)

EDB: SYZ analysis

This example shows how you can use PyAEDT to set up SYZ analysis on Serdes channel. The input is the name of the differential nets. The positive net is PCIe_Gen4_TX3_CAP_P. The negative net is PCIe_Gen4_TX3_CAP_N. The code will place ports on driver and receiver components.

Perform required imports

Perform required imports, which includes importing a section.

```
import time

from pyaedt import Hfss3dLayout

import pyedb
from pyedb.generic.general_methods import generate_unique_folder_name
from pyedb.misc.downloads import download_file
```

Download file

Download the AEDB file and copy it in the temporary folder.

```
temp_folder = generate_unique_folder_name()
targetfile = download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
time.sleep(5)

print(targetfile)
```

```
C:\Users\ansys\AppData\Local\Temp\pyedb_prj_0QT\edb/ANSYS-HSD_V1.aedb
```

Configure EDB

Launch the pyedb.Edb class, using EDB 2023 R2.

```
edbapp = pyedb.Edb(edbpath=targetfile, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_pyaedt_integration\15_ac_
↪analysis.py", line 61, in <module>
    edbapp = pyedb.Edb(edbpath=targetfile, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Generate extended nets

An extended net is a connection between two nets that are usually connected through a passive component like a resistor or capacitor.

```
edbapp.extended_nets.auto_identify_signal(resistor_below=10, inductor_below=1, capacitor_
↪above=1e-9)
```

Review extended net properties

Review extended net properties.

```
diff_p = edbapp.nets["PCIe_Gen4_TX3_CAP_P"]
diff_n = edbapp.nets["PCIe_Gen4_TX3_CAP_N"]

nets_p = list(diff_p.extended_net.nets.keys())
nets_n = list(diff_n.extended_net.nets.keys())

comp_p = list(diff_p.extended_net.components.keys())
comp_n = list(diff_n.extended_net.components.keys())

rlc_p = list(diff_p.extended_net.rlc.keys())
rlc_n = list(diff_n.extended_net.rlc.keys())

print(comp_p, rlc_p, comp_n, rlc_n, sep="\n")
```

Prepare input data for port creation

Prepare input data for port creation.

```
ports = []
for net_name, net_obj in diff_p.extended_net.nets.items():
    for comp_name, comp_obj in net_obj.components.items():
        if comp_obj.type not in ["Resistor", "Capacitor", "Inductor"]:
            ports.append(
                {"port_name": "{}_{}".format(comp_name, net_name), "comp_name": comp_
↪name, "net_name": net_name}
            )

for net_name, net_obj in diff_n.extended_net.nets.items():
    for comp_name, comp_obj in net_obj.components.items():
        if comp_obj.type not in ["Resistor", "Capacitor", "Inductor"]:
            ports.append(
                {"port_name": "{}_{}".format(comp_name, net_name), "comp_name": comp_
↪name, "net_name": net_name}
            )

print(*ports, sep="\n")
```

Create ports

Solder balls are generated automatically. The default port type is coax port.

```
for d in ports:
    port_name = d["port_name"]
    comp_name = d["comp_name"]
    net_name = d["net_name"]
    edbapp.components.create_port_on_component(component=comp_name, net_list=net_name,␣
↪port_name=port_name)
```

Cutout

Delete all irrelevant nets.

```
nets = []
nets.extend(nets_p)
nets.extend(nets_n)

edbapp.cutout(signal_list=nets, reference_list=["GND"], extent_type="Bounding")
```

Create SYZ analysis setup

Create SIwave SYZ setup.

```
setup = edbapp.create_siwave_syz_setup("setup1")
setup.add_frequency_sweep(
    frequency_sweep=[
        ["linear count", "0", "1kHz", 1],
        ["log scale", "1kHz", "0.1GHz", 10],
        ["linear scale", "0.1GHz", "10GHz", "0.1GHz"],
    ]
)
```

Save and close AEDT

Close AEDT.

```
edbapp.save()
edbapp.close_edb()
```

Launch Hfss3dLayout

To do SYZ analysis, you must launch HFSS 3D Layout and import EDB into it.

```
h3d = Hfss3dLayout(targetfile, specified_version="2024.1", new_desktop_session=True)
```

Set differential pair

Set differential pair.

```
h3d.set_differential_pair(
    positive_terminal="U1_PCIe_Gen4_TX3_CAP_P", negative_terminal="U1_PCIe_Gen4_TX3_CAP_N
↪", diff_name="PAIR_U1"
)
h3d.set_differential_pair(
    positive_terminal="X1_PCIe_Gen4_TX3_P", negative_terminal="X1_PCIe_Gen4_TX3_N", diff_
↪name="PAIR_X1"
)
```

Solve and plot results

Solve and plot the results.

```
h3d.analyze(num_cores=4)
```

Create report outside AEDT

Create a report.

```
h3d.post.create_report("dB(S(PAIR_U1,PAIR_U1))", context="Differential Pairs")
```

Close AEDT

Close AEDT.

```
h3d.save_project()
print("Project is saved to {}".format(h3d.project_path))
h3d.release_desktop(True, True)
```

Total running time of the script: (0 minutes 5.438 seconds)

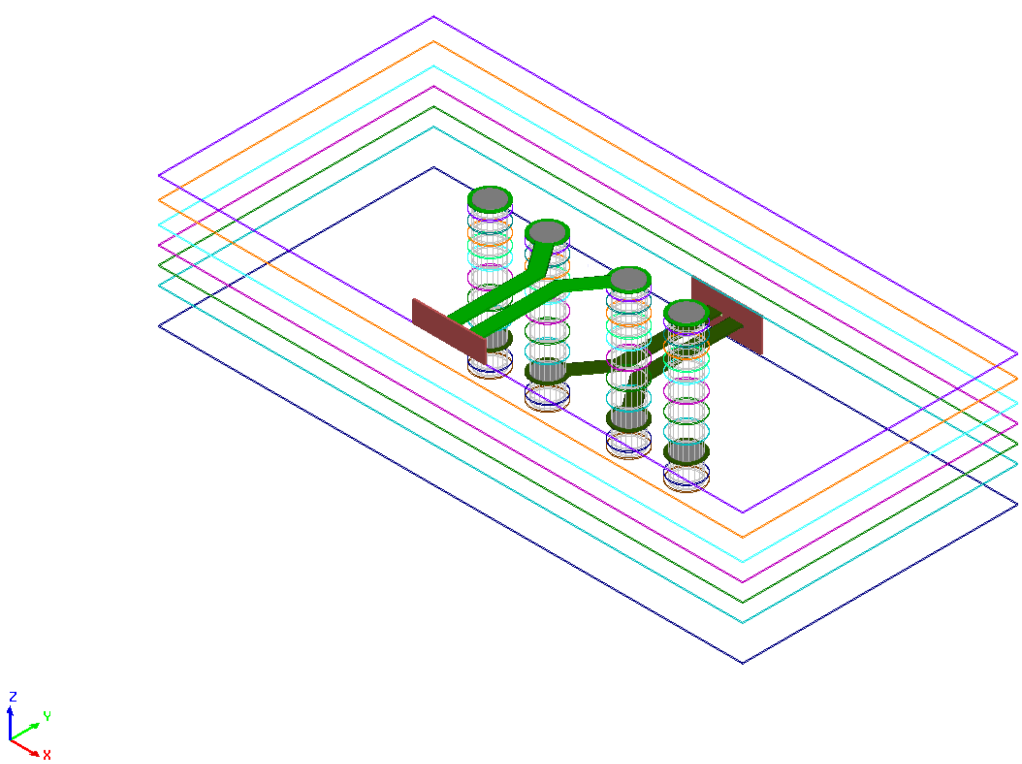
4.2.2 Standalone

The following examples illustrate the use of the legacy PyEDB API as a standalone package.

EDB: geometry creation

This example shows how you can use EDB to create a layout.

Final expected project



Import EDB layout object

Import the EDB layout object and initialize it on version 2023 R2.

```
import os
import time

import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)

start = time.time()

aedb_path = os.path.join(generate_unique_folder_name(), generate_unique_name("pcb") + ".
↪aedb")
edb = pyedb.Edb()
edb.save_edb_as(aedb_path)
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\00_EDB_Create_VIA.
↪py", line 55, in <module>
    edb = pyedb.Edb()
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Add stackup layers

Add stackup layers. A stackup can be created layer by layer or imported from a csv file or xml file.

```
edb.stackup.add_layer("GND")
edb.stackup.add_layer("Diel", "GND", layer_type="dielectric", thickness="0.1mm",␣
↪material="FR4_epoxy")
edb.stackup.add_layer("TOP", "Diel", thickness="0.05mm")
```

Create signal net and ground planes

Create a signal net and ground planes.

```
points = [
    [0.0, 0],
    [100e-3, 0.0],
]
edb.modeler.create_trace(points, "TOP", width=1e-3)
points = [[0.0, 1e-3], [0.0, 10e-3], [100e-3, 10e-3], [100e-3, 1e-3], [0.0, 1e-3]]
edb.modeler.create_polygon(points, "TOP")

points = [[0.0, -1e-3], [0.0, -10e-3], [100e-3, -10e-3], [100e-3, -1e-3], [0.0, -1e-3]]
edb.modeler.create_polygon(points, "TOP")
```

Create vias with parametric positions

Create vias with parametric positions.

```
edb.padstacks.create("MyVia")
edb.padstacks.place([5e-3, 5e-3], "MyVia")
edb.padstacks.place([15e-3, 5e-3], "MyVia")
edb.padstacks.place([35e-3, 5e-3], "MyVia")
edb.padstacks.place([45e-3, 5e-3], "MyVia")
edb.padstacks.place([5e-3, -5e-3], "MyVia")
edb.padstacks.place([15e-3, -5e-3], "MyVia")
edb.padstacks.place([35e-3, -5e-3], "MyVia")
edb.padstacks.place([45e-3, -5e-3], "MyVia")
```

Geometry Plot

```
edb.nets.plot(None, color_by_net=True)
```

Stackup Plot

```
edb.stackup.plot(plot_definitions="MyVia")
```

Save and close EDB

Save and close EDB.

```
if edb:
    edb.save_edb()
    edb.close_edb()
print("EDB saved correctly to {}. You can import in AEDT.".format(aedb_path))
end = time.time() - start
print(end)
```

Total running time of the script: (0 minutes 0.016 seconds)

EDB: Siwave analysis from EDB setup

This example shows how you can use EDB to interact with a layout.

Perform required imports

Perform required imports.

```
import os
import time

import pyedb
from pyedb.generic.general_methods import generate_unique_folder_name
from pyedb.misc.downloads import download_file

temp_folder = generate_unique_folder_name()
targetfile = download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)

siwave_file = os.path.join(os.path.dirname(targetfile), "ANSYS-HSD_V1.siw")
print(targetfile)
aedt_file = targetfile[:-4] + "aedt"
```

```
C:\Users\ansys\AppData\Local\Temp\pyedb_prj_FRE\edb/ANSYS-HSD_V1.aedb
```

Launch EDB

Launch the pyedb.Edb class, using EDB 2023 R2 and SI units.

```
edb_version = "2024.1"
if os.path.exists(aedt_file):
    os.remove(aedt_file)
edb = pyedb.Edb(edbpath=targetfile, edbversion=edb_version)
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\01_edb_example.py
↪", line 55, in <module>
    edb = pyedb.Edb(edbpath=targetfile, edbversion=edb_version)
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Compute nets and components

Computes nets and components. There are queries for nets, stackups, layers, components, and geometries.

```
print("Nets {}".format(len(edb.nets.netlist)))
start = time.time()
print("Components {}".format(len(edb.components.components.keys())))
print("elapsed time = ", time.time() - start)
```

Get pin position

Get the position for a specific pin. The next section shows how to get all pins for a specific component and the positions of each of them. Each pin is a list of `[X, Y]` coordinate positions.

```
pins = edb.components["U2"].pins
for pin in edb.components["U2"].pins.values():
    print(pin.position)
```

Get all nets connected to a component

Get all nets connected to a specific component.

```
edb.components.get_component_net_connection_info("U2")
```

Compute rats

Computes rats.

```
rats = edb.components.get_rats()
```

Get all DC-connected net lists through inductance

Get all DC-connected net lists through inductance. The inputs needed are ground net lists. The returned list contains all nets connected to a ground through an inductor.

```
GROUND_NETS = ["GND", "GND_DP"]
dc_connected_net_list = edb.nets.get_dcconnected_net_list(GROUND_NETS)
print(dc_connected_net_list)
```

Get power tree based on a specific net

Get the power tree based on a specific net.

```
VRM = "U1"
OUTPUT_NET = "AVCC_1V3"
powertree_df, component_list_columns, net_group = edb.nets.get_powertree(OUTPUT_NET,
↪GROUND_NETS)
for el in powertree_df:
    print(el)
```

Delete all RLCs with only one pin

Delete all RLCs with only one pin. This method provides a useful way of removing components not needed in the simulation.

```
edb.components.delete_single_pin_rlc()
```

Delete components

Delete manually one or more components.

```
edb.components.delete("C380")
```

Delete nets

Delete manually one or more nets.

```
edb.nets.delete("PDEN")
```

Get stackup limits

Get the stackup limits (top and bottom layers and elevations).

```
print(edb.stackup.limits())
```

Create voltage source and Siwave DCIR analysis

Create a voltage source and then set up a DCIR analysis.

```
edb.siwave.create_voltage_source_on_net("U1", "AVCC_1V3", "U1", "GND", 1.3, 0, "V1")
edb.siwave.create_current_source_on_net("IC2", "NetD3_2", "IC2", "GND", 1.0, 0, "I1")
setup = edb.siwave.add_siwave_dc_analysis("myDCIR_4")
setup.use_dc_custom_settings = True
setup.set_dc_slider = 0
setup.add_source_terminal_to_ground("V1", 1)
```

Save modifications

Save modifications.

```
edb.save_edb()
edb.nets.plot(None, "1_Top", plot_components_on_top=True)

siw_file = edb.solve_siwave()
```

Export Siwave Reports

Export all DC Reports quantities.

```
outputs = edb.export_siwave_dc_results(
    siw_file,
    setup.name,
)
```

Close EDB

Close EDB. After EDB is closed, it can be opened by AEDT.

```
edb.close_edb()
```

Total running time of the script: (0 minutes 0.422 seconds)

EDB: IPC2581 export

This example shows how you can use PyEDB to export an IPC2581 file.

Perform required imports

Perform required imports, which includes importing a section.

```
import os

import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)
from pyedb.misc.downloads import download_file
```

Download file

Download the AEDB file and copy it in the temporary folder.

```
temp_folder = generate_unique_folder_name()
targetfile = download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)


ipc2581_file = os.path.join(temp_folder, "Ansys_Hsd.xml")

print(targetfile)
```

```
C:\Users\ansys\AppData\Local\Temp\pyedb_prj_CZ9\edb/ANSYS-HSD_V1.aedb
```

Launch EDB

Launch the `pyedb.Edb` class, using EDB 2023 R2 and SI units.

```
edb = pyedb.Edb(edbpath=targetfile, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\02_edb_to_ipc2581.
↪py", line 63, in <module>
    edb = pyedb.Edb(edbpath=targetfile, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Parametrize net

Parametrize a net.

```
edb.modeler.parametrize_trace_width("A0_N", parameter_name=generate_unique_name("Par"),␣
↪variable_value="0.4321mm")
```

Cutout

Create a cutout.

```
signal_list = []
for net in edb.nets.netlist:
    if "PCIe" in net:
        signal_list.append(net)
power_list = ["GND"]
edb.cutout(
    signal_list=signal_list,
    reference_list=power_list,
    extent_type="ConvexHull",
    expansion_size=0.002,
    use_round_corner=False,
    number_of_threads=4,
    remove_single_pin_components=True,
    use_pyaedt_extent_computing=True,
    extent_defeature=0,
)
```

Plot cutout

Plot cutout before exporting to IPC2581 file.

```
edb.nets.plot(None, None, color_by_net=True)
```

Create IPC2581 file

Create the IPC2581 file.

```
edb.export_to_ipc2581(ipc2581_file, "inch")
print("IPC2581 File has been saved to {}".format(ipc2581_file))
```

Close EDB

Close EDB.

```
edb.close_edb()
```

Total running time of the script: (0 minutes 0.437 seconds)

EDB: Rename nets and ports

This example shows how you can use PyEDB to rename ports and nets.

Perform required imports

Perform required imports, which includes importing a section.

```
from pyedb import Edb
from pyedb.generic.general_methods import generate_unique_folder_name
import pyedb.misc.downloads as downloads
```

Download ANSYS EDB

Download ANSYS generic design from public repository.

```
temp_folder = generate_unique_folder_name()
targetfile = downloads.download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
```

opening EDB

Opening EDB with ANSYS release 2024.1

```
edbapp = Edb(edbpath=targetfile, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\03_rename_nets_
↪and_ports.py", line 52, in <module>
    edbapp = Edb(edbpath=targetfile, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Renaming all signal nets

Using the net name setter to rename.

```
for net_name, net in edbapp.nets.signal.items():
    net.name = f"{net_name}_test"
```

Creating coaxial port on component U1 and all ddr4_dqs nets

Selecting all nets from ddr4_dqs and component U1 and create coaxial ports On corresponding pins.

```
comp_u1 = edbapp.components.instances["U1"]
signal_nets = [net for net in comp_u1.nets if "ddr4_dqs" in net.lower()]
edbapp.hfss.create_coax_port_on_component("U1", net_list=signal_nets)
edbapp.components.set_solder_ball(component="U1", sball_diam="0.3mm", sball_height="0.3mm
↪")
```

Renaming all ports

Renaming all port with _renamed string as suffix example.

```
for port_name, port in edbapp.ports.items():
    port.name = f"{port_name}_renamed"
```

Saving and closing EDB

Once the EDB saved and closed, this can be imported in ANSYS AEDT as an HFSS 3D Layout project.

```
edbapp.save()
edbapp.close()
```

Total running time of the script: (0 minutes 0.422 seconds)

EDB: plot nets with Matplotlib

This example shows how you can use the Edb class to plot a net or a layout.

Perform required imports

Perform required imports, which includes importing a section.

```
import pyedb
from pyedb.generic.general_methods import generate_unique_folder_name
from pyedb.misc.downloads import download_file
```

Download file

Download the AEDT file and copy it into the temporary folder.

```
temp_folder = generate_unique_folder_name()

targetfolder = download_file("edb/ANSYS-HSD_V1.aedb", destination=temp_folder)
```

Launch EDB

Launch the `pyedb.Edb` class, using EDB 2023 R2 and SI units.

```
edb = pyedb.Edb(edbpath=targetfolder, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\05_Plot_nets.py",␣
↪line 53, in <module>
    edb = pyedb.Edb(edbpath=targetfolder, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Plot custom set of nets colored by layer

Plot a custom set of nets colored by layer (default).

```
edb.nets.plot("AVCC_1V3")
```

Plot custom set of nets colored by nets

Plot a custom set of nets colored by nets.

```
edb.nets.plot(["GND", "GND_DP", "AVCC_1V3"], color_by_net=True)
```

Plot all nets on a layer colored by nets

Plot all nets on a layer colored by nets

```
edb.nets.plot(None, ["1_Top"], color_by_net=True, plot_components_on_top=True)
```

Plot stackup and some padstack definition

Plot all nets on a layer colored by nets

```
edb.stackup.plot(scale_elevation=False, plot_definitions=["c100hn140", "c35"])
```

Close EDB

Close EDB.

```
edb.close_edb()
```

Total running time of the script: (0 minutes 0.437 seconds)

EDB: parametric via creation

This example shows how you can use EDB to create a layout.

Perform required imports

Perform required imports.

```
import os

import numpy as np

import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)

aedb_path = os.path.join(generate_unique_folder_name(), generate_unique_name("via_opt")␣
↪+ ".aedb")
```

Create stackup

The `StackupSimple` class creates a stackup based on few inputs. This stackup is used later.

Create ground plane

Create a ground plane on specific layers.

```
def _create_ground_planes(edb, layers):
    plane = edb.modeler.Shape("rectangle", pointA=["-3mm", "-3mm"], pointB=["3mm", "3mm
↪"])
    for i in layers:
        edb.modeler.create_polygon(plane, i, net_name="GND")
```

Create EDB

Create EDB. If the path doesnt exist, PyAEDT automatically generates a new AEDB folder.

```
edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\06_Advanced_EDB.py
↪", line 69, in <module>
    edb = pyedb.Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Create stackup layers

Create stackup layers.

```
layout_count = 12
diel_material_name = "FR4_epoxy"
diel_thickness = "0.15mm"
cond_thickness_outer = "0.05mm"
cond_thickness_inner = "0.017mm"
soldermask_thickness = "0.05mm"
trace_in_layer = "TOP"
trace_out_layer = "L10"
gvia_num = 10
gvia_angle = 30
```

(continues on next page)

(continued from previous page)

```
edb.stackup.create_symmetric_stackup(
    layer_count=layout_count,
    inner_layer_thickness=cond_thickness_inner,
    outer_layer_thickness=cond_thickness_outer,
    soldermask_thickness=soldermask_thickness,
    dielectric_thickness=diel_thickness,
    dielectric_material=diel_material_name,
)
```

Create variables

Create all variables. If a variable has a $ prefix, it is a project variable. Otherwise, is a design variable.

```
giva_angle_rad = gvia_angle / 180 * np.pi

edb["$via_hole_size"] = "0.3mm"
edb["$antipaddiam"] = "0.7mm"
edb["$paddiam"] = "0.5mm"
edb.add_design_variable("via_pitch", "1mm", is_parameter=True)
edb.add_design_variable("trace_in_width", "0.2mm", is_parameter=True)
edb.add_design_variable("trace_out_width", "0.1mm", is_parameter=True)
```

Create definitions

Create two definitions, one for the ground and one for the signal. The definitions are parametric.

```
edb.padstacks.create(
    padstackname="SVIA",
    holediam="$via_hole_size",
    antipaddiam="$antipaddiam",
    paddiam="$paddiam",
    start_layer=trace_in_layer,
    stop_layer=trace_out_layer,
)
edb.padstacks.create(padstackname="GVIA", holediam="0.3mm", antipaddiam="0.7mm", paddiam=
↪"0.5mm")
```

Place padstack for signal

Place the padstack for the signal.

```
edb.padstacks.place([0, 0], "SVIA", net_name="RF")
```

Place padstack for ground

Place the padstack for the ground. A loop iterates and places multiple ground vias on different positions.

```
gvia_num_side = gvia_num / 2

if gvia_num_side % 2:
    # Even number of ground vias on each side
    edb.padstacks.place(["via_pitch", 0], "GVIA", net_name="GND")
    edb.padstacks.place(["via_pitch*-1", 0], "GVIA", net_name="GND")
    for i in np.arange(1, gvia_num_side / 2):
        xloc = "{}*{}".format(np.cos(giva_angle_rad * i), "via_pitch")
        yloc = "{}*{}".format(np.sin(giva_angle_rad * i), "via_pitch")
        edb.padstacks.place([xloc, yloc], "GVIA", net_name="GND")
        edb.padstacks.place([xloc, yloc + "*-1"], "GVIA", net_name="GND")

        edb.padstacks.place([xloc + "*-1", yloc], "GVIA", net_name="GND")
        edb.padstacks.place([xloc + "*-1", yloc + "*-1"], "GVIA", net_name="GND")
else:
    # Odd number of ground vias on each side
    for i in np.arange(0, gvia_num_side / 2):
        xloc = "{}*{}".format(np.cos(giva_angle_rad * (i + 0.5)), "via_pitch")
        yloc = "{}*{}".format(np.sin(giva_angle_rad * (i + 0.5)), "via_pitch")
        edb.padstacks.place([xloc, yloc], "GVIA", net_name="GND")
        edb.padstacks.place([xloc, yloc + "*-1"], "GVIA", net_name="GND")

        edb.padstacks.place([xloc + "*-1", yloc], "GVIA", net_name="GND")
        edb.padstacks.place([xloc + "*-1", yloc + "*-1"], "GVIA", net_name="GND")
```

Generate traces

Generate and place parametric traces.

```
edb.modeler.create_trace(
    [[0, 0], [0, "-3mm"]],
    layer_name=trace_in_layer,
    net_name="RF",
    width="trace_in_width",
    start_cap_style="Flat",
    end_cap_style="Flat",
)

edb.modeler.create_trace(
    [[0, 0], [0, "3mm"]],
    layer_name=trace_out_layer,
    net_name="RF",
    width="trace_out_width",
    start_cap_style="Flat",
    end_cap_style="Flat",
)
```

Generate ground layers

Generate and place ground layers.

```
ground_layers = [i for i in edb.stackup.signal_layers.keys()]
ground_layers.remove(trace_in_layer)
ground_layers.remove(trace_out_layer)
_create_ground_planes(edb=edb, layers=ground_layers)
```

Plot Layout

Generate and plot the layout.

```
# edb.nets.plot(layers=["TOP", "L10"])
edb.stackup.plot(plot_definitions=["GVIA", "SVIA"])
```

Save EDB and close

Save EDB and close.

```
edb.save_edb()
edb.close_edb()

print("aedb Saved in {}".format(aedb_path))
```

Total running time of the script: (0 minutes 0.016 seconds)

EDB: fully parametrized CPWG design

This example shows how you can use HFSS 3D Layout to create a parametric design for a CPWG (coplanar waveguide with ground).

Perform required imports

Perform required imports. Importing the `Hfss3dlayout` object initializes it on version 2023 R2.

```
import os

import numpy as np

from pyedb import Edb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)
```

Set non-graphical mode

Set non-graphical mode. The default is `False`.

```
non_graphical = False
```

Launch EDB

Launch EDB.

```
aedb_path = os.path.join(generate_unique_folder_name(), generate_unique_name("pcb") + ".
↪aedb")
print(aedb_path)
edbapp = Edb(edbpath=aedb_path, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\08_CPWG.py", line␣
↪60, in <module>
    edbapp = Edb(edbpath=aedb_path, edbversion="2024.1")
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Define parameters

Define parameters.

```
params = {
    "$ms_width": "0.4mm",
    "$ms_clearance": "0.3mm",
    "$ms_length": "20mm",
}
for par_name in params:
    edbapp.add_project_variable(par_name, params[par_name])
```

Create stackup

Create a symmetric stackup.

```
edbapp.stackup.create_symmetric_stackup(2)
edbapp.stackup.plot()
```

Draw planes

Draw planes.

```
plane_lw_pt = ["0mm", "-3mm"]
plane_up_pt = ["$ms_length", "3mm"]

top_layer_obj = edbapp.modeler.create_rectangle(
    "TOP", net_name="gnd", lower_left_point=plane_lw_pt, upper_right_point=plane_up_pt
)
bot_layer_obj = edbapp.modeler.create_rectangle(
    "BOTTOM", net_name="gnd", lower_left_point=plane_lw_pt, upper_right_point=plane_up_pt
)
layer_dict = {"TOP": top_layer_obj, "BOTTOM": bot_layer_obj}
```

Draw trace

Draw a trace.

```
trace_path = [["0", "0"], ["$ms_length", "0"]]
edbapp.modeler.create_trace(
    trace_path, layer_name="TOP", width="$ms_width", net_name="sig", start_cap_style=
↪"Flat", end_cap_style="Flat"
)
```

Create trace to plane clearance

Create a trace to the plane clearance.

```
poly_void = edbapp.modeler.create_trace(
    trace_path,
    layer_name="TOP",
    net_name="gnd",
    width="{}+2*{}".format("$ms_width", "$ms_clearance"),
    start_cap_style="Flat",
    end_cap_style="Flat",
)
edbapp.modeler.add_void(layer_dict["TOP"], poly_void)
```

Create ground via padstack and place ground stitching vias

Create a ground via padstack and place ground stitching vias.

```
edbapp.padstacks.create(
    padstackname="GVIA",
    holediam="0.3mm",
    paddiam="0.5mm",
)

yloc_u = "$ms_width/2+$ms_clearance+0.25mm"
yloc_l = "-$ms_width/2-$ms_clearance-0.25mm"

for i in np.arange(1, 20):
    edbapp.padstacks.place([str(i) + "mm", yloc_u], "GVIA", net_name="GND")
    edbapp.padstacks.place([str(i) + "mm", yloc_l], "GVIA", net_name="GND")
```

Save and close EDB

Save and close EDB.

```
edbapp.save_edb()
edbapp.close_edb()
```

EDB: Edit Control File and import gds

This example shows how you can use PyAEDT to import a gds from an IC file.

Perform required imports

Perform required imports, which includes importing a section.

```
import os
import shutil
import tempfile

from pyedb import Edb
from pyedb.dotnet.edb_core.edb_data.control_file import ControlFile
from pyedb.misc.downloads import download_file
```

Download file

Download the AEDB file and copy it in the temporary folder.

```
temppath = tempfile.gettempdir()
local_path = download_file("gds")
c_file_in = os.path.join(local_path, "sky130_fictitious_dtc_example_control_no_map.xml")
c_map = os.path.join(local_path, "dummy_layermap.map")
gds_in = os.path.join(local_path, "sky130_fictitious_dtc_example.gds")
gds_out = os.path.join(temppath, "example.gds")
shutil.copy2(gds_in, gds_out)
```

```
'C:\\Users\\ansys\\AppData\\Local\\Temp\\example.gds'
```

Control file

A Control file is an xml file which purpose if to provide additional information during import phase. It can include, materials, stackup, setup, boundaries and settings. In this example we will import an existing xml, integrate it with a layer mapping file of gds and then adding setup and boundaries.

```
c = ControlFile(c_file_in, layer_map=c_map)
```

Simulation setup

Here we setup simulation with HFSS and add a frequency sweep.

```
setup = c.setups.add_setup("Setup1", "1GHz")
setup.add_sweep("Sweep1", "0.01GHz", "5GHz", "0.1GHz")
```

```
<pyedb.dotnet.edb_core.edb_data.control_file.ControlFileSweep object at␣
↪0x000001CE15BB6CB0>
```

Additional stackup settings

After import user can change stackup settings and add/remove layers or materials.

```
c.stackup.units = "um"
c.stackup.dielectrics_base_elevation = -100
c.stackup.metal_layer_snapping_tolerance = "10nm"
for via in c.stackup.vias:
    via.create_via_group = True
    via.snap_via_group = True
```

Boundaries settings

Boundaries can include ports, components and boundary extent.

```
c.boundaries.units = "um"
c.boundaries.add_port("P1", x1=223.7, y1=222.6, layer1="Metal6", x2=223.7, y2=100,␣
↪layer2="Metal6")
c.boundaries.add_extent()
comp = c.components.add_component("B1", "BGA", "IC", "Flip chip", "Cylinder")
comp.solder_diameter = "65um"
comp.add_pin("1", "81.28", "84.6", "met2")
comp.add_pin("2", "211.28", "84.6", "met2")
comp.add_pin("3", "211.28", "214.6", "met2")
comp.add_pin("4", "81.28", "214.6", "met2")
c.import_options.import_dummy_nets = True
```

Write xml

After all settings are ready we can write xml.

```
c.write_xml(os.path.join(temppath, "output.xml"))
```

```
True
```

Open Edb

Import the gds and open the edb.

```
edb = Edb(gds_out, edbversion="2024.1", technology_file=os.path.join(temppath, "output.
↪xml"))
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\10_GDS_workflow.py
↪", line 111, in <module>
    edb = Edb(gds_out, edbversion="2024.1", technology_file=os.path.join(temppath,
↪"output.xml"))
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Plot Stackup

Stackup plot.

```
edb.stackup.plot(first_layer="met1")
```

Close Edb

Close the project.

```
edb.close_edb()
```

Total running time of the script: (0 minutes 0.969 seconds)

EDB: post-layout parameterization

This example shows you how to parameterize the signal net in post-layout.

Define input parameters

```
signal_net_name = "DDR4_ALERT3"
coplanar_plane_net_name = "1V0"  # Specify coplanar plane net name for adding clearance
layers = ["16_Bottom"]  # Specify layers to be parameterized
```

Perform required imports

```
import os

import pyedb
from pyedb.generic.general_methods import (
    generate_unique_folder_name,
    generate_unique_name,
)
from pyedb.misc.downloads import download_file

temppath = generate_unique_folder_name()
```

Download and open example layout file in edb format

```
edb_fpath = download_file("edb/ANSYS-HSD_V1.aedb", destination=temppath)
appedb = pyedb.Edb(edb_fpath, edbversion="2024.1")
```

```
Traceback (most recent call last):
  File "C:\actions-runner\_work\pyedb\pyedb\examples\legacy_standalone\11_post_layout_
↪parameterization.py", line 54, in <module>
    appedb = pyedb.Edb(edb_fpath, edbversion="2024.1")
```

(continues on next page)

(continued from previous page)

```
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\generic\design_
↪types.py", line 108, in Edb
    from pyedb.dotnet.edb import Edb as app
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\dotnet\edb.py",
↪ line 115, in <module>
    from pyedb.workflow import Workflow
  File "C:\actions-runner\_work\pyedb\pyedb\.venv\lib\site-packages\pyedb\workflow.py",␣
↪line 4, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

Cutout

```
appedb.cutout([signal_net_name], [coplanar_plane_net_name, "GND"], remove_single_pin_
↪components=True)
```

Get all trace segments from the signal net

```
net = appedb.nets[signal_net_name]
trace_segments = []
for p in net.primitives:
    if p.layer_name not in layers:
        continue
    if not p.type == "Path":
        continue
    trace_segments.append(p)
```

Create and assign delta w variable per layer

```
for p in trace_segments:
    vname = f"{p.net_name}_{p.layer_name}_dw"
    if vname not in appedb.variables:
        appedb[vname] = "0mm"
    new_w = f"{p.width}+{vname}"
    p.width = new_w
```

Delete existing clearance

```
for p in trace_segments:
    for g in appedb.modeler.get_polygons_by_layer(p.layer_name, coplanar_plane_net_name):
        for v in g.voids:
            if p.is_intersecting(v):
                v.delete()
```

Create and assign clearance variable per layer

```
for p in trace_segments:
    clr = f"{p.net_name}_{p.layer_name}_clr"
    if clr not in appedb.variables:
        appedb[clr] = "0.5mm"
    path = p.get_center_line()
    for g in appedb.modeler.get_polygons_by_layer(p.layer_name, coplanar_plane_net_name):
        void = appedb.modeler.create_trace(path, p.layer_name, f"{p.width}+{clr}*2")
        g.add_void(void)
```

Plot

```
appedb.nets.plot(layers=layers[0], size=2000)
```

Save and close Edb

```
save_edb_fpath = os.path.join(temppath, generate_unique_name("post_layout_
↪parameterization") + ".aedb")
appedb.save_edb_as(save_edb_fpath)
print("Edb is saved to ", save_edb_fpath)
appedb.close_edb()
```

Total running time of the script: (0 minutes 0.422 seconds)

CHAPTER

FIVE

CONTRIBUTE

Overall guidance on contributing to a PyAnsys repository appears in Contribute in the *PyAnsys Developers Guide*. Ensure that you are thoroughly familiar with this guide before attempting to contribute to PyEDB.

The following contribution information is specific to PyEDB.

5.1 Clone the repository

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

```
git clone https://github.com/ansys/pyedb
cd pyedb
python -m pip install --upgrade pip
pip install -e .
```

5.2 Post issues

Use the PyEDB Issues page to submit questions, report bugs, and request new features.

To reach the product support team, email pyansys.core@ansys.com.

5.3 View PyEDB documentation

Documentation for the latest stable release of PyEDB is hosted at PyEDB 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

PyEDB is compliant with PyAnsys code style. It uses the tool pre-commit to check 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. For example:

```
$ 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 Log errors

PyEDB has an internal logging tool named `Messenger` and a log file that is automatically generated in the project folder.

The following examples show how `Messenger` is used to write both to the internal AEDT message windows and the log file:

```
self.logger.error("This is an error message.")
self.logger.warning("This is a warning message.")
self.logger.info("This is an info message.")
```

These examples show how to write messages only to the log file:

```
self.logger.error("This is an error message.")
self.logger.warning("This is a warning message.")
self.logger.info("This is an info message.")
```

5.4.2 Hard-coded values

Do not write hard-coded values to the registry. Instead, use the Configuration service.

5.4.3 Maximum line length

Best practice is to keep the 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

INDICES AND TABLES

- genindex
- modindex
- search