



© 2024 ANSYS, Inc. or affiliated companies
Unauthorized use, distribution, or duplication prohibited.

PyAEDT



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

May 31, 2024

ANSYS, Inc. and
ANSYS Europe,
Ltd. are UL
registered ISO
9001:2015
companies.

CONTENTS

Useful links: [Installation](#) | [Source Repository](#) | [Issues](#)

PyAEDT is a Python client library that interacts directly with the Ansys Electronics Desktop (AEDT) API, enabling straightforward and efficient automation in your workflow.

Note: Also consider viewing the [PyEDB documentation](#). PyEDB is a Python client library for processing complex and large layout designs in the Ansys Electronics Database (EDB) format, which stores information describing designs for AEDT.

Getting started New to PyAEDT? This section provides the information that you need to get started with PyAEDT.

Getting started User guide This section provides in-depth information on PyAEDT key concepts.

User guide API reference This section contains descriptions of the functions and modules included in PyAEDT. It describes how the methods work and the parameters that can be used.

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

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

Contribute

**CHAPTER
ONE**

GETTING STARTED

About PyAnsys and AEDT Learn more about PyAnsys and AEDT.

About PyAnsys and AEDT Installation Learn how to install PyAEDT from PyPi or Conda.

Installation User guide This section provides in-depth information on PyAEDT key concepts.

User guide Client-Server Launch PyAEDT on a client machine and control Electronics Desktop on a remote server.

Client-server Versions and interfaces Discover the compatibility between PyAEDT and Ansys AEDT versions.

Versions and interfaces Troubleshooting Any questions? Refer to Q&A before submitting an issue.

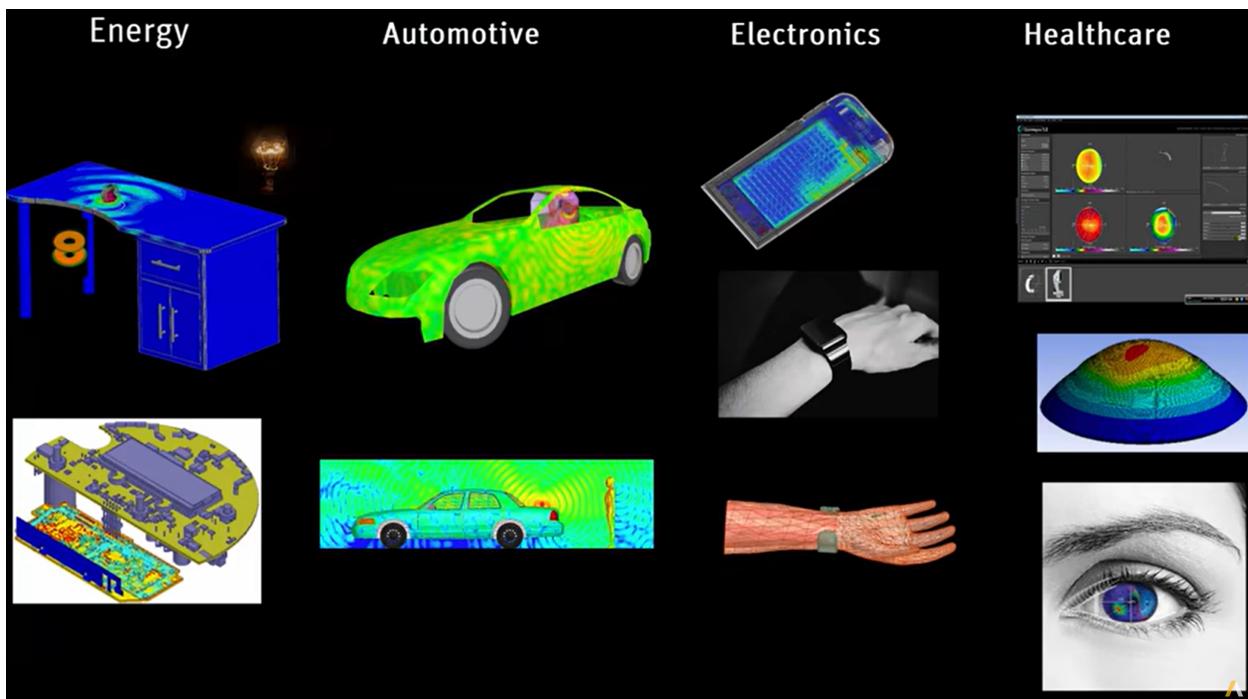
Troubleshooting

1.1 What is PyAEDT?

PyAEDT is a Python library that interacts directly with the API for Ansys Electronics Desktop (AEDT) to make scripting simpler. The architecture for PyAEDT can be reused for all AEDT 3D products (HFSS, Icepak, Maxwell 3D, and Q3D Extractor), 2D tools, and Ansys Mechanical. PyAEDT also provides support for Circuit tools like Nexxim and system simulation tools like Twin Builder. Finally, PyAEDT provides scripting capabilities in Ansys layout tools like HFSS 3D Layout and EDB. The PyAEDT class and method structures simplify operation while reusing information as much as possible across the API.

To run PyAEDT, you must have a licensed copy of Ansys Electronics Desktop (AEDT) installed.

The Ansys Electronics Desktop (AEDT) is a platform that enables true electronics system design. [AEDT](#) provides access to the Ansys gold-standard electro-magnetics simulation solutions such as Ansys HFSS, Ansys Maxwell, Ansys Q3D Extractor, Ansys Siwave, and Ansys Icepak using electrical CAD (ECAD) and Mechanical CAD (MCAD) workflows. In addition, it includes direct links to the complete Ansys portfolio of thermal, fluid, and Mechanical solvers for comprehensive multiphysics analysis. Tight integration among these solutions provides unprecedented ease of use for setup and faster resolution of complex simulations for design and optimization.



For more information, see [Ansys Electronics](#) on the Ansys website.

1.2 PyAEDT cheat sheets

PyAEDT cheat sheets introduce the basics that you need to use PyAEDT. These one-page references providing syntax rules and commands for using PyAEDT API and EDB API:

PyAEDT cheat sheet: [PyAEDT API](#)

EDB cheat sheet: [EDB API](#)

1.3 Get help

Development issues: For PyAEDT development-related matters, see the [PyAEDT Issues](#) page. You can create issues to report bugs and request new features.

User questions: The best way to get help is to post your question on the [PyAEDT Discussions](#) page or the [Discussions](#) page on the Ansys Developer portal. You can post questions, share ideas, and get community feedback.

1.4 License

PyAEDT is licensed under the MIT license.

PyAEDT makes no commercial claim over Ansys whatsoever. This library extends the functionality of AEDT by adding a Python interface to AEDT without changing the core behavior or license of the original software. The use of PyAEDT requires a legally licensed local copy of AEDT.

To get a copy of AEDT, see the [Ansys Electronics](#) page on the Ansys website.

1.4.1 Installation

PyAEDT consolidates and extends all existing capital around scripting for AEDT, allowing re-use of existing code, sharing of best practices, and collaboration.

This PyAnsys library has been tested on HFSS, Icepak, and Maxwell 3D. It also provides basic support for EDB and Circuit (Nexxim).

Requirements

In addition to the runtime dependencies listed in the installation information, PyAEDT requires Ansys Electronics Desktop (AEDT) 2022 R1 or later. The AEDT Student Version is also supported.

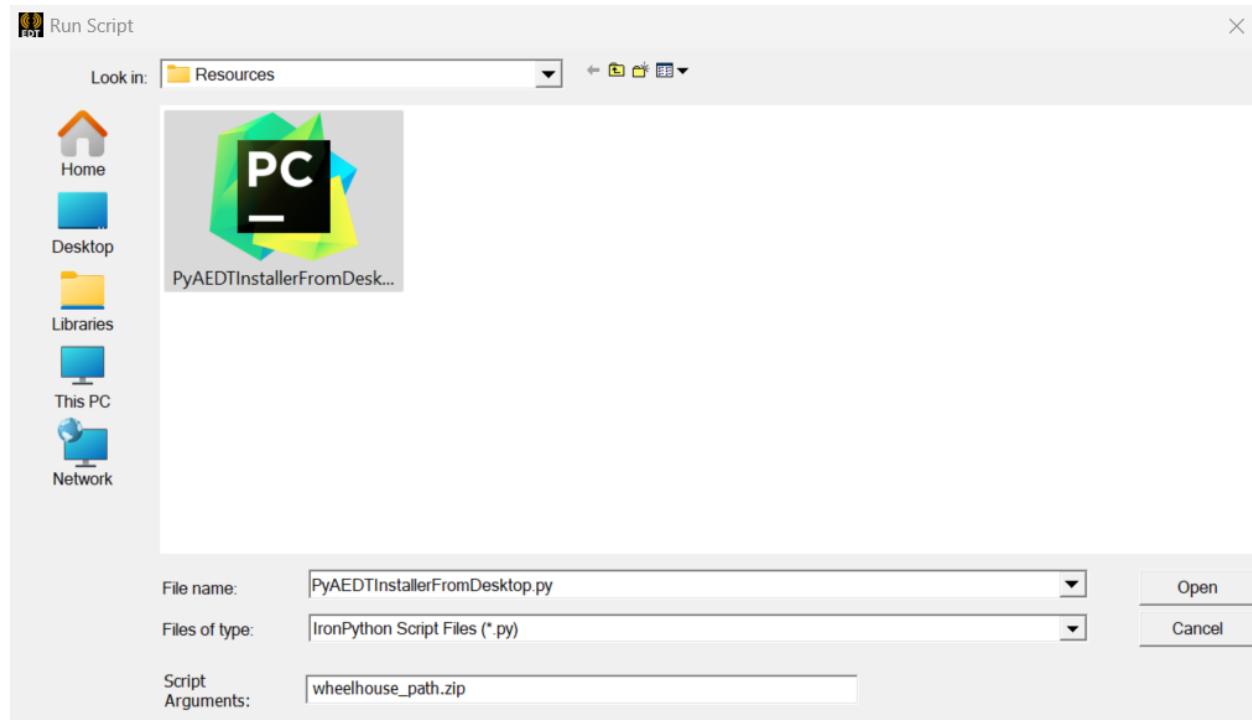
Install from PyAEDT installer

The following python script automatically installs PyAEDT from AEDT, using the CPython interpreter included in the AEDT installation.

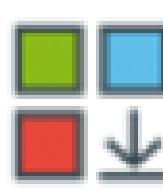
In order to do that you can:

- Download the following file: [PyAEDT Installer Python file](#)
- Open an Electronics Desktop Session and click on Tools->Run Script and execute the file.
- Offline install is also possible using wheelhouses.

Note: A wheelhouse is a zip containing all needed packages that can be installed offline. PyAEDT wheelhouse can be found at [Releases](#). After downloading the wheelhouse zip specific for your distribution and Python release, run the script from Electronics Desktop using the zip full path as argument. Please note that AEDT 2023 R1 and lower requires Python 3.7 wheelhouse while AEDT 2023 R2 and higher requires the Python 3.10 wheelhouse.



Starting from 2023R2, buttons are available in the Automation Tab as in the example below.

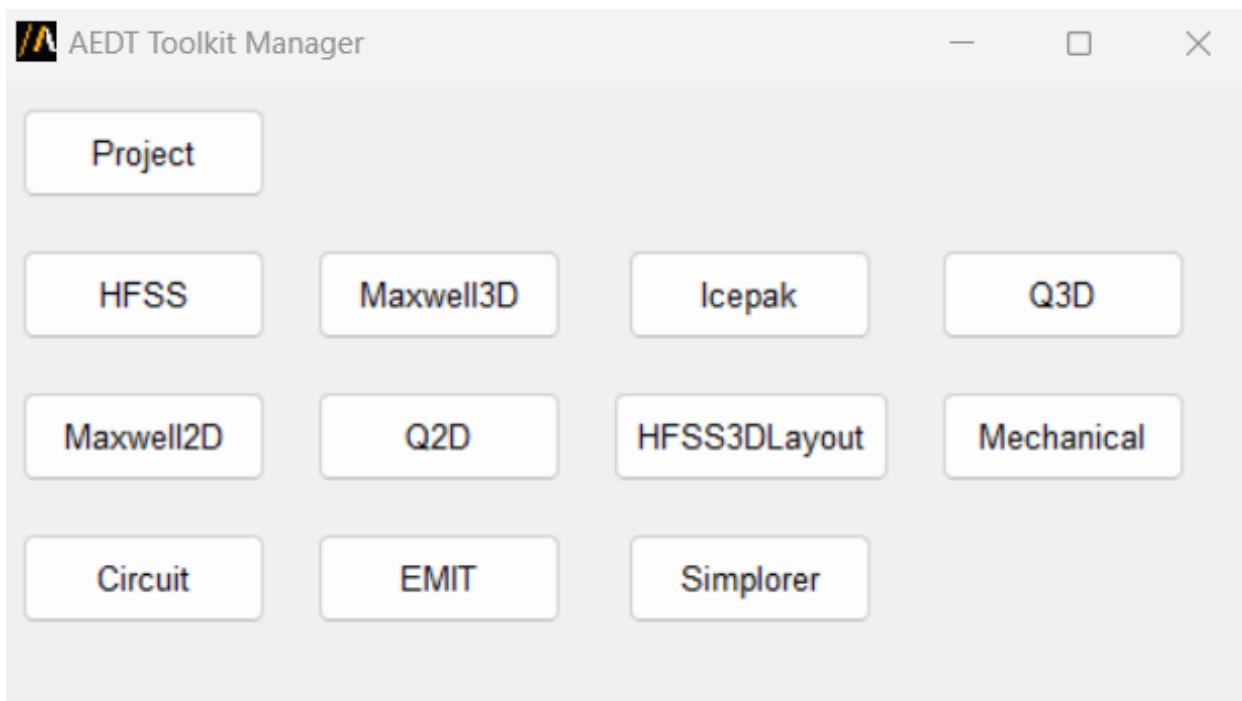


PyAEDT Jupyter Run PyAEDT Toolkit
Console Notebook Script Manager

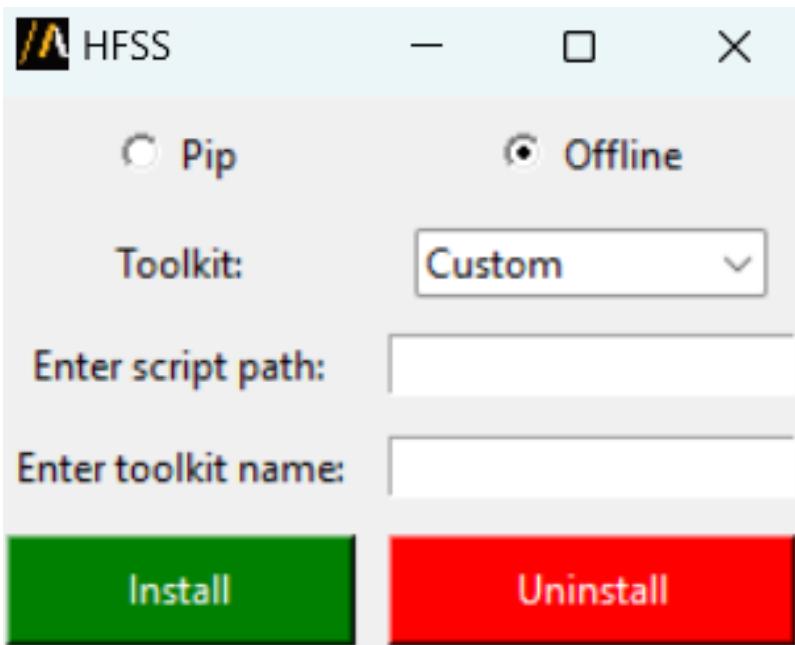
Extension manager

The user can install or uninstall automated workflows using the extension manager. There are three options:

- Custom PyAEDT scripts.
- Existing workflows in the PyAEDT library.
- Open source PyAEDT toolkits described in the [PyAEDT Common Toolkit documentation](#).



The user can select the AEDT application to install the specific workflow.



Install on CPython from PyPI

You can install PyAEDT on CPython 3.7 through 3.10 from PyPI:

```
pip install pyaedt
```

You can also install PyAEDT from Conda-Forge:

```
conda install -c conda-forge pyaedt
```

To ensure you have all the necessary dependencies, including optional components, use the following command:

```
pip install pyaedt[all]
```

If you are not utilizing gRPC, you can install the required dotnet dependencies separately:

```
pip install pyaedt[dotnet]
```

Linux support

PyAEDT works with CPython 3.7 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
```

Install offline from a wheelhouse

Using a wheelhouse can be helpful if you work for a company that restricts access to external networks. Wheelhouses for CPython 3.7, 3.8, and 3.9 are available in the releases for PyAEDT v0.4.70 and later for both Windows and Linux. From the [Releases](#) page in the PyAEDT repository, you can find the wheelhouses for a particular release in its assets and download the wheelhouse specific to your setup.

You can then install PyAEDT and all of its dependencies from one single entry point that can be shared internally, which eases the security review of the PyAEDT package content.

For example, on Windows with Python 3.7, install PyAEDT and all its dependencies from a wheelhouse with code like this:

```
pip install --no-cache-dir --no-index --find-links=file:///<path_to_wheelhouse>/PyAEDT-v
˓→<release_version>-wheelhouse-Windows-3.7 pyaedt
```

Use IronPython in AEDT

PyAEDT is designed to work in CPython 3.7+ and supports many advanced processing packages like `matplotlib`, `numpy`, and `pystyle`. A user can still use PyAEDT in the IronPython environment available in AEDT with many limitations.

To use IronPython in AEDT:

1. Download the PyAEDT package from <https://pypi.org/project/pyaedt/#files>.
2. Extract the files.
3. Install PyAEDT into AEDT, specifying the full paths to `ipy64` and `setup-distutils.py` as needed:

```
ipy64 setup-distutils.py install --user
```

Install PyAEDT in Conda virtual environment

Create virtual environment

```
conda create --name pyaedt_py310 python=3.10
```

Activate virtual environment

```
conda activate pyaedt_py310
```

You can also install PyAEDT from Conda-Forge with this command:

```
conda install -c conda-forge pyaedt
```

Upgrade PyAEDT to the latest version

```
pip install -U pyaedt
```

1.4.2 Troubleshooting

This section contains common issues and suggestions related to installation and use of PyAEDT.

Installation

Error installing Python or Conda

Sometimes 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 3.7 is available in AEDT 2023 R1 and earlier. Python 3.10 is available in AEDT 2023 R2.

Here is the path to the Python 3.7 interpreter for the 2023 R1 installation:

```
path\to\AnsysEM\v231\commonfiles\CPython\3_7\winx64\Release\python"
```

Error installing PyAEDT using pip

- **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.
- **Install 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. If you face this issue, you'll have to work with your IT administrator to enable pip. The proxy server settings (described earlier) allow you to explicitly define the ports used by pip.

If downloads from [pypi](#) are not allowed, you may use a [wheelhouse](#). The wheelhouse file contains all dependencies for PyAEDT and allows full installation without a need to download additional files. The wheelhouse for PyAEDT can be found [here](#). After downloading the wheelhouse for your distribution and Python release, unzip the file to a folder and run the Python command:

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

Another option to install PyAEDT from the wheelhouse is to download the following file [PyAEDT Installer Python file](#). Run this script directly from AEDT and pass the wheelhouse file name as an argument.

Run PyAEDT

COM and gRPC

Prior to the 2022 R2 release, CPython automation in AEDT used [COM](#), which requires all interfaces to be registered in the Windows Registry. Communication between Python and the AEDT API were translated through an intermediate layer using [pywin32](#) and [PythonNET](#).

[gRPC](#) is a modern open source high performance Remote Procedure Call (RPC) framework that can run in any environment and supports client/server remote calls. Starting from 2022R2 the AEDT API has replaced the COM interface with a gRPC interface.

Table 1: *gRPC Compatibility:*

< 2022 R2	2022 R2	> 2022 R2
Only Python.NET	<p>Python.NET: <i>Default</i> Enable gRPC: <code>pyaedt.settings.use_grpc_api = True</code></p>	<p>gRPC: <i>Default</i> Enable Python.NET: <code>pyaedt.settings.use_grpc_api = False</code></p>

The options shown here apply only to the Windows platform. On Linux, the Python interface to AEDT uses gRPC for all versions.

Check the AEDT API configuration

Run the following command to start AEDT as a gRPC server:

Windows:

```
path\to\AnsysEM\v231\Win64\ansysedt.exe -grpcsrv 50001
```

On Linux:

```
path\to\AnsysEM\v231\Lin64\ansysedt -grpcsrv 50352
```

The server port number is used by AEDT to listen and receive commands from the PyAEDT client. This configuration supports multiple sessions of AEDT running on a single server and listening on the same port.

Check the gRPC interface

The native Electronics Desktop API can be used to launch AEDT from the command line. PyAEDT is not required to verify the setup for the server and ensure that all environment variables have been defined correctly.

```
import sys
sys.path.append(r"ANSYSEM_ROOT231\PythonFiles\DesktopPlugin")
import ScriptEnv
print(dir())
ScriptEnv.Initialize("", False, "", 50051)
print(dir())
```

Failure connecting to the gRPC server

On Linux, PyAEDT may fail to initialize a new instance of the gRPC server or connect to an existing server session. This may be due to:

- Firewall
- Proxy
- Permissions
- License
- Scheduler (for example if the gRPC server was started from LSF or Slurm)

For issues related to use of a proxy server, you may set the following environment variable to disable the proxy server for the *localhost*.

```
export no_proxy=localhost,127.0.0.1
```

Run your PyAEDT script.

If it still fails, you can disable the proxy server:

```
export http_proxy=
```

Run your PyAEDT script. If the errors persist, perform these steps:

1. Check that AEDT starts correctly from the command line by starting the *gRPC server*.
2. Enable debugging.

```
export ANSOFT_DEBUG_LOG=/tmp/testlogs/logs/lg
export ANSOFT_DEBUG_LOG_SEPARATE=1
export ANSOFT_DEBUG_LOG_TIMESTAMP=1
export ANSOFT_DEBUG_LOG_THREAD_ID=1
export ANSOFT_DEBUG_MODE=3
```

Enable the gRPC trace on the server:

```
export GRPC_VERBOSITY=DEBUG
export GRPC_TRACE=all
```

Then run *ansysedt.exe* as a gRPC server and redirect the output.

```
ansysedt -grpcsrv 50051 > /path/to/file/server.txt
```

The preceding command redirects the gRPC trace to the file *server.txt*.

Open another terminal window to trace the gRPC calls on the client where the Python script is to be run.

```
export GRPC_VERBOSITY=DEBUG
export GRPC_TRACE=all
```

Now run the PyAEDT script, (making sure it connects to the same port as the gRPC server - 50051). Capture the output in a file. For example *client.txt*. Then send all the logs to [Ansys Support](#).

1.4.3 User guide

This section provides brief tutorials for helping you understand how to use PyAEDT effectively.

For end-to-end examples, see [Examples](#).

Basic tutorial How to launch AEDT and create a project.

Basic tutorial Modeler How to use 2D and 3D Modeler.

Modeler Mesh How PyAEDT handles mesh operations.

Mesh Setup How to create a setup and run simulations.

Setup Variables How to generate parametric models and run optimizations.

Variables Load and manage files How to load and manage input and output files.

Load and manage files Postprocessing How to generate reports, images, and PDF files.

Postprocessing EMIT Modeler How to create and analyze EMIT designs.

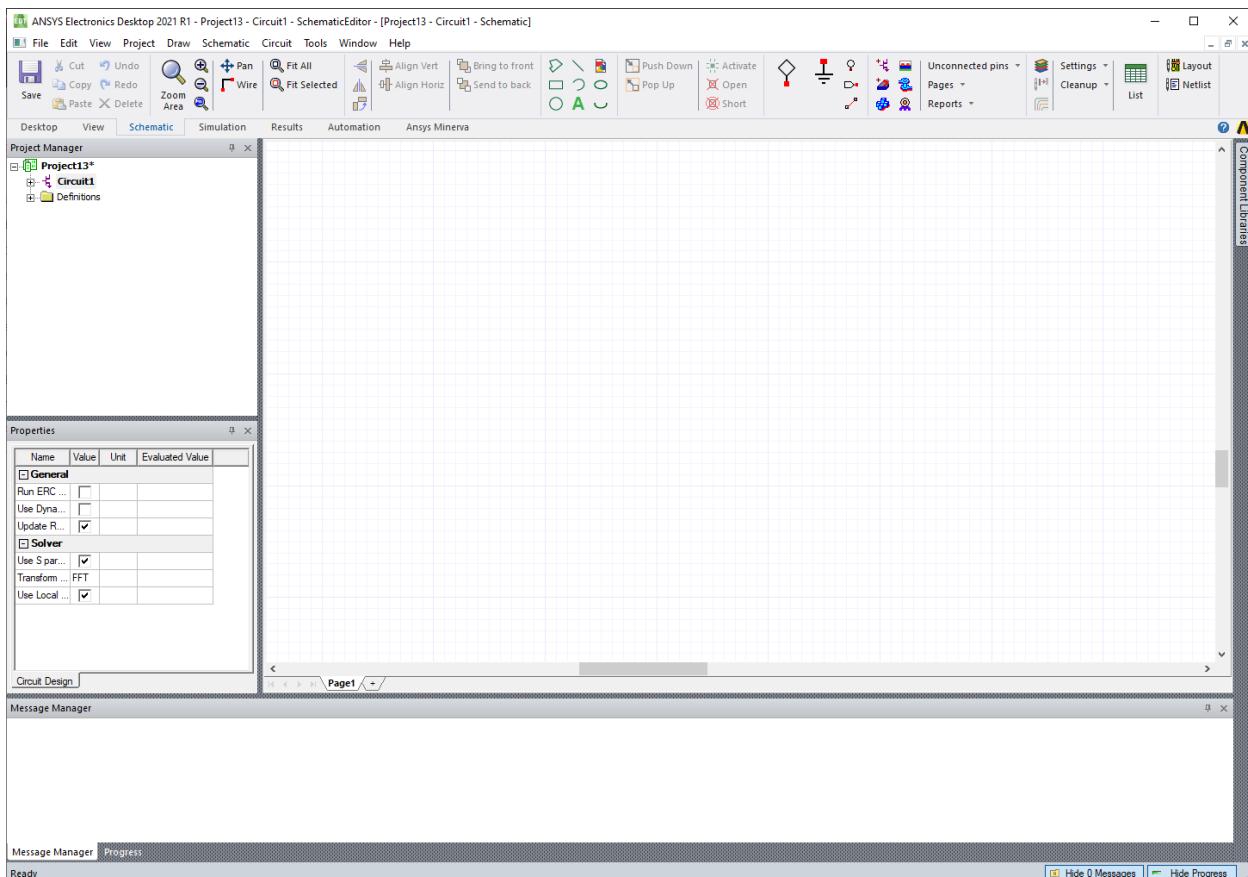
EMIT modeler

Basic tutorial

You can initiate AEDT in non-graphical mode from Python using this code:

```
# Launch AEDT 2023 R2 in non-graphical mode
import pyaedt
with pyaedt.Desktop(specified_version="2023.2", non_graphical=True, new_desktop_=True, close_on_exit=True,
                     student_version=False):
    circuit = pyaedt.Circuit()
    ...
    # Any error here is caught by AEDT.
    ...
# AEDT is automatically closed here.
```

The preceding code launches AEDT and initializes a new Circuit design.



This code creates a project and saves it with PyAEDT:

```
# Launch the latest installed version of AEDT in graphical mode.
import pyaedt
cir = pyaedt.Circuit(non_graphical=False)
cir.save_project(my_path)

...
cir.release_desktop(save_project=True, close_desktop=True)
# Desktop is released here.
```

This code uses PyAEDT to access the Ansys EDB proprietary layout format:

```
# Launch the latest installed version of EDB.
import pyaedt
edb = pyaedt.Edb("mylayout.aedb")

# User can launch EDB directly from the PyEDB class.

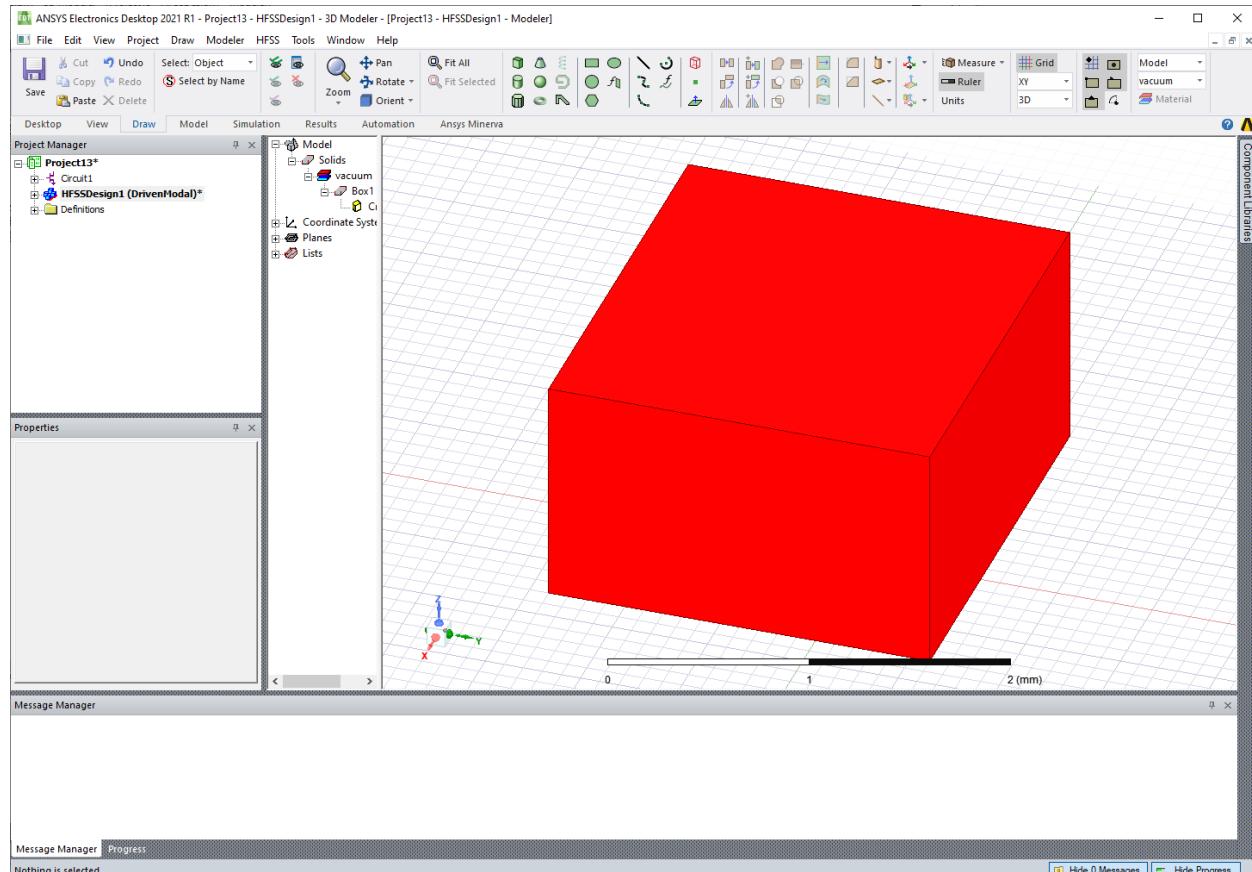
import pyedb
edb = pyedb.Edb("mylayout.aedb")
```

Modeler

The AEDT 3D and 2D Modelers use object-oriented programming to create and manage objects. You can use getters and setters to create an object and change its properties.

The following code creates a box and changes the color to red:

```
from pyaedt.hfss import Hfss
hfss = Hfss()
box = hfss.modeler.create_box(origin=[0, 0, 0],
                               sizes=[10, "dim", 10],
                               name="mybox",
                               material="aluminum")
print(box.faces)
box.color = "Red"
```



Similarly, you can change other properties, such as the material and transparency.

```
box.material_name = "copper"
box.transparency = 0.4
print(box.material_name)
```

Once an object is created or is present in the design (from a loaded project), you can use a getter to get the related object. A getter works either with an object ID or object name. The object returned has all features, even if it has not been created in PyAEDT.

This example shows how easily you can go deeper into edges and vertices of faces or 3D objects:

```
box = hfss.modeler["mybox2"]
for face in box.faces:
    print(face.center)
    for edge in face:
        print(edge.midpoint)
        for vertice in edge.vertices:
            print(edge.position)
for vertice in box.vertices:
    print(edge.position)
```

All objects support executing any modeler operation, such as union or subtraction:

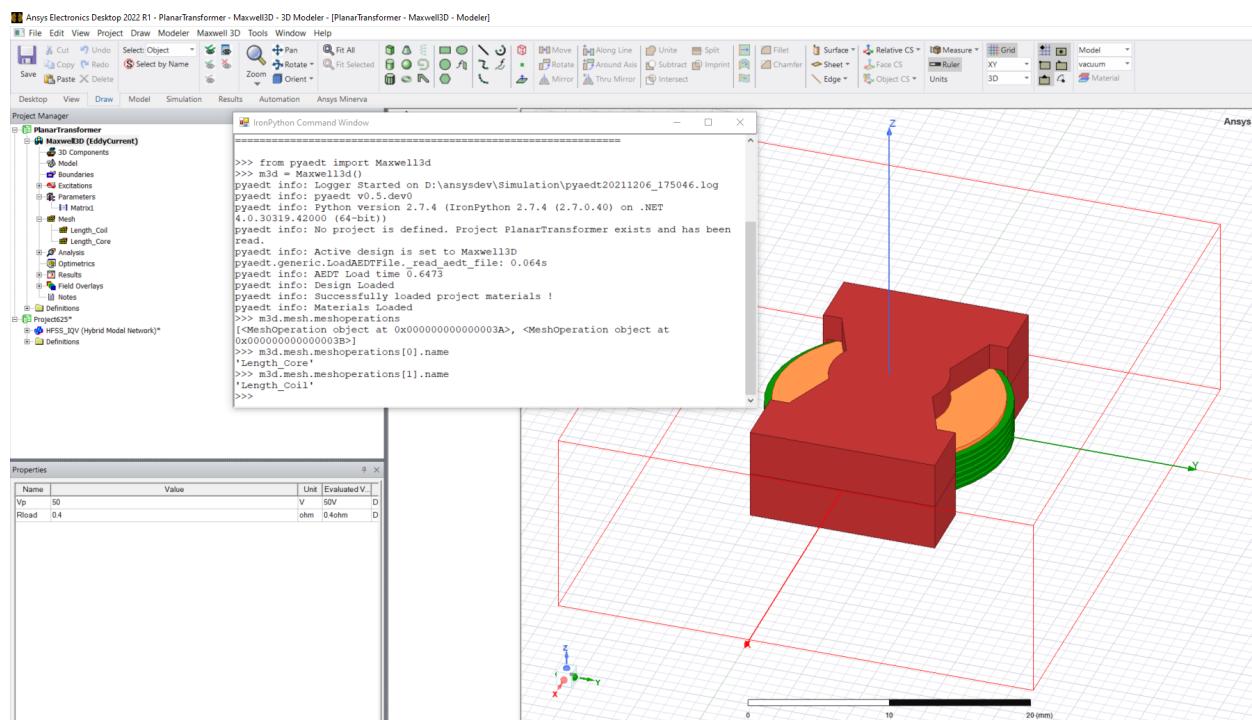
```
box = hfss.modeler["mybox2"]
cyl = hfss.modeler["mycyl"]
box.unite(cyl)
box.subtract(cyl)
```

The following demo is presented as an animated GIF. [View online](#) if you are reading the PDF version of this documentation.

Mesh

Mesh operations are very important in engineering simulation. PyAEDT can read existing mesh operations in a design, make edits, and create new operations. All mesh operations are conveniently listed within the mesh object:

```
from pyaedt import Maxwell3d
m3d = Maxwell3d()
all_mesh_ops = m3d.mesh.meshoperations
my_mesh_op = all_mesh_ops[0]
# All properties are in props dictionary.
my_mesh_op.props["my_prop"] = "my_value"
my_mesh_op.update()
```



Icepak has a different approach to the mesh operations. Those are managed through mesh regions and can be edited directly from the pyaedt object.

```
icepak_b = Icepak()

# Global mesh region

icepak_b.mesh.global_mesh_region.MaxElementSizeX = "2mm"
icepak_b.mesh.global_mesh_region.MaxElementSizeY = "3mm"
icepak_b.mesh.global_mesh_region.MaxElementSizeZ = "4mm"
icepak_b.mesh.global_mesh_region.MaxSizeRatio = 2
icepak_b.mesh.global_mesh_region.UserSpecifiedSettings = True
icepak_b.mesh.global_mesh_region.UniformMeshParametersType = "XYZ Max Sizes"
icepak_b.mesh.global_mesh_region.MaxLevels = 2
icepak_b.mesh.global_mesh_region.BufferLayers = 1
icepak_b.mesh.global_mesh_region.update()

box1 = icepak_b.modeler.create_box([0,0,0], [10,20,20])

# Local mesh region

new_mesh_region = icepak_b.mesh.assign_mesh_region([box1.name])
```

In HFSS 3D Layout, you add mesh operations to nets and layers like this:

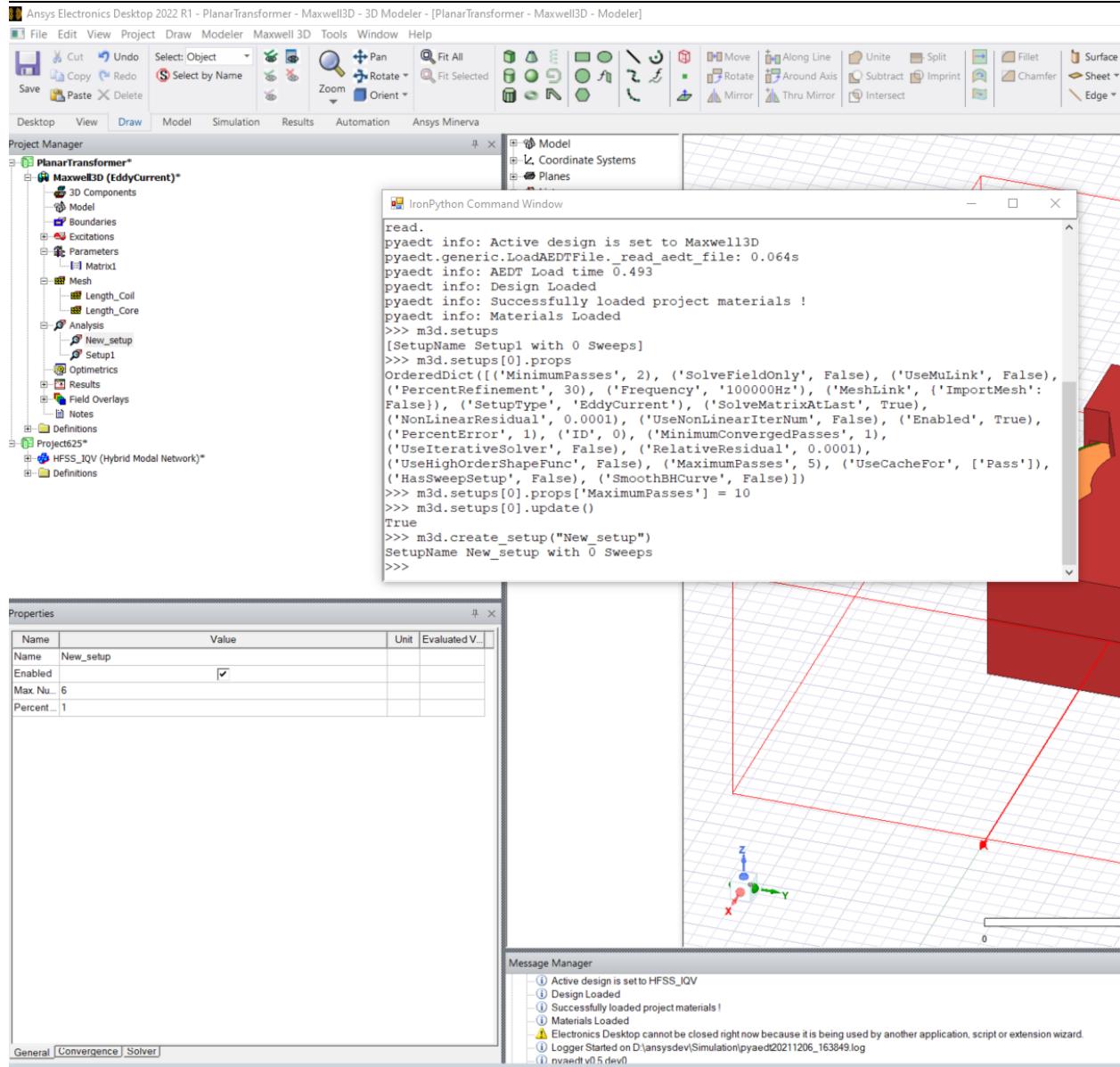
```
from pyedt import Hfss3dLayout

h3d = Hfss3dLayout("myproject.aedt")
setup = h3d.create_setup("HFSS")
mop1 = h3d.mesh.assign_length_mesh("HFSS", layer_name="PWR", net_name="GND")
mop2 = h3d.mesh.assign_skin_depth("HFSS", layer_name="LAY2", net_name="VCC")
```

Setup

Setup and sweeps are the last operations before running an analysis. PyAEDT facilitates seamless interaction with these operations by enabling the reading, editing, and creation of setups and sweeps within a design. All setup operations are conveniently accessible and organized in the setups list, providing a clear and intuitive interface for managing and customizing these essential components of the simulation process:

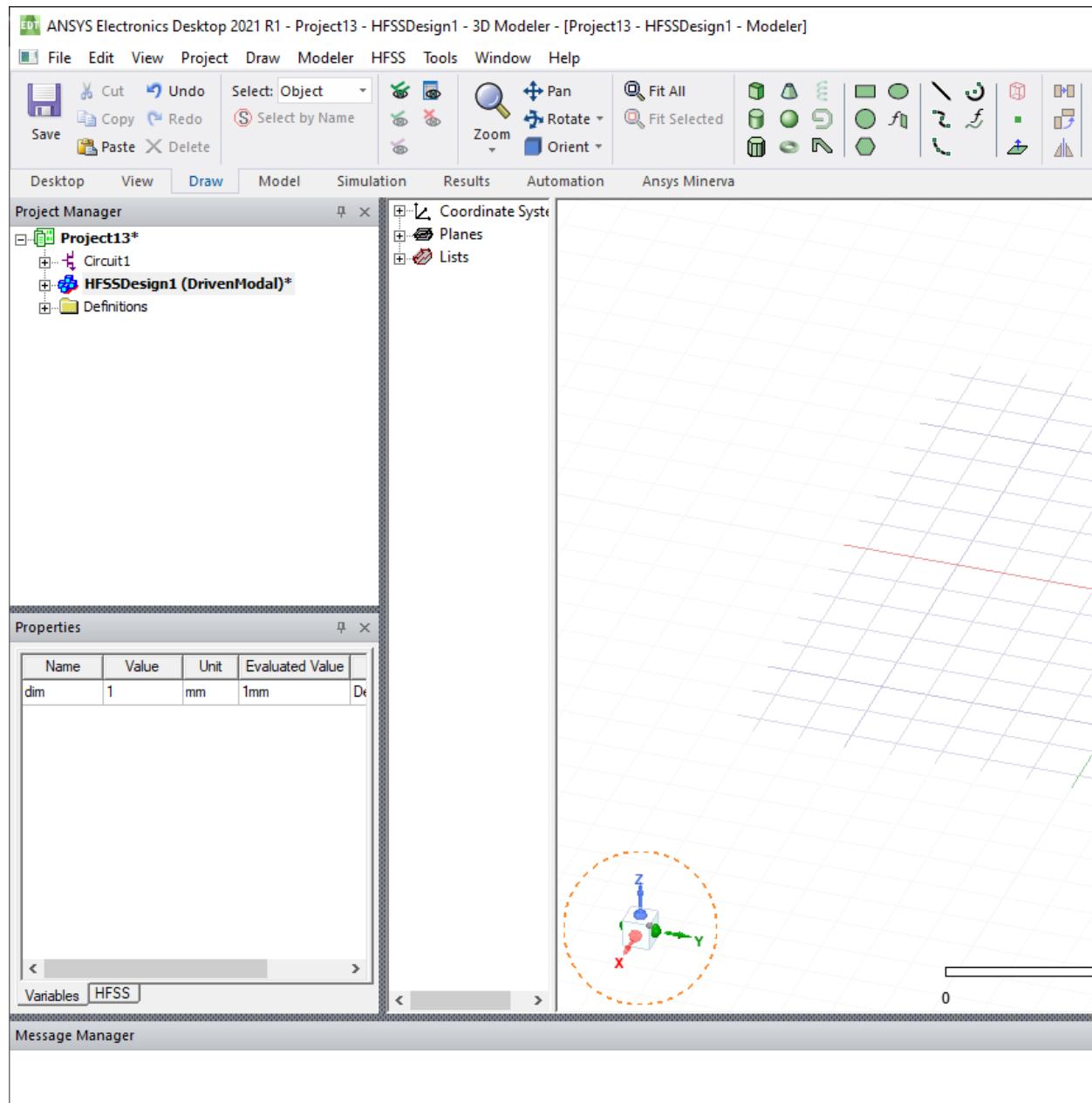
```
from pyaedt import Maxwell3d
m3d = Maxwell3d()
all_setups = m3d.setups
my_setup = all_setups[0]
# All properties are in props dictionary.
my_setup.props['MaximumPasses'] = 10
new_setup = m3d.create_setup("New_Setup")
```



Variables

PyAEDT provides a simplified interface for getting and setting variables inside a project or a design. You simply need to initialize a variable as a dictionary key. If you use \$ as the prefix for the variable name, a project-wide variable is created:

```
from pyaedt import Hfss
with Hfss as hfss:
    hfss["dim"] = "1mm"      # design variable
    hfss["$dim"] = "1mm"     # project variable
```

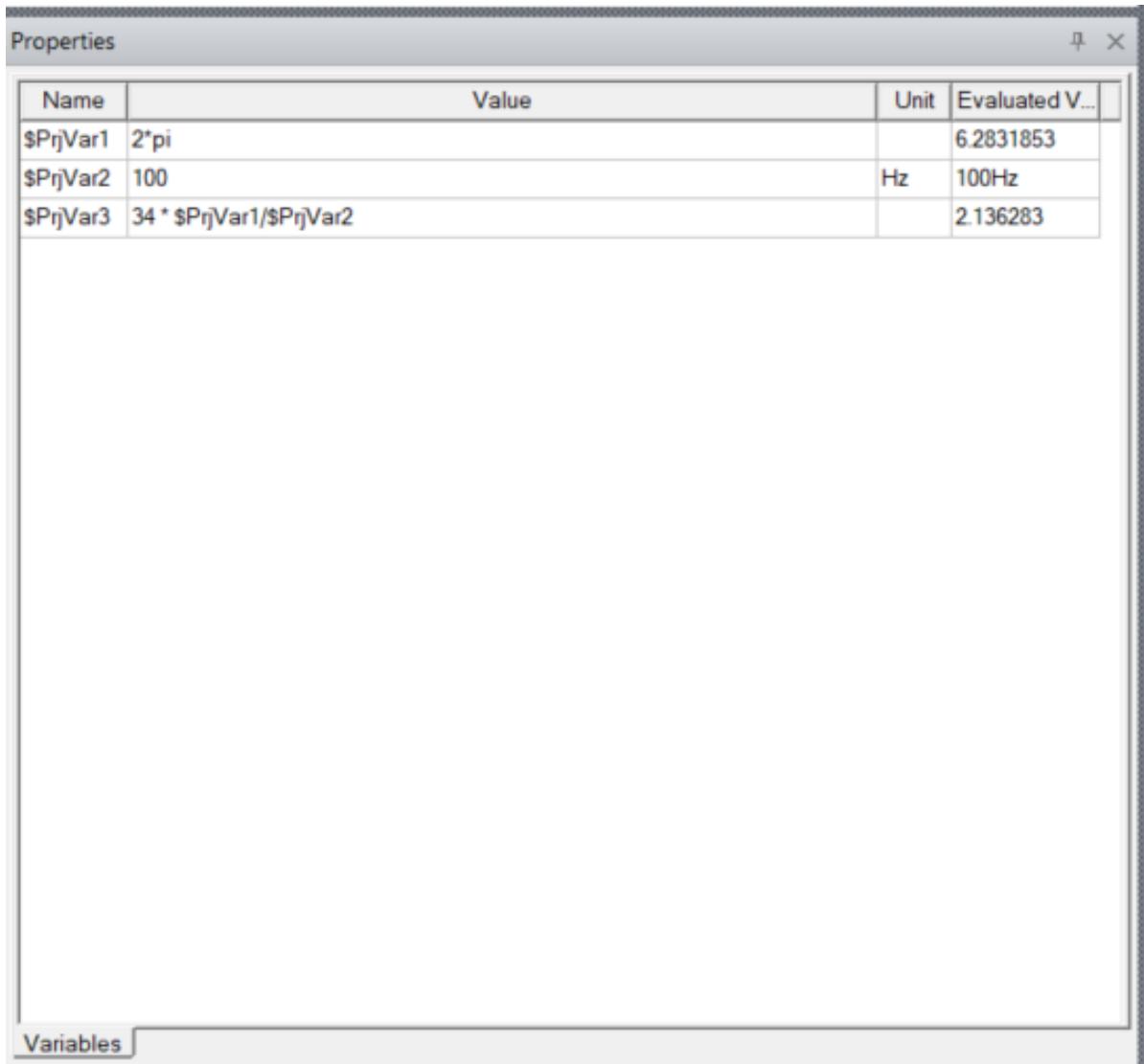


With Variable Manager, you can create advanced equations and manage them through PyAEDT.

While you can set or get the variable value using the apps setter and getter, you can access the `variable_manager`

object for a more comprehensive set of functions:

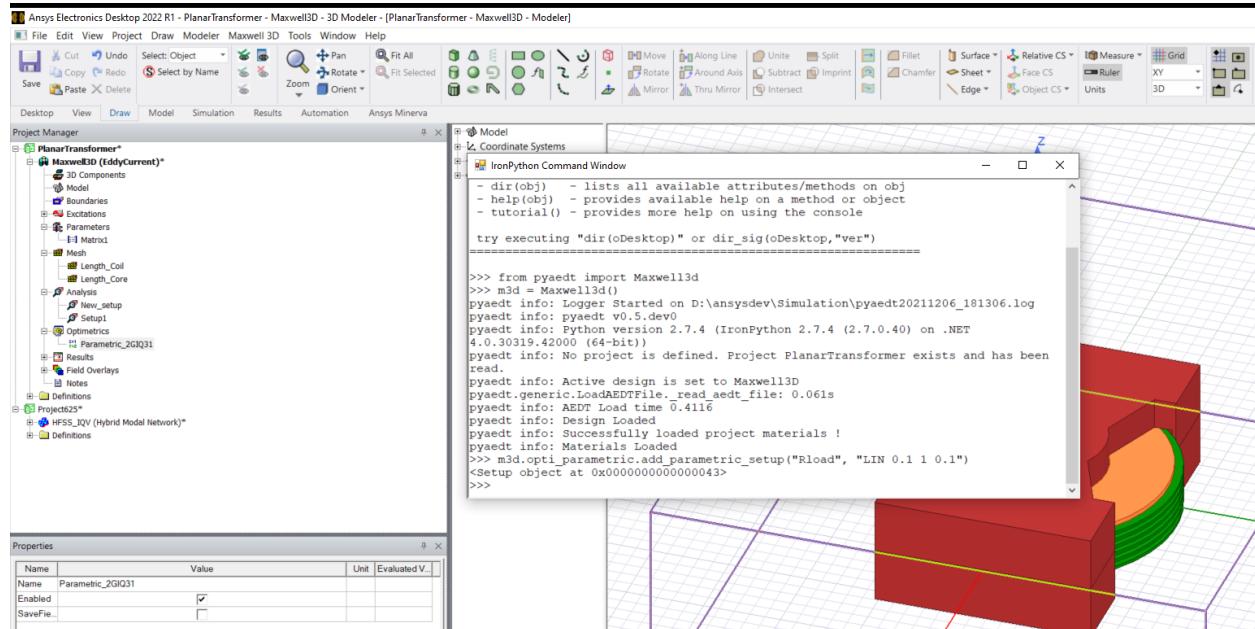
```
>>> hfss["$PrjVar1"] = "2*pi"
>>> hfss["$PrjVar2"] = "100Hz"
>>> hfss["$PrjVar3"] = "34 * $PrjVar1/$PrjVar2"
>>> hfss["$PrjVar3"]
2.13628300444106
>>> hfss.evaluate_expression(hfss["$PrjVar3"])
2.13628300444106
hfss.variable_manager["$PrjVar3"].expression
'34 * $PrjVar1/$PrjVar2'
```



Optimetrics

Optimetrics allows you to set up parametric analyses, sensitivity analyses, optimization, and design of experiments. PyAEDT provides the capability to seamlessly interact with optimetrics setups by allowing you to read existing configurations, make edits, and create setups:

```
from pyaedt import Maxwell3d
m3d = Maxwell3d()
m3d.parametrics.add("Rload", 0.1, 1, 0.1)
```



Load and manage files

PyAEDT supports different file formats to manage simulation and setup data. Descriptions follow for the most used formats.

Project file The project configuration file allows you to apply a set of variables, materials, setup, mesh, and boundaries.

Project configuration file Report file The report configuration files allows you to easily create customized AEDT reports.

Report configuration file Materials file The report configuration files allows you to import and export materials in AEDT.

Materials file

Primitives file Create primitives from file.

Primitives file

Array File to control 3D Component array geometry.

Antenna array file

Cables File to control cable design.

Cable file

Choke File to control choke synthesis.

Choke file

Project configuration file

The project configuration file is the most comprehensive format that allows to apply a set of variables, materials, setup, mesh, and boundaries to a new project or save an existing one. This code creates the JSON file:

```
from pyaedt import Icepak
ipk = Icepak()
filename = "test"
ipk.export_3d_model(file_name=filename,
                     file_path=ipk.working_directory,
                     file_format=".step",
                     object_list=[],
                     removed_objects[])
conf_file = ipk.configurations.export_config()
ipk.release_desktop()
```

This code imports the project configuration from the JSON file:

```
app.modeler.import_3d_cad(file_path)
out = app.configurations.import_config(conf_file)
```

File structure examples:

[Icepak Example](#)

[HFSS 3D Layout Example](#)

```
{
    "general": {
        "pyaedt_version": "0.8.dev0",
        "model_units": "mm",
        "design_name": "IcepakDesign1",
        "date": "09/01/2024 08:22:17",
        "object_mapping": {
            "# object_id": [
                "#     object_name",
                "#     object_center
            ],
            "12": [
                "Region",
                [
                    80.0,
                    14.243,
                    -55.0
                ]
            ]
        },
        "output_variables": {},
        "variables": {},
        "postprocessing_variables": {}
    },
    "setups": {
        # Setup Name : {Setup Properties}
        "MySetupAuto": {

```

(continues on next page)

(continued from previous page)

```
        "Enabled": true,
        "Flow Regime": "Turbulent",
        "Include Temperature": true,
    }
},
"boundaries": {
    # Boundary Name : {Boundary Properties}
    "CPU": {
        "Objects": [
            "CPU"
        ],
        "Block Type": "Solid",
        "Use External Conditions": false,
        "Total Power": "25W",
        "BoundType": "Block"
    },
},
"mesh": {
    "Settings": {
        # mesh_properties,
        "MeshMethod": "MesherHD",
        "UserSpecifiedSettings": true,
        "ComputeGap": true,
        "MaxElementSizeX": "16mm",
        "MaxElementSizeY": "3.5mm",
        "MaxElementSizeZ": "11mm",
        "# ....
    }
},
"materials": {
    # Material Name : {Material Properties}
    "Al-Extruded": {
        "CoordinateSystemType": "Cartesian",
        "BulkOrSurfaceType": 1,
        "PhysicsTypes": {
            "set": [
                "Thermal"
            ]
        },
        "AttachedData": {
            "MatAppearanceData": {
                "property_data": "appearance_data",
                "Red": 232,
                "Green": 235,
                "Blue": 235
            }
        },
        "thermal_conductivity": "205",
        "mass_density": "2800",
        "specific_heat": "900",
        "thermal_material_type": {
            "property_type": "ChoiceProperty",

```

(continues on next page)

(continued from previous page)

```

        "Choice": "Solid"
    },
    "clarity_type": {
        "property_type": "ChoiceProperty",
        "Choice": "Opaque"
    }
},
{
    "objects": {
        # Object Name: {object properties}
        "Region": {
            "SurfaceMaterial": "",
            "Material": "air",
            "SolveInside": true,
            "Model": true,
            "Group": "",
            "Transparency": 0.0,
            "Color": [
                255,
                0,
                0
            ],
            "CoordinateSystem": "Global"
        },
        ...
    },
    "datasets": [
        # Dataset Name : {Dataset Properties}
    ],
    "monitors": [
        # Monitor Name : {Monitor Properties}
    ],
    "native_components": {
        # Component Name : {Component Properties}
    }
}

```

For a practical demonstration, see the [Project configuration file example](#)

Report configuration file

The report configuration file allows to create a report based on a JSON file or a dictionary of properties.

This code shows how to create the JSON file:

```

from pyaedt import Hfss
hfss = Hfss()
compfile = hfss.components3d["Dipole_Antenna_DM"]
geometriparams = hfss.get_components3d_vars("Dipole_Antenna_DM")
hfss.modeler.insert_3d_component(compfile, geometriparams)
hfss.create_setup()
filename = "hfss_report_example.json"

```

(continues on next page)

(continued from previous page)

```
hfss.post.create_report_from_configuration(input_file=filename)
hfss.release_desktop()
```

File structure example:

HFSS report example

For a practical demonstration, see the [Report configuration file example](#)

Materials file

The material configuration file allows you to create materials from a JSON file.

This code creates the JSON file:

```
from pyaedt import Maxwell3d
maxwell = Maxwell3d()
maxwell.materials.export_materials_to_file("materials.json")
maxwell.release_desktop()
```

This code imports materials from the JSON file:

```
from pyaedt import Maxwell3d
maxwell = Maxwell3d()
maxwell.materials.import_materials_from_file("material_example.json")
maxwell.release_desktop()
```

File structure example:

Material example

Primitives file

The primitives configuration file allows you to create primitive shapes from a JSON file.

This code creates primitive shapes from the JSON file:

```
from pyaedt import Icepak
ipk = Icepak()
ipk.modeler.import_primitives_from_file("primitive_example.json")
ipk.release_desktop()
```

File structure example:

Primitive example

Antenna array file

The array configuration file allows you to create a 3D Component array from a JSON file.

This file can be created using the following command:

```
from pyaedt import Hfss
from pyaedt.generic.DataHandlers import json_to_dict
hfss = Hfss()
dict_in = json_to_dict("array_simple.json")
dict_in["Circ_Patch_5GHz1"] = "Circ_Patch_5GHz_232.a3dcomp"
dict_in["cells"][(3, 3)] = {"name": "Circ_Patch_5GHz1"}
component_array = hfss.add_3d_component_array_from_json(dict_in)
hfss.release_desktop()
```

File structure example:

[Antenna Array example](#)

For a practical demonstration, see the [Antenna array example](#).

Cable file

The cable configuration file allows you to create a cable geometry with shielding from a JSON file.

This file can be created using the following command:

```
from pyaedt import Hfss
from pyaedt.modules.CableModeling import Cable
from pyaedt.generic.DataHandlers import json_to_dict
hfss = Hfss()
cable = Cable(hfss, "set_cable_properties.json")
cable.create_cable()
hfss.release_desktop()
```

File structure example:

[Cable example](#)

For a practical demonstration, see the [Cable API example](#).

Choke file

The choke configuration file allows you to synthesize a choke from a JSON file.

This code creates the choke geometry:

```
from pyaedt import Hfss

hfss = Hfss()
choke_file1 = "choke_example.json"
choke = hfss.modeler.create_choke(choke_file1)
hfss.release_desktop()
```

File structure example:

Choke example

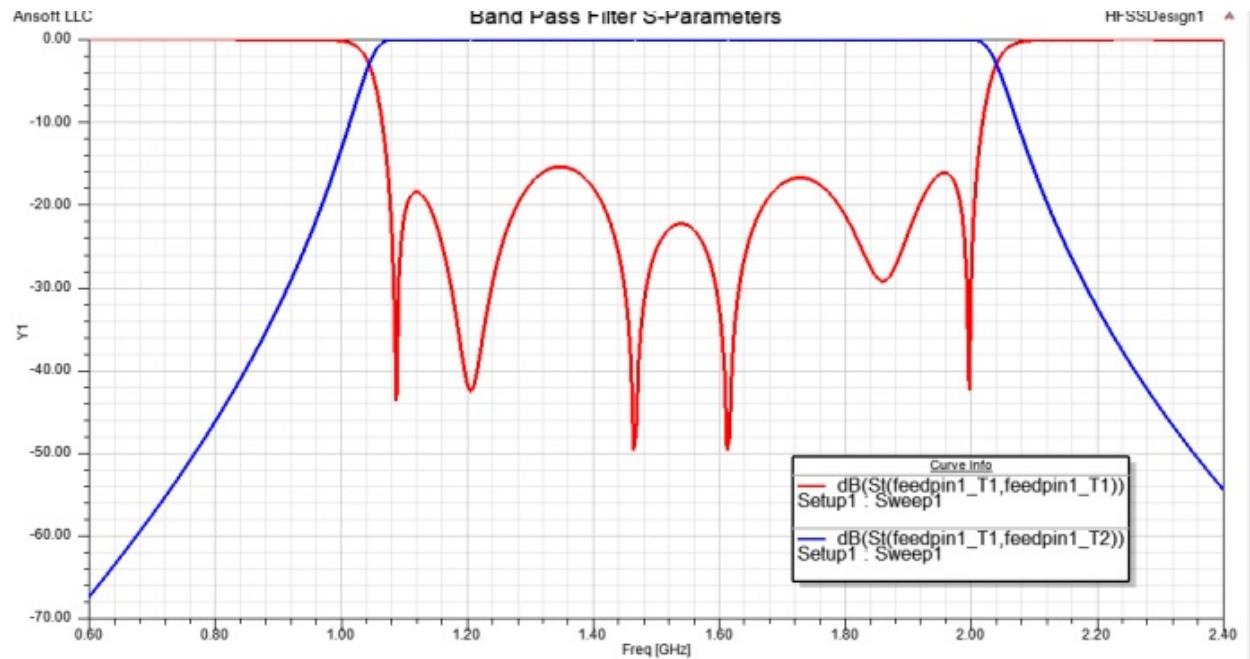
For a practical demonstration, see the [Choke example](#).

Postprocessing

Postprocessing is essential in simulation. PyAEDT offers the capability to read solutions and visualize results both within AEDT and externally using the `pystyle` and `matplotlib` packages.

To use PyAEDT to create a report in AEDT, you can follow this general structure:

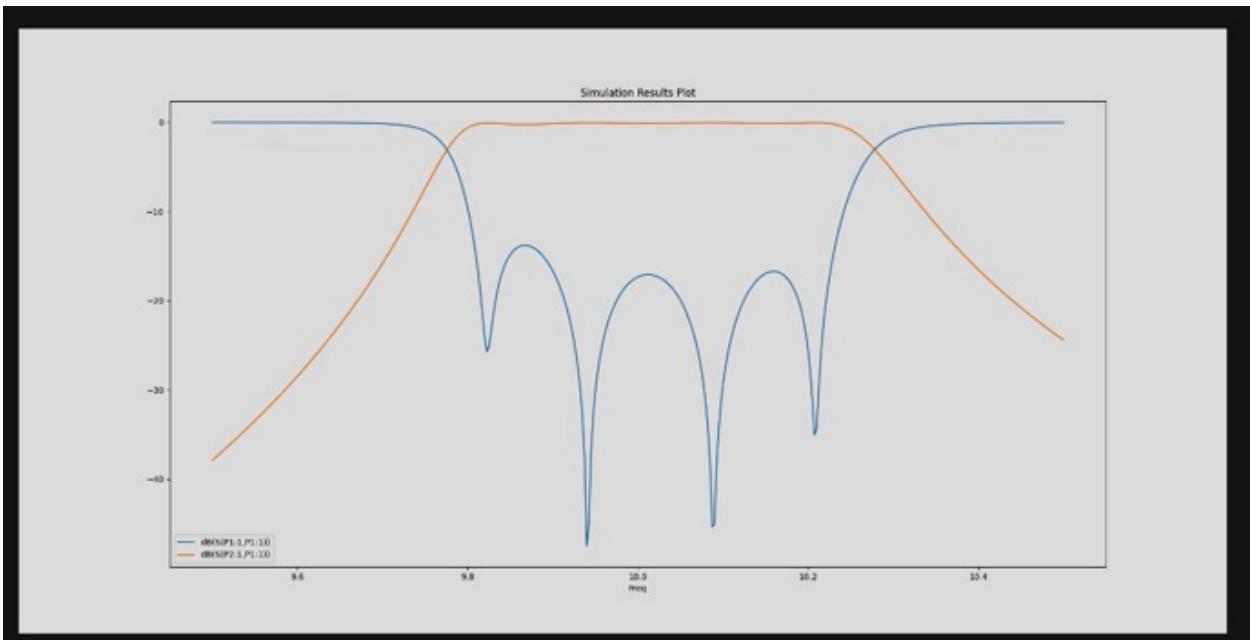
```
from pyaedt import Hfss
hfss = Hfss()
hfss.analyze_nominal()
hfss.post.create_report(["db(S11)", "db(S12)"])
```



You can also generate reports in Matplotlib:

```
from pyaedt import Hfss
hfss = Hfss()
hfss.analyze_nominal()

traces_to_plot = hfss.get_traces_for_plot(second_element_filter="P1*")
report = hfss.post.create_report(traces_to_plot) # Creates a report in HFSS
solution = report.get_solution_data()
plt = solution.plot(solution.expressions) # Matplotlib axes object.
```



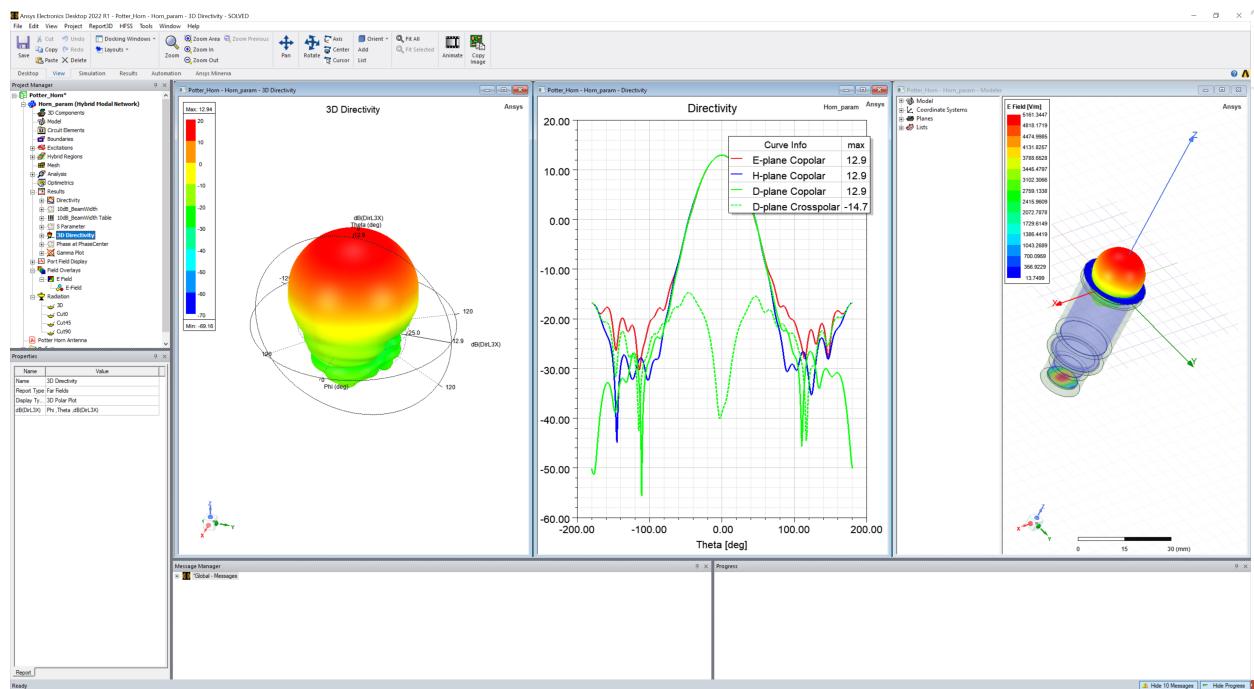
You can use PyAEDT to plot any kind of report available in the Electronics Desktop interface. To access all available category, use the `reports_by_category` class.

```
from pyaedt import Hfss
hfss = Hfss()
hfss.analyze_nominal()
# Create a 3D far field
new_report = hfss.post.reports_by_category.far_field(expressions="db(RealizedGainTotal)",
                                                       setup=hfss.nominal_adaptive)
```

You can plot the field plot directly in HFSS and export it to image files.

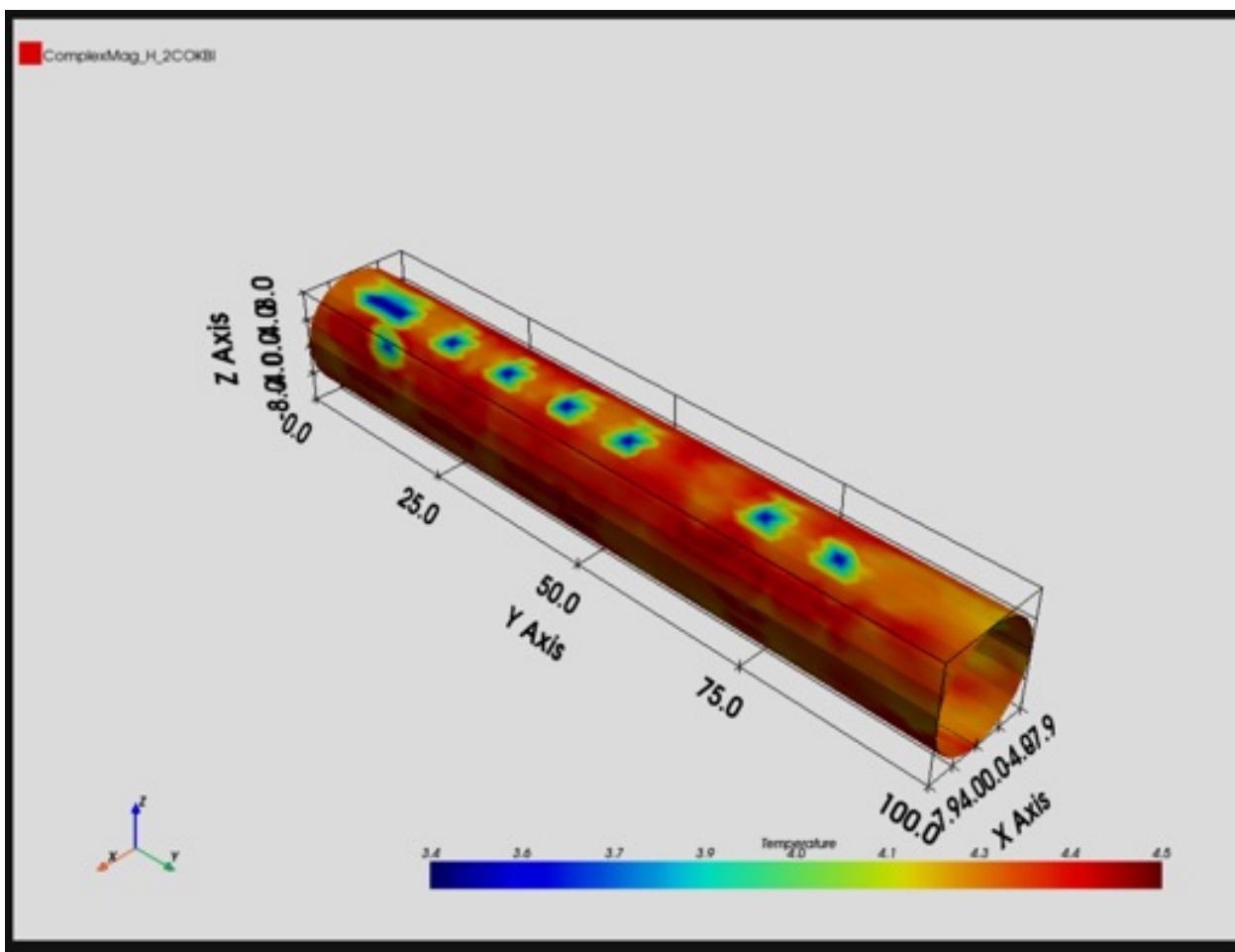
```
from pyaedt import Hfss
hfss = Hfss()
hfss.analyze_nominal()

cutlist = ["Global:XY"]
setup_name = hfss.existing_analysis_sweeps[0]
quantity_name = "ComplexMag_E"
intrinsic = {"Freq": "5GHz", "Phase": "180deg"}
# Create a field plot
plot1 = hfss.post.create_fieldplot_cutplane(objlist=cutlist,
                                              quantityName=quantity_name,
                                              setup=setup_name,
                                              intrinsic=intrinsic)
```



PyAEDT leverages PyVista to export and plot fields outside AEDT, generating images and animations:

```
from pyaedt import Hfss
hfss = Hfss()
hfss.analyze_nominal()
cutlist = ["Global:XY"]
setup_name = hfss.existing_analysis_sweeps[0]
quantity_name = "ComplexMag_E"
intrinsic = {"Freq": "5GHz", "Phase": "180deg"}
hfss.logger.info("Generating the image")
plot_obj = hfss.post.plot_field(
    quantity="Mag_E",
    objects_list=cutlist,
    plot_type="CutPlane",
    setup=setup_name,
    intrinsic=intrinsic
)
```



PyAEDT includes a powerful class to generate a PDF report that is based on the Python `fpdf2` package.

To see the capabilities offered by PyAEDT, download a sample PDF report that uses this class:

PDF report example

This code creates the previous PDF report:

```
from pyaedt.generic.pdf import AnsysReport
import os
report = AnsysReport()
report.aedt_version = "2024R1"
report.template_name = "AnsysTemplate"
report.project_name = "Coaxial1"
report.design_name = "Design2"
report.template_data.font = "times"
report.create()
report.add_chapter("Chapter 1")
report.add_sub_chapter("C1")
report.add_text("Hello World.\nlorem ipsum....")
report.add_text("ciao2", True, True)
report.add_empty_line(2)
report.add_page_break()
report.add_chapter("Chapter 2")
```

(continues on next page)

(continued from previous page)

```
report.add_sub_chapter("Charts")
local_path = r'C:\result'
report.add_section(portrait=False, page_format="a3")
report.add_image(os.path.join(local_path, "return_loss.jpg"), width=400, caption="S-Parameters")
report.add_section(portrait=False, page_format="a5")
report.add_table("MyTable", [{"x": "0", "y": "0"}, {"x": "1", "y": "1"}, {"x": "2", "y": "2"}, {"x": "3", "y": "3"}, {"x": "4", "y": "10"}, {"x": "5", "y": "20"}])
report.add_section()
report.add_chart([0, 1, 2, 3, 4, 5], [10, 20, 4, 30, 40, 12], "Freq", "Val", "MyTable")
report.add_toc()
report.save_pdf(r'c:\temp', "report_example.pdf")
```

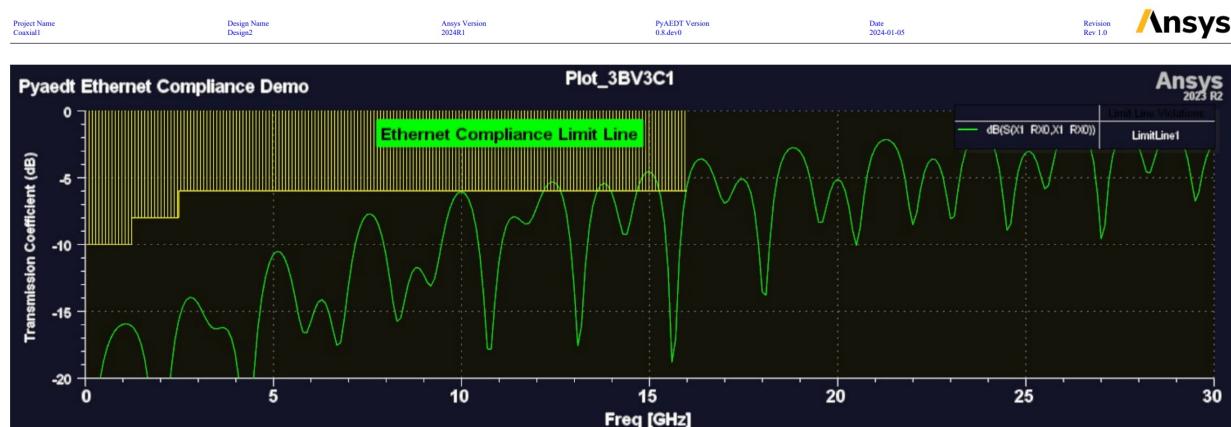


Figure 1. S-Parameters

Copyright (c) 2023, ANSYS Inc. Unauthorised use, distribution or duplication is prohibited

Page 4/7

EMIT modeler

The EMIT Modeling module includes several classes to enable modeling in EMIT:

- `modeler` to return the schematic modeler for an EMIT design.
- `couplings` to return a list of all linked couplings within an EMIT design.
- `version` to provide the EMIT version information.
- `set_units` to set the units globally for the EMIT design.
- `get_units` to get the value of the current EMIT design global units.

EMIT version check and set units example:

```
import pyaedt
from pyaedt import Emit

emit = Emit(pyaedt.generate_unique_project_name(),
            specified_version="2024.1", non_graphical=False,
            new_desktop_session=True, close_on_exit=True)

# This call returns detailed version info for EMIT
ver = emit.version(detailed=True)

# This call sets the global units for EMIT
unit_types = ["Power", "Frequency", "Length", "Time"]
unit_vals = ["kW", "kHz", "meter", "ns"]
emit.set_units(unit_types, unit_vals)

# This call gets all the global units for the EMIT design
all_units = emit.get_units()

# This call gets the global Frequency units for the EMIT design
freq_units = emit.get_units("Frequency")

# Close AEDT
emit.release_desktop(close_projects=True, close_desktop=True)
```

EMIT-HFSS link creation example:

```
import os
import pyaedt
from pyaedt import Emit
from pyaedt.generic.filesystem import Scratch

scratch_path = pyaedt.generate_unique_folder_name()
temp_folder = os.path.join(scratch_path, ("EmitHFSSEExample"))
if not os.path.exists(temp_folder):
    os.mkdir(temp_folder)

# Launch AEDT
aedtapp = pyaedt.launch_desktop(specified_version="2024.1", non_graphical=False,
                                 new_desktop_session=True, close_on_exit=True)

# Verify the ``Cell Phone RFT Defense`` example exists
example_name = "Cell Phone RFI Desense"
example_aedt = example_name + ".aedt"
example_results = example_name + ".aedtresults"
example_lock = example_aedt + ".lock"
example_pdf_file = example_name + " Example.pdf"

example_dir = os.path.join(aedtapp.install_path, "Examples\\EMIT")
example_project = os.path.join(example_dir, example_aedt)
example_results_folder = os.path.join(example_dir, example_results)
example_pdf = os.path.join(example_dir, example_pdf_file)
```

(continues on next page)

(continued from previous page)

```

# If the ``Cell Phone RFT Defense`` example is not
# in the installation directory, exit from this example.
if not os.path.exists(example_project):
    exit()

# Copy the project to a temp directory
my_project = os.path.join(temp_folder, example_aedt)
my_results_folder = os.path.join(temp_folder, example_results)
my_project_lock = os.path.join(temp_folder, example_lock)
my_project_pdf = os.path.join(temp_folder, example_pdf_file)

if os.path.exists(my_project):
    os.remove(my_project)

if os.path.exists(my_project_lock):
    os.remove(my_project_lock)

with Scratch(scratch_path) as local_scratch:
    local_scratch.copyfile(example_project, my_project)
    local_scratch.copyfolder(example_results_folder, my_results_folder)
    if os.path.exists(example_pdf):
        local_scratch.copyfile(example_pdf, my_project_pdf)

emit = Emit(my_project)

# Remove all existing links
for link in emit.couplings.coupling_names:
    emit.couplings.delete_link(link)

# Add the HFSS design as a coupling in EMIT
for link in emit.couplings.linkable_design_names:
    emit.couplings.add_link(link)

# Get all the antennas in the EMIT design
antennas = emit.couplings.antenna_nodes
for ant in antennas:
    print(ant)

# Close AEDT
emit.release_desktop(close_projects=True, close_desktop=True)

```

Create and Analyze an EMIT project:

```

import pyaedt
from pyaedt import Emit
from pyaedt.emit_core.emit_constants import TxRxMode, ResultType

emit = Emit(pyaedt.generate_unique_project_name(),
            specified_version="2024.1", non_graphical=False,
            new_desktop_session=True, close_on_exit=True)

# Create a radio and connect an antenna to it

```

(continues on next page)

(continued from previous page)

```

rad1 = emit.modeler.components.create_component("New Radio")
ant1 = emit.modeler.components.create_component("Antenna")
if rad1 and ant1:
    ant1.move_and_connect_to(rad1)

# Quickly create 2 more radios with antennas automatically
# connected to them
rad2, ant2 = emit.modeler.components.create_radio_antenna("GPS Receiver")
rad3, ant3 = emit.modeler.components.create_radio_antenna("Bluetooth Low Energy (LE)",
    ↴ "Bluetooth")

# Create a new ``Revision``
rev = emit.results.analyze()

# Get the receive bands enabled for the GPS Rx
rx_bands = rev.get_band_names(rad2.name, TxRxMode.RX)

# Get the transmit bands enabled for the Bluetooth radio
tx_bands = rev.get_band_names(rad3.name, TxRxMode.TX)

# Configure the interaction domain that will be analyzed
domain = emit.results.interaction_domain()
domain.set_receiver(rad2.name, rx_bands[0], -1)
domain.set_interferer(rad3.name, tx_bands[0])

# Analyze the domain and get the worst case interference
interaction = rev.run(domain)
worst = interaction.get_worst_instance(ResultType.EMI)
emi = worst.get_value(ResultType.EMI)
print("Worst case interference is: {} dB".format(emi))

# Close AEDT
emit.release_desktop(close_projects=True, close_desktop=True)

```

1.4.4 Client-server

You can launch PyAEDT on a remote machine if these conditions are met:

- PyAEDT is installed on client and server machines. (There is no need to have AEDT installed on the client machine.)
- The same Python version is used on the client and server machines. (CPython 3.7+ or IronPython is embedded in the AEDT installation.)

gRPC connection in AEDT 2022 R2 and later

In AEDT 2022 R2 and later, PyAEDT fully supports the gRPC API (except for EDB):

```
# Launch the latest installed version of AEDT in graphical mode.
from pyaedt import Hfss
from pyaedt import settings
settings.use_grpc_api=True
hfss = Hfss(machine="fullmachinename", port=portnumber)
```

If the `machine` argument is provided and the machine is a remote machine, AEDT must be up and running on the remote server listening on the specified port `portnumber`.

To start AEDT in listening mode on the remote machine:

```
path/to/ANSYSEM/v222/Win64/ansysedt.exe -grpcsrv portnumber #windows
path/to/ANSYSEM/v222/Lin64/ansysedt -grpcsrv portnumber #linux
```

If the connection is local, the `machine` argument must be left empty. PyAEDT then starts the AEDT session automatically. Machine and port arguments are available to all applications except EDB.

PyAEDT remote service manager

PyAEDT includes a service manager that can be run on the server machine and can be launched on-demand in AEDT sessions and act as a file manager. You can make a remote application call on a CPython server or any Windows client machine in AEDT 2022 R2 and later.

On a CPython Server run the `pyaedt_service_manager` service that listens on port 17878 for incoming requests of connections from clients. The port is configurable. Requirements:

- Python 3.7+ Virtual Environment.
- `pyaedt > 0.6.0`

On Linux, in addition to the preceding requirements, these environments are needed: - You can use the CPython version in the AEDT installation folder if you first add the Python library folder to the `LD_LIBRARY_PATH` environment variable. - You can use the Python 3.7 or later version that is installed. - You can export `ANSYSEM_ROOT222=/path/to/AnsysEM/v222/Linux64`. - You can export `LD_LIBRARY_PATH=$ANSYSEM_ROOT222/common/mono/Linux64/lib:$ANSYSEM_ROOT222/Delcross:$LD_LIBRARY_PATH`.

On the server, the `pyaedt_service_manager` service listen for incoming connections:

```
# Launch PyAEDT remote server on CPython
from pyaedt.common_rpc import pyaedt_service_manager
pyaedt_service_manager()
```

On any client machine, the user must establish the connection as shown in following example. AEDT can be launched directly while creating the session or after the connection is established.

```
from pyaedt.common_rpc import create_session
# User can establish the connection and start a new AEDT session
c11 = create_session("server_name", launch_aedt_on_server=True, aedt_port=17880, non_
graphical=True)

# Optionally AEDT can be launched after the connection is established
c12 = create_session("server_name", launch_aedt_on_server=False)
c12.aedt(port=17880, non_graphical=True)
```

Once AEDT is started then user can connect an application to it.

```
hfss = Hfss(machine=cl1.server_name, port=cl1.aedt_port)
# your code here
```

The client can be used also to upload or download files from the server.

```
cl1.filemanager.upload(local_path, remote_path)
file_content = cl1.open_file(remote_file)
```

1.4.5 Versions and interfaces

The PyAEDT project attempts to maintain compatibility with legacy versions of AEDT while allowing for support of faster and better interfaces with the latest versions of AEDT.

There are two interfaces PyAEDT can use to connect to AEDT. You can see a table with the AEDT version and the supported interfaces in *Table of supported versions*

gRPC interface

This is the default and preferred interface to connect to AEDT. Ansys 2022 R2 and later support the latest gRPC interface, allowing for remote management of MAPDL with rapid streaming of mesh, results, and files from the MAPDL service.

Legacy interfaces

COM interface

AnsysEM supports the legacy COM interface, enabled with the settings option.

This interface works only on Windows and uses .NET COM objects.

```
from pyaedt import settings
settings.use_grpc_api = False
```

Compatibility between AEDT and interfaces

The following table shows the supported versions of Ansys EDT and the recommended interface for each one of them in PyAEDT.

Table of supported versions

Ansys Version	Recommended interface	Support	
		gRPC	COM
AnsysEM 2024 R1	gRPC	YES	NO*
AnsysEM 2023 R2	gRPC	YES	YES*
AnsysEM 2023 R1	gRPC	YES	YES*
AnsysEM 2022 R2	gRPC	YES*	YES
AnsysEM 2022 R1	gRPC	NO	YES
AnsysEM 2021 R2	gRPC	NO YES	

Where:

- YES means that the interface is supported and recommended.
- YES* means that the interface is supported, but not recommended. Their support might be dropped in the future.
- NO means that the interface is not supported.
- NO* means that the interface is still supported but it is deprecated.

1.4.6 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, paying particular attention to [Guidelines and Best Practices](#), before attempting to contribute to PyAEDT.

The following contribution information is specific to PyAEDT.

Clone the repository

To clone and install the latest version of PyAEDT in development mode, run:

```
git clone https://github.com/ansys/pyaedt
cd pyaedt
python -m pip install --upgrade pip
pip install -e .
```

Post issues

Use the [PyAEDT Issues](#) page to submit questions, report bugs, and request new features.

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

View PyAEDT documentation

Documentation for the latest stable release of PyAEDT is hosted at [PyAEDT 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.

Code style

PyAEDT complies with the [PyAnsys code style](#). `pre-commit` is applied within the CI/CD to ensure compliance. The `pre-commit` Python package can be installed and run as follows:

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

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

```
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
debug statements (python).....Passed
trim trailing whitespace.....Passed
Validate GitHub Workflows.....Passed
blacken-docs.....Passed
```

Naming conventions

Consistency of names helps improve readability and ease of use. Starting with release 0.8 a concerted effort has been made to improve consistency of naming and adherence to :ref:`PEP-8<<https://peps.python.org/pep-0008/>>`.

For example, methods used to create or access entities in AEDT require that a name be passed to the method or function as an argument. It is tempting to include context as part of that variable name. For example, while it is tempting to use `setupname` as an argument to :meth:`Hfss.create_setup`, the context setup is explicitly defined by the method name. The variable name provides a more compact description of the variable in this context.

In previous PyAEDT versions, you can also find both `setup_name` and `setupname` used for various methods or classes. Improving naming consistency improves maintainability and readability.

The following table illustrates the recommended conventions:

Table 2: Keywords and object names

Old name	New name	Example	
<code>setupname</code> ,	<code>name</code>	<code>Hfss.create_setup()</code> ,	<code>Hfss.</code>
<code>setup_name</code> , <code>sweepname</code>		<code>create_linear_step_sweep()</code>	
<code>usethickness</code>	<code>thickness</code>	<code>Hfss.assign_coating()</code>	
<code>entities</code>	<code>assignment</code>	<code>Maxwell.assign_current_density()</code>	
<code>entity_list</code>	<code>assignment</code>	<code>Maxwell.assign_symmetry()</code>	

Take care to use descriptive names for variables and classes that adhere to PEP-8 and are consistent with conventions already used in PyAEDT.

Log errors

PyAEDT has an internal logging tool named `Messenger` and a log file that is automatically generated in the project folder.

The following examples demonstrate 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 demonstrate 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.")
```

Handle exceptions

PyAEDT uses a specific decorator, `@pyaedt_function_handler`, to handle exceptions caused by methods and by the AEDT API. This exception handler decorator makes PyAEDT fault tolerant to errors that can occur in any method.

For example:

```
@pyaedt_function_handler()
def my_method(self, var):
    pass
```

Every method can return a value of `True` when successful or `False` when failed. When a failure occurs, the error handler returns information about the error in both the console and log file.

Here is an example of an error:

```
PyAEDT error on method create_box: General or AEDT error. Check again
the arguments provided:
position = [0, 0, 0]
dimensions_list = [0, 10, 10]
name = None
material = None

(-2147352567, 'Exception occurred.', (0, None, None, None, 0, -2147024381), None)
File "C:\GIT\repos\AnsysAutomation\PyAEDT\Primitives.py", line 1930, in create_box
    o.name = self.oeditor.createbox(vArg1, vArg2)

*****
Method Docstring:
```

Create a box.

Parameters

...

Hard-coded values

Do not write hard-coded values to the registry. Instead, use the Configuration service.

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.

1.4.7 About PyAnsys and AEDT

PyAnsys

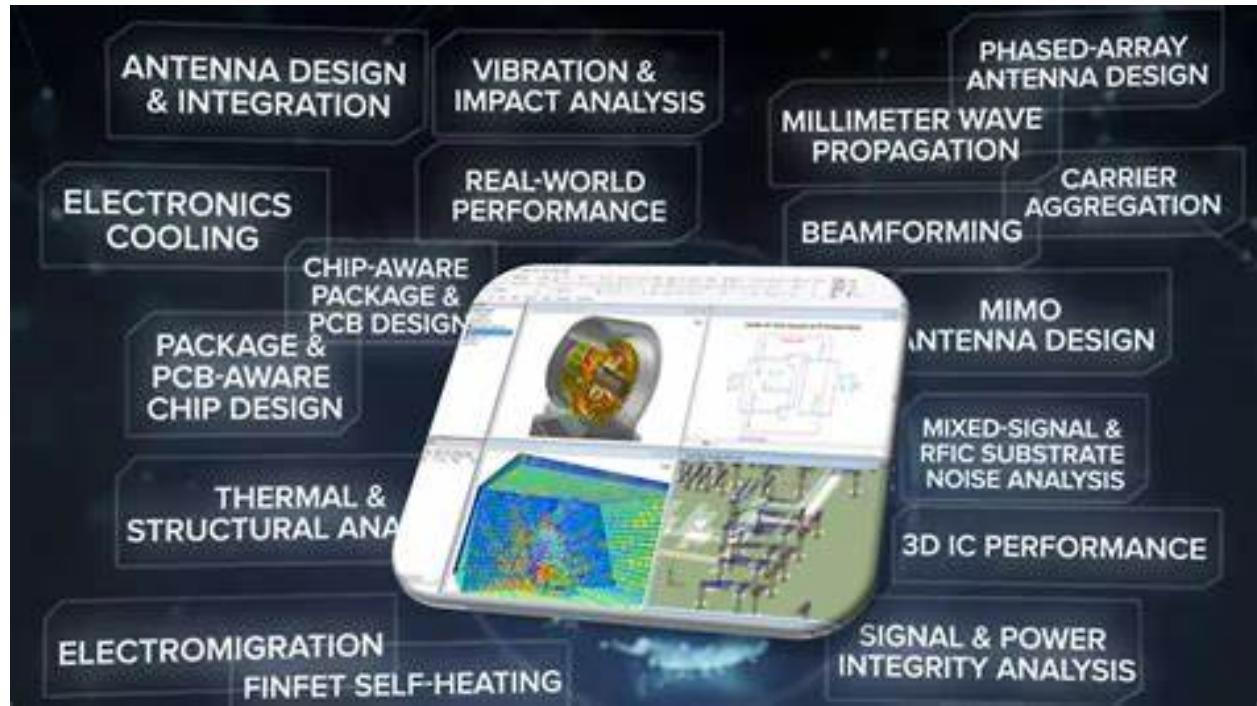
PyAEDT is part of the larger [PyAnsys](#) effort to facilitate the use of Ansys technologies directly from Python.

PyAEDT is intended to consolidate and extend all existing functionalities around scripting for AEDT to allow reuse of existing code, sharing of best practices, and increased collaboration.

About AEDT

AEDT is a platform that enables true electronics system design. AEDT provides access to the Ansys gold-standard electro-magnetics simulation solutions, such as Ansys HFSS, Ansys Maxwell, Ansys Q3D Extractor, Ansys Siwave, and Ansys Icepak using electrical CAD (ECAD) and Mechanical CAD (MCAD) workflows.

In addition, AEDT includes direct links to the complete Ansys portfolio of thermal, fluid, and mechanical solvers for comprehensive multiphysics analysis. Tight integration among these solutions provides unprecedented ease of use for setup and faster resolution of complex simulations for design and optimization.



PyAEDT is licensed under the [MIT License](#).

PyAEDT includes functionality for interacting with the following AEDT tools and Ansys products:

- HFSS and HFSS 3D Layout
- Icepak
- Maxwell 2D, Maxwell 3D, and RMXprt

- 2D Extractor and Q3D Extractor
- Mechanical
- Nexxim
- EDB
- Twin Builder

Dependencies

To run PyAEDT, you must have a local licensed copy of AEDT. PyAEDT supports AEDT versions 2022 R2 and later.

Student version

PyAEDT supports AEDT Student versions 2022 R2 and later. For more information, see the [Ansys Electronics Desktop Student - Free Software Download](#) page on the Ansys website.

Why PyAEDT?

A quick and easy approach for automating a simple operation in the AEDT UI is to record and reuse a script. However, here are some disadvantages of this approach:

- Recorded code is dirty and difficult to read and understand.
- Recorded scripts are difficult to reuse and adapt.
- Complex coding is required by many global users of AEDT.

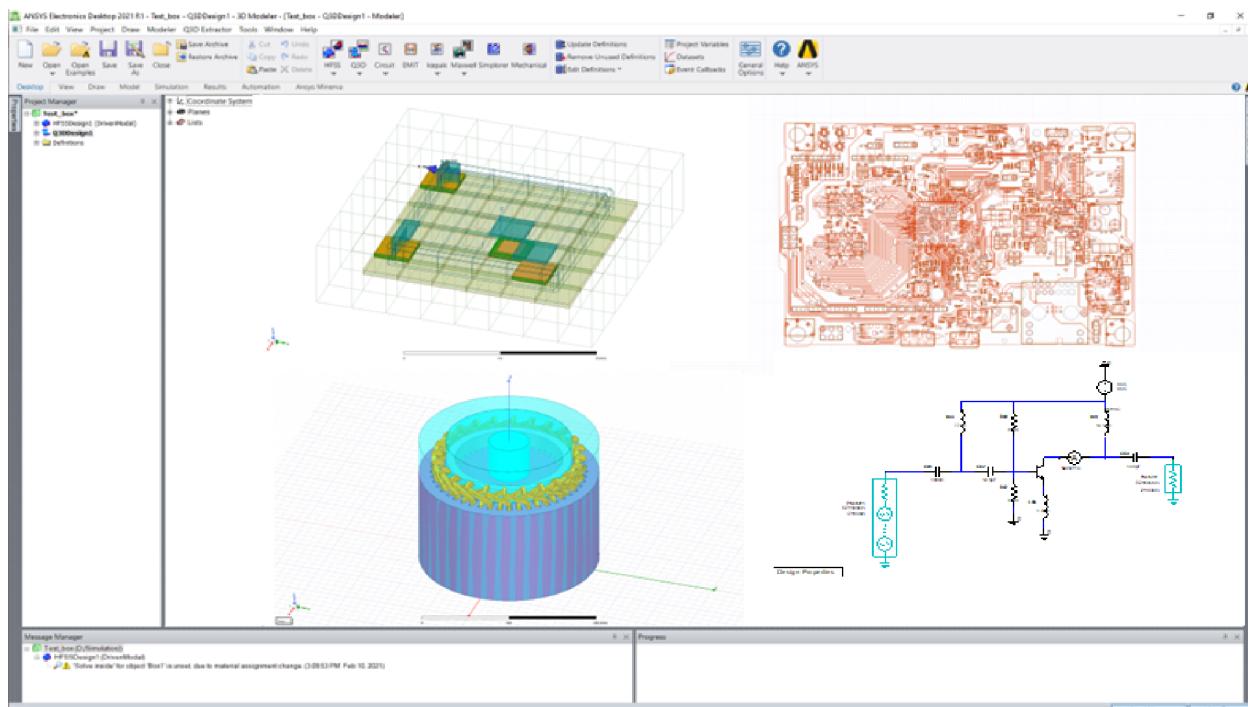
Here are the main advantages that PyAEDT provides:

- Automatic initialization of all AEDT objects, such as desktop objects like the editor, boundaries, and more
- Error management
- Log management
- Variable management
- Compatibility with IronPython (limited) and CPython
- Simplification of complex API syntax using data objects while maintaining PEP8 compliance.
- Code reusability across different solvers
- Clear documentation on functions and API
- Unit tests of code to increase quality across different AEDT versions

CHAPTER TWO

API REFERENCE

This section describes PyAEDT core classes, methods, and functions for AEDT apps and modules. Use the search feature or click links to view API documentation. The Ansys Electronics Desktop (AEDT) is a platform that enables true electronics system design. [AEDT](#) provides access to the Ansys gold-standard electro-magnetics simulation solutions such as Ansys HFSS, Ansys Maxwell, Ansys Q3D Extractor, Ansys Siwave, and Ansys Icepak using electrical CAD (ECAD) and Mechanical CAD (MCAD) workflows. In addition, it includes direct links to the complete Ansys portfolio of thermal, fluid, and Mechanical solvers for comprehensive multiphysics analysis. Tight integration among these solutions provides unprecedented ease of use for setup and faster resolution of complex simulations for design and optimization.



The PyAEDT API includes classes for apps and modules. You must initialize the PyAEDT app to get access to all modules and methods. Available apps are:

- HFSS
- HFSS 3D Layout
- Maxwell 3D
- Maxwell 2D
- Maxwell Circuit

- [Q3D](#)
- [Q2D Extractor](#)
- [Icepak](#)
- [Mechanical](#)
- [RMXprt](#)
- [EMIT](#)
- [Circuit](#)
- [TwinBuilder](#)

All other classes and methods are inherited into the app class. The desktop app is implicitly launched in any of the other applications. Before accessing a PyAEDT app, the desktop app has to be launched and initialized. The desktop app can be explicitly or implicitly initialized as shown in the following examples.

Example with Desktop class explicit initialization:

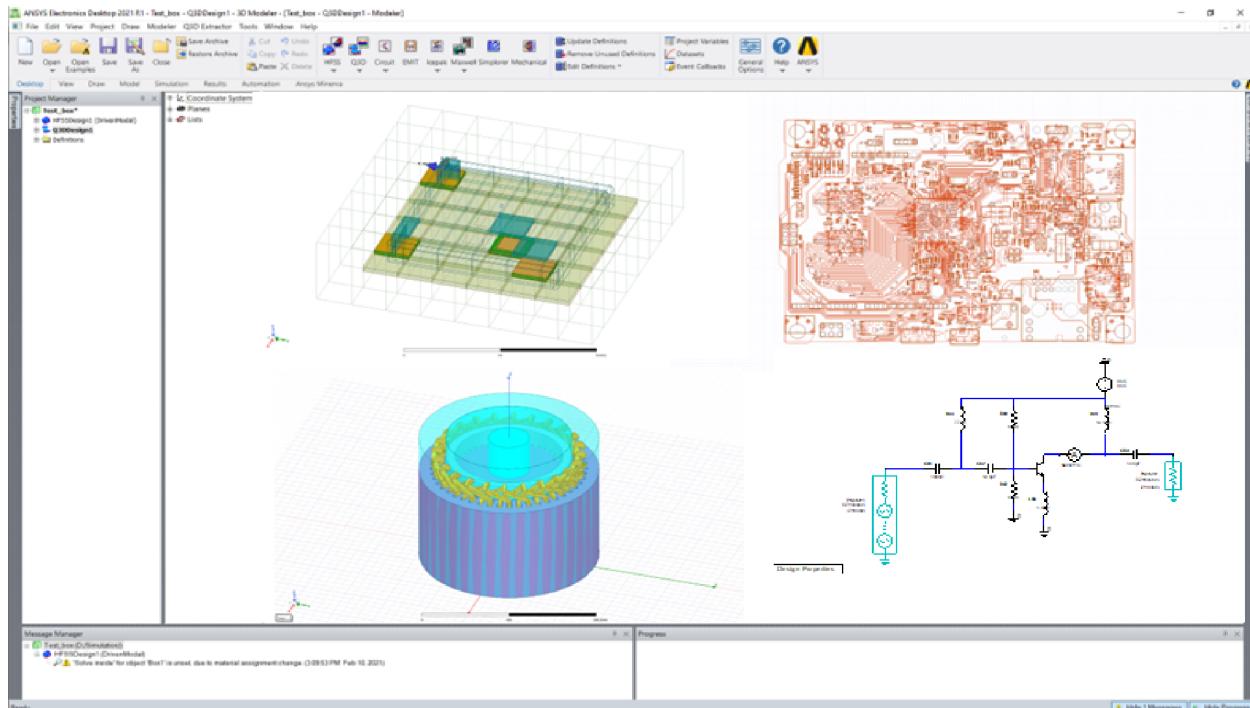
```
from pyaedt import launch_desktop, Circuit
d = launch_desktop(specified_version="2023.1",
                     non_graphical=False,
                     new_desktop_session=True,
                     close_on_exit=True,
                     student_version=False):
    circuit = Circuit()
    ...
    # Any error here should be caught by the desktop app.
    ...
d.release_desktop()
```

Example with Desktop class implicit initialization:

```
from pyaedt import Circuit
circuit = Circuit(specified_version="2023.1",
                  non_graphical=False,
                  new_desktop_session=True,
                  close_on_exit=True,
                  student_version=False):
    circuit = Circuit()
    ...
    # Any error here should be caught by the desktop app.
    ...
circuit.release_desktop()
```

2.1 Application and solvers

The PyAEDT API includes classes for different applications available in Ansys Electronics Desktop (AEDT). You must initialize AEDT to get access to all PyAEDT modules and methods.



Available PyAEDT apps are:

<code>pyaedt.desktop.Desktop(*args, **kwargs)</code>	Provides the Ansys Electronics Desktop (AEDT) interface.
<code>pyaedt.hfss.Hfss([projectname, designname, ...])</code>	Provides the HFSS application interface.
<code>pyaedt.q3d.Q3d([projectname, designname, ...])</code>	Provides the Q3D app interface.
<code>pyaedt.q3d.Q2d([projectname, designname, ...])</code>	Provides the Q2D app interface.
<code>pyaedt.maxwell.Maxwell2d([projectname, ...])</code>	Provides the Maxwell 2D app interface.
<code>pyaedt.maxwell.Maxwell3d([projectname, ...])</code>	Provides the Maxwell 3D app interface.
<code>pyaedt.icepak.Icepak([projectname, ...])</code>	Provides the Icepak application interface.
<code>pyaedt.hfss3dlayout.Hfss3dLayout([...])</code>	Provides the HFSS 3D Layout application interface.
<code>pyaedt.mechanical.Mechanical([projectname, ...])</code>	Provides the Mechanical application interface.
<code>pyaedt.rmxprt.Rmxprt([projectname, ...])</code>	Provides the RMxprt app interface.
<code>pyaedt.circuit.Circuit([projectname, ...])</code>	Provides the Circuit application interface.
<code>pyaedt.maxwellcircuit.MaxwellCircuit([...])</code>	Provide the Maxwell Circuit application interface.
<code>pyaedt.emit.Emit([projectname, designname, ...])</code>	Provides the EMIT application interface.
<code>pyaedt.twinbuilder.TwinBuilder([...])</code>	Provides the Twin Builder application interface.

2.1.1 pyaedt.desktop.Desktop

```
class pyaedt.desktop.Desktop(*args, **kwargs)
```

Provides the Ansys Electronics Desktop (AEDT) interface.

Parameters

specified_version

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active setup or latest installed version is used. Examples of input values are `232`, `23.2`, `2023.2`, `2023.2`.

non_graphical

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `True`.

close_on_exit

[`bool`, optional] Whether to close AEDT on exit. The default is `True`. This option is used only when Desktop is used in a context manager (`with` statement). If Desktop is used outside a context manager, see the `release_desktop` arguments.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This parameter works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number on which to start the oDesktop communication on the already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Launch AEDT 2023 R1 in non-graphical mode and initialize HFSS.

```
>>> import pyaedt
>>> desktop = pyaedt.Desktop(specified_version="2023.2", non_graphical=True)
PyAEDT INFO: pyaedt v...
PyAEDT INFO: Python version ...
>>> hfss = pyaedt.Hfss(designname="HFSSDesign1")
PyAEDT INFO: Project...
PyAEDT INFO: Added design 'HFSSDesign1' of type HFSS.
```

Launch AEDT 2023 R2 in graphical mode and initialize HFSS.

```
>>> desktop = Desktop(232)
PyAEDT INFO: pyaedt v...
PyAEDT INFO: Python version ...
>>> hfss = pyaedt.Hfss(designname="HFSSDesign1")
PyAEDT INFO: No project is defined. Project...
```

`__init__(specified_version=None, non_graphical=False, new_desktop_session=True, close_on_exit=True, student_version=False, machine='', port=0, aedt_process_id=None)`

Methods

active_design([project_object, name, ...])	Get the active design.
active_project([name])	Get the active project.
analyze_all([project, design])	Analyze all setups in a project.
change_active_dso_config_name([...])	Change a specific registry key to a new value.
change_license_type([license_type])	Change the license type.
change_registry_from_file(registry_file[, ...])	Apply desktop registry settings from an ACF file.
change_registry_key(key_full_name, key_value)	Change an AEDT registry key to a new value.
clear_messages()	Clear all AEDT messages.
close_desktop()	Close all projects and shut down AEDT.
design_list([project])	Get a list of the designs.
design_type([project_name, design_name])	Get the type of design.
disable_autosave()	Disable the autosave option.
download_job_results(job_id, project_path, ...)	Download job results to a specific folder from Ansys Cloud.
enable_autosave()	Enable the autosave option.
get_ansyscloud_job_info([job_id, job_name])	Monitor a job submitted to Ansys Cloud.
get_available_cloud_config([region])	Get available Ansys Cloud machines configuration.
get_available_toolkits()	Get toolkit ready for installation.
get_monitor_data()	Check and get monitor data of an existing analysis.
load_project(project_file[, design_name])	Open an AEDT project based on a project and optional design.
project_list()	Get a list of projects.
project_path([project_name])	Get the path to the project.
release_desktop([close_projects, close_on_exit])	Release AEDT.
save_project([project_name, project_path])	Save the project.
select_scheduler(scheduler_type[, address, ...])	Select a scheduler to submit the job.
stop_simulations([clean_stop])	Check if there are simulation running and stops them.
submit_ansys_cloud_job(project_file, ...[, ...])	Submit a job to be solved on a cluster.
submit_job(project_file, clustername[, ...])	Submit a job to be solved on a cluster.

Attributes

are_there_simulations_running	Check if there are simulation running.
current_student_version	Current AEDT student version.
current_version	Current AEDT version.
install_path	Installation path for AEDT.
installed_versions	Dictionary of AEDT versions installed on the system and their installation paths.
logger	AEDT logger.
messenger	Messenger manager for the AEDT logger.
odesktop	AEDT instance containing all projects and designs.
personallib	PersonalLib directory.
pyaedt_dir	PyAEDT directory.
src_dir	Python source directory.
syslib	SysLib directory.
userlib	UserLib directory.

2.1.2 pyaedt.hfss.Hfss

```
class pyaedt.hfss.Hfss(projectname=None, designname=None, solution_type=None, setup_name=None,
specified_version=None, non_graphical=False, new_desktop_session=False,
close_on_exit=False, student_version=False, machine='', port=0,
aedt_process_id=None)
```

Provides the HFSS application interface.

This class allows you to create an interactive instance of HFSS and connect to an existing HFSS design or create a new HFSS design if one does not exist.

Parameters

projectname

[str, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is None, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[str, optional] Name of the design to select. The default is None, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[str, optional] Solution type to apply to the design. The default is None, in which case the user-defined default type is applied. Options are:

- Terminal
- Modal
- SBR+
- Transient
- Eigenmode

setup_name

[str, optional] Name of the setup to use as the nominal. The default is None, in which case the active setup is used or nothing is used.

specified_version

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active version or latest installed version is used. This parameter is ignored when a script is launched within AEDT. Examples of input values are `232`, `23.2`, `"2023.2"`, `"2023.2"`.

non_graphical

[`bool`, optional] Whether to run AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`. This parameter is ignored when a script is launched within AEDT.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This parameter works only on 2022 R2 or later. The remote Server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server starts if it is not present.

port

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of HFSS and connect to an existing HFSS design or create a new HFSS design if one does not exist.

```
>>> from pyaedt import Hfss
>>> hfss = Hfss()
PyAEDT INFO: No project is defined...
PyAEDT INFO: Active design is set to...
```

Create an instance of HFSS and link to a project named `HfssProject`. If this project does not exist, create one with this name.

```
>>> hfss = Hfss("HfssProject")
PyAEDT INFO: Project HfssProject has been created.
PyAEDT INFO: No design is present. Inserting a new design.
PyAEDT INFO: Added design ...
```

Create an instance of HFSS and link to a design named `HfssDesign1` in a project named `HfssProject`.

```
>>> hfss = Hfss("HfssProject", "HfssDesign1")
PyAEDT INFO: Added design 'HfssDesign1' of type HFSS.
```

Create an instance of HFSS and open the specified project, which is named "`myfile.aedt`".

```
>>> hfss = Hfss("myfile.aedt")
PyAEDT INFO: Project myfile has been created.
PyAEDT INFO: No design is present. Inserting a new design.
PyAEDT INFO: Added design...
```

Create an instance of HFSS using the 2023 R2 release and open the specified project, which is named "`myfile2.aedt`".

```
>>> hfss = Hfss(specified_version=232, projectname="myfile2.aedt")
PyAEDT INFO: Project myfile2 has been created.
PyAEDT INFO: No design is present. Inserting a new design.
PyAEDT INFO: Added design...
```

Create an instance of HFSS using the 2023 R2 student version and open the specified project, which is named "`myfile3.aedt`".

```
>>> hfss = Hfss(specified_version="2023.2", projectname="myfile3.aedt", student_
    ↵version=True)
PyAEDT INFO: Project myfile3 has been created.
PyAEDT INFO: No design is present. Inserting a new design.
PyAEDT INFO: Added design...
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,
        specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,
        student_version=False, machine="", port=0, aedt_process_id=None)
```

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name)</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name)</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.
<code>add_3d_component_array_from_json(input_data)</code>	Add or edit a 3D component array from a JSON file, TOML file, or dictionary.
<code>add_error_message(message_text[, message_type])</code>	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	Add a new toolkit to the current application.
<code>add_info_message(message_text[, message_type])</code>	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>	Create a stackup 3D object.

continues on next page

Table 1 – continued from previous page

<code>add_warning_message(message_text[, message_type])</code>	mes-	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>		Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>		Analyze a design setup.
<code>archive_project([project_file, ...])</code>		Archive the AEDT project and add a message.
<code>assign_coating(assignment[, material, ...])</code>		Assign finite conductivity to one or more objects or faces of a given material.
<code>assign_current_source_to_sheet(assignment[, ...])</code>		Create a current source taking one sheet.
<code>assign_febi(assignment[, name])</code>		Assign an FE-BI region to one or more objects.
<code>assign_hybrid_region(assignment[, name, ...])</code>		Assign a hybrid region to one or more objects.
<code>assign_impedance_to_sheet(assignment[, ...])</code>		Create an impedance taking one sheet.
<code>assign_lattice_pair(assignment[, reverse_v, ...])</code>		Assign a lattice pair to a couple of faces.
<code>assign_lumped_rlc_to_sheet(assignment[, ...])</code>		Create a lumped RLC taking one sheet.
<code>assign_material(obj, mat)</code>		Assign a material to one or more objects.
<code>assign_perfecte_to_sheets(assignment[, ...])</code>		Create a Perfect E taking one sheet.
<code>assign_perfecth_to_sheets(assignment[, name])</code>		Assign a Perfect H to sheets.
<code>assign_primary(assignment, u_start, u_end[, ...])</code>		Assign the primary boundary condition.
<code>assign_radiation_boundary_to_faces(assignment[, ...])</code>		Assign a radiation boundary to one or more faces.
<code>assign_radiation_boundary_to_objects(assignment[, ...])</code>		Assign a radiation boundary to one or more objects (usually airbox objects).
<code>assign_secondary(assignment, primary, ...[, ...])</code>		Assign the secondary boundary condition.
<code>assign_symmetry(assignment[, name, is_perfect_e])</code>		Assign symmetry to planar entities.
<code>assign_voltage_source_to_sheet(assignment[, ...])</code>		Create a voltage source taking one sheet.
<code>assignmaterial_from_sherlock_files(...)</code>		Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>auto_assign_lattice_pairs(assignment[, ...])</code>		Assign lattice pairs to a geometry automatically.
<code>autosave_disable()</code>		Disable autosave in AEDT.
<code>autosave_enable()</code>		Enable autosave in AEDT.
<code>change_automatically_use_causal_materials()</code>		Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>		Set Design Settings.
<code>change_material_override([material_override])</code>		Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>		Change a property.
<code>change_validation_settings([...])</code>		Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>		Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>		Check if a project path is already loaded in active Desktop.
<code>circuit_port(assignment, reference[, ...])</code>		Create a circuit port from two objects.
<code>clean_proj_folder([directory, name])</code>		Delete a project folder.
<code>cleanup_solution([variations, ...])</code>		Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>		Close AEDT and release it.
<code>close_project([name, save_project])</code>		Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	de-	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>		Copy the project to another destination.

continues on next page

Table 1 – continued from previous page

<code>copy_solid_bodies_from(design[, ...])</code>	Copy a list of objects and user defined models from one design to the active design.
<code>create_boundary([boundary_type, assignment, ...])</code>	Assign a boundary condition to a sheet or surface. This method is generally
<code>create_current_source_from_objects(...[, ...])</code>	Create a current source taking the closest edges of two objects.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>	Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>	Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>	Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>	Create a 3D dataset.
<code>create_floquet_port(assignment[, ...])</code>	Create a floquet port on a face.
<code>create_impedance_between_objects(...[, ...])</code>	Create an impedance taking the closest edges of two objects.
<code>create_linear_count_sweep(setup, units, ...)</code>	Create a sweep with a specified number of points.
<code>create_linear_step_sweep(setup, unit, ...[, ...])</code>	Create a sweep with a specified frequency step.
<code>create_lumped_rc_between_objects(...[, ...])</code>	Create a lumped RLC taking the closest edges of two objects.
<code>create_new_project(proj_name)</code>	Create a project within AEDT.
<code>create_open_region([frequency, boundary, ...])</code>	Create an open region on the active editor.
<code>create_output_variable(variable, expression)</code>	Create or modify an output variable.
<code>create_perfecte_from_objects(assignment, ...)</code>	Create a Perfect E taking the closest edges of two objects.
<code>create_perfecth_from_objects(assignment, ...)</code>	Create a Perfect H taking the closest edges of two objects.
<code>create_qfactor_report(project_dir, output, ...)</code>	Export a CSV file of the EigenQ plot.
<code>create_sbr_antenna([antenna_type, ...])</code>	Create a parametric beam antennas in SBR+.
<code>create_sbr_chirp_i_doppler_setup([...])</code>	Create an SBR+ Chirp I setup.
<code>create_sbr_chirp_iq_doppler_setup([...])</code>	Create an SBR+ Chirp IQ setup.
<code>create_sbr_file_based_antenna(far_field_data)</code>	Create a linked antenna.
<code>create_sbr_linked_antenna(assignment[, ...])</code>	Create a linked antennas.
<code>create_sbr_pulse_doppler_setup([...])</code>	Create an SBR+ pulse Doppler setup.
<code>create_sbr_radar_from_json(radar_file, name)</code>	Create an SBR+ radar setup from a JSON file.
<code>create_scattering([plot, sweep, ports, ...])</code>	Create an S-parameter report.
<code>create_setup([name, setup_type])</code>	Create an analysis setup for HFSS.
<code>create_single_point_sweep(setup, unit, freq)</code>	Create a sweep with a single frequency point.
<code>create_source_excitation(assignment, point1, ...)</code>	Create a source excitation.
<code>create_spiral_lumped_port(assignment, reference)</code>	Create a spiral lumped port between two adjacent objects.
<code>create_voltage_source_from_objects(...[, ...])</code>	Create a voltage source taking the closest edges of two objects.
<code>dataset_exists(name[, is_project_dataset])</code>	Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>	Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>	Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>	Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>	Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>	Delete a design from the current project.
<code>delete_project(project_name)</code>	Delete a project.
<code>delete_separator(separator_name)</code>	Delete a separator from either the active project or a design.
<code>delete_setup(name)</code>	Delete a setup.
<code>delete_unused_variables()</code>	Delete design and project unused variables.

continues on next page

Table 1 – continued from previous page

<code>delete_variable(sVarName)</code>	Delete a variable.
<code>design_variation([variation_string])</code>	Generate a string to specify a desired variation.
<code>duplicate_design(label[, save_after_duplicate])</code>	Copy a design to a new name.
<code>edit_setup(name, properties)</code>	Modify a setup.
<code>edit_source([assignment, power, phase])</code>	Set up the power loaded for HFSS postprocessing.
<code>edit_source_from_file(assignment, file_name)</code>	Edit a source from file data.
<code>edit_sources(assignment[, ...])</code>	Set up the power loaded for HFSS postprocessing in multiple sources simultaneously.
<code>edit_sources_from_file(file_name)</code>	Update all sources from a csv.
<code>evaluate_expression(expression_string)</code>	Evaluate a valid string expression and return the numerical value in SI units.
<code>export_3d_model([file_name, file_path, ...])</code>	Export the 3D model.
<code>export_convergence(setup_name[, ...])</code>	Export a solution convergence to a file.
<code>export_design_preview_to_jpg(filename)</code>	Export design preview image to a JPG file.
<code>export_mesh_stats(setup[, variation_string, ...])</code>	Export mesh statistics to a file.
<code>export_parametric_results(sweep, filename[, ...])</code>	Export a list of all parametric variations solved for a sweep to a CSV file.
<code>export_profile(setup_name[, ...])</code>	Export a solution profile to a PROF file.
<code>export_results([analyze, export_folder, ...])</code>	Export all available reports to a file, including profile, and convergence and sNp when applicable.
<code>export_rl_matrix(matrix_name, file_path[, ...])</code>	Export R/L matrix after solving.
<code>export_touchstone([setup_name, sweep_name, ...])</code>	Export a Touchstone file.
<code>export_variables_to_csv(filename[, ...])</code>	Export design properties, project variables, or both to a CSV file.
<code>flatten_3d_components([component_name, ...])</code>	Flatten one or multiple 3d components in the actual layout.
<code>generate_temp_project_directory(subdir_name)</code>	Generate a unique directory string to save a project to.
<code>generate_unique_setup_name([setup_name])</code>	Generate a new setup with an unique name.
<code>get_all_conductors_names()</code>	Retrieve all conductors in the active design.
<code>get_all_dielectrics_names()</code>	Retrieve all dielectrics in the active design.
<code>get_all_insertion_loss_list([trlist, ...])</code>	Get a list of all insertion losses from two lists of excitations (driver and receiver).
<code>get_all_return_loss_list([excitation_names, ...])</code>	Get a list of all return losses for a list of excitations.
<code>get_all_sources()</code>	Retrieve all setup sources.
<code>get_antenna_ffd_solution_data(frequencies[, ...])</code>	Export the antenna parameters to Far Field Data (FFD) files and return an instance of the <code>FfdSolutionDataExporter</code> object.
<code>get_components3d_vars(component3dname)</code>	Read the A3DCOMP file and check for variables.
<code>get_dxf_layers(file_path)</code>	Read a DXF file and return all layer names.
<code>get_evaluated_value(variable_name[, units])</code>	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
<code>get_fext_xtalk_list([trlist, reclist, ...])</code>	Get a list of all the far end XTalks from two lists of excitations (driver and receiver).
<code>get_hdm_plotter([file_name])</code>	Get the HDM plotter``.
<code>get_module(module_name)</code>	Aedt Module object.
<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_next_xtalk_list([trlist, tx_prefix, ...])</code>	Get a list of all the near end XTalks from a list of excitations (driver and receiver).
<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.

continues on next page

Table 1 – continued from previous page

get_object_material_properties([assignment, ...])	Retrieve the material properties for a list of objects and return them in a dictionary.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.
get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_output_variable(variable[, solution])	Retrieve the value of the output variable.
get_property_value(objectname, property[, type])	Retrieve a property value.
get_registry_key_int(key_full_name)	Get the value for the AEDT registry key if one exists.
get_registry_key_string(key_full_name)	Get the value for the AEDT registry key if one exists.
get_setup(name)	Get the setup from the current design.
get_setups()	Retrieve setups.
get_sweeps(name)	Retrieve all sweeps for a setup.
get_touchstone_data([setup_name, ...])	Return a Touchstone data plot.
get_traces_for_plot([get_self_terms, ...])	Retrieve a list of traces of specified designs ready to use in plot reports.
hidden_variable(variable_name[, value])	Set the variable to a hidden or unhidden variable.
identify_touching_conductors([object_name])	Identify all touching components and group in a dictionary.
import_dataset1d(filename[, dsname, ...])	Import a 1D dataset.
import_dataset3d(filename[, dsname, ...])	Import a 3D dataset.
import_dxf(file_path, layers_list[, ...])	Import a DXF file.
import_gds_3d(input_file, mapping_layers[, ...])	Import a GDSII file.
insert_design([design_name, solution_type])	Add a design of a specified type.
insert_infinite_sphere([definition, ...])	Create an infinite sphere.
insert_near_field_box([u_length, u_samples, ...])	Create a near field box.
insert_near_field_line(assignment[, points, ...])	Create a near field line.
insert_near_field_rectangle([u_length, ...])	Create a near field rectangle.
insert_near_field_sphere([radius, ...])	Create a near field sphere.
list_of_variations([setup_name, sweep_name])	Retrieve a list of active variations for input setup.
load_project(project_file[, design_name, ...])	Open an AEDT project based on a project and optional design.
lumped_port(assignment[, reference, ...])	Create a waveport taking the closest edges of two objects.
number_with_units(value[, units])	Convert a number to a string with units.
parse_hdm_file(file_name)	Parse an HFSS SBR+ or Creeping Waves hdm file.
plot([objects, show, export_path, ...])	Plot the model or a subset of objects.
read_design_data()	Read back the design data as a dictionary.
read_only_variable(variable_name[, value])	Set the variable to a read-only or not read-only variable.
release_desktop([close_projects, close_desktop])	Release AEDT.
remove_all_unused_definitions()	Remove all unused definitions in the project.
rename_design(new_name[, save_after_duplicate])	Rename the active design.
sar_setup([assignment, tissue_mass, ...])	Define SAR settings.
save_project([project_file, overwrite, ...])	Save the project and add a message.
set_active_design(name)	Change the active design to another design.

continues on next page

Table 1 – continued from previous page

<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_auto_open([enable, opening_type])</code>	Set the HFSS auto open type.
<code>set_differential_pair(assignment, reference)</code>	Add a differential pair definition.
<code>set_export_touchstone(activate[, export_dir])</code>	Set automatic export of the Touchstone file after simulation.
<code>set_impedance_multiplier(multiplier)</code>	Set impedance multiplier.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_material_threshold([threshold])</code>	Set the material conductivity threshold.
<code>set_mesh_fusion_settings([assignment, ...])</code>	Set mesh fusion settings in HFSS.
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_phase_center_per_port([coordinate_system])</code>	Set phase center per port.
<code>set_radiated_power_calc_method([method])</code>	Set the radiated power calculation method in Hfss.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_sbr_current_sources_options([...])</code>	Set SBR+ setup options for the current source.
<code>set_sbr_txrx_settings(txrx_settings)</code>	Set SBR+ TX RX antennas settings.
<code>set_source_context(sources[, number_of_modes])</code>	Set the source context.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>thicken_port_sheets(assignment, value[, ...])</code>	Create thickened sheets over a list of input port sheets.
<code>validate_full_design([design, output_dir, ports])</code>	Validate a design based on an expected value and save information to the log file.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".
<code>wave_port(assignment[, reference, ...])</code>	Create a waveport from a sheet (<code>start_object</code>) or taking the closest edges of two objects.

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>components3d</code>	3D components.
<code>composite</code>	HFSS composite mode for the active solution.
<code>configurations</code>	Property to import and export configuration files.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.

continues on next page

Table 2 – continued from previous page

design_list	Design list.
design_name	Design name.
design_properties	Design properties.
design_type	Design type.
desktop_class	Desktop class.
desktop_install_dir	AEDT installation directory.
excitation_objects	Get all excitation.
excitations	Get all excitation names.
excitations_by_type	Design excitations by type.
existing_analysis_setups	Existing analysis setups.
existing_analysis_sweeps	Existing analysis sweeps.
field_setup_names	List of AEDT radiation field names.
field_setups	List of AEDT radiation fields.
get_all_sparameter_list	List of all S parameters for a list of excitations.
hybrid	HFSS hybrid mode for the active solution.
info	Dictionary of the PyAEDT session information.
layoutheditor	Return the Circuit Layout Editor.
library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
materials	Materials in the project.
mesh	Mesh.
modeler	Modeler.
native_components	Native Component dictionary.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.

continues on next page

Table 2 – continued from previous page

output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
working_directory	Path to the working directory.

2.1.3 pyaedt.q3d.Q3d

```
class pyaedt.q3d.Q3d(projectname=None, designname=None, solution_type=None, setup_name=None,
                      specified_version=None, non_graphical=False, new_desktop_session=False,
                      close_on_exit=False, student_version=False, machine='', port=0,
                      aedt_process_id=None)
```

Provides the Q3D app interface.

This class allows you to create an instance of Q3D and link to an existing project or create a new one.

Parameters

projectname

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

setup_name

[**str**, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

specified_version

[**str**, **int**, **float**, optional] Version of AEDT to use. The default is `None`, in which case the active version or latest installed version is used. This parameter is ignored when Script is launched within AEDT. Examples of input values are `232`, `23.2`, ```2023.2```, ```2023.2```.

non_graphical

[**bool**, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[**bool**, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[**bool**, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[**bool**, optional] Whether to open the AEDT student version. The default is `False`. This parameter is ignored when a script is launched within AEDT.

machine

[**str**, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[**int**, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when a new server is created. It works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[**int**, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of Q3D and connect to an existing Q3D design or create a new Q3D design if one does not exist.

```
>>> from pyaedt import Q3d  
>>> app = Q3d()
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,  
specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,  
student_version=False, machine="", port=0, aedt_process_id=None)
```

Methods

<code>activate_variable_optimization(variable_name[, ranges])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name[, ranges])</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name[, ranges])</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ranges])</code>	Activate tuning analysis for a variable and optionally set up ranges.
<code>add_error_message(message_text[, message_type])</code>	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	Add a new toolkit to the current application.
<code>add_info_message(message_text[, message_type])</code>	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>	Create a stackup 3D object.
<code>add_warning_message(message_text[, message_type])</code>	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>	Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>	Analyze a design setup.
<code>archive_project([project_file, ...])</code>	Archive the AEDT project and add a message.
<code>assign_material(obj, mat)</code>	Assign a material to one or more objects.
<code>assign_net(assignment[, net_name, net_type])</code>	Assign a net to a list of objects.
<code>assign_thin_conductor(assignment[, ...])</code>	Assign a thin conductor to a sheet.
<code>assignmaterial_from_sherlock_files(...)</code>	Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>auto_identify_nets()</code>	Identify nets automatically.
<code>autosave_disable()</code>	Disable autosave in AEDT.
<code>autosave_enable()</code>	Enable autosave in AEDT.
<code>change Automatically_use_causal_materials([enable])</code>	Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>	Set Design Settings.
<code>change_material_override([material_override])</code>	Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>	Change a property.
<code>change_validation_settings([...])</code>	Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>	Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>	Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>	Delete a project folder.
<code>cleanup_solution([variations, ...])</code>	Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>	Close AEDT and release it.
<code>close_project([name, save_project])</code>	Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>	Copy the project to another destination.
<code>copy_solid_bodies_from(design[, ...])</code>	Copy a list of objects and user defined models from one design to the active design.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>	Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>	Create a design dataset.

continues on next page

Table 3 – continued from previous page

create_dataset1d_project(dsname, xlist, ylist)	Create a project dataset.
create_dataset3d(dsname, xlist, ylist[, ...])	Create a 3D dataset.
create_new_project(proj_name)	Create a project within AEDT.
create_output_variable(variable, expression)	Create or modify an output variable.
create_setup([name])	Create an analysis setup for Q3D Extractor.
dataset_exists(name[, is_project_dataset])	Check if a dataset exists.
deactivate_variable_optimization(variable_na	Deactivate the optimization analysis for a variable.
deactivate_variable_sensitivity(variable_nam	Deactivate the sensitivity analysis for a variable.
deactivate_variable_statistical(variable_nam	Deactivate the statistical analysis for a variable.
deactivate_variable_tuning(variable_name)	Deactivate the tuning analysis for a variable.
delete_all_nets()	Delete all nets in the design.
delete_design([name, fallback_design])	Delete a design from the current project.
delete_project(project_name)	Delete a project.
delete_separator(separator_name)	Delete a separator from either the active project or a design.
delete_setup(name)	Delete a setup.
delete_unused_variables()	Delete design and project unused variables.
delete_variable(sVarName)	Delete a variable.
design_variation([variation_string])	Generate a string to specify a desired variation.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
edit_setup(name, properties)	Modify a setup.
edit_sources([cg, acrl, dcrl])	Set up the source loaded for Q3D or Q2D in multiple sources simultaneously.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_3d_model([file_name, file_path, ...])	Export the 3D model.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_equivalent_circuit(output_file[, ...])	Export matrix data.
export_matrix_data(file_name[, ...])	Export matrix data.
export_mesh_stats(setup[, variation_string, ...])	Export mesh statistics to a file.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.
export_rl_matrix(matrix_name, file_path[, ...])	Export R/L matrix after solving.
export_variables_to_csv(filename[, ...])	Export design properties, project variables, or both to a CSV file.
flatten_3d_components([component_name, ...])	Flatten one or multiple 3d components in the actual layout.
generate_temp_project_directory(subdir_name	Generate a unique directory string to save a project to.
generate_unique_setup_name([setup_name])	Generate a new setup with an unique name.
get_all_conductors_names()	Retrieve all conductors in the active design.
get_all_dielectrics_names()	Retrieve all dielectrics in the active design.
get_all_sources()	Get all setup sources.
get_components3d_vars(component3dname)	Read the A3DCOMP file and check for variables.
get_dxf_layers(file_path)	Read a DXF file and return all layer names.
get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.

continues on next page

Table 3 – continued from previous page

<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.
<code>get_object_material_properties([assignment, ...])</code>	Retrieve the material properties for a list of objects and return them in a dictionary.
<code>get_oo_name(aedt_object[, object_name])</code>	Return the object-oriented AEDT property names.
<code>get_oo_object(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object.
<code>get_oo_properties(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_oo_property_value(aedt_object, ...)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_output_variable(variable[, solution])</code>	Retrieve the value of the output variable.
<code>get_property_value(objectname, property[, type])</code>	Retrieve a property value.
<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_setup(name)</code>	Get the setup from the current design.
<code>get_setups()</code>	Retrieve setups.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Get a list of traces of specified designs ready to use in plot reports.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>identify_touching_conductors([object_name])</code>	Identify all touching components and group in a dictionary.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>import_dxf(file_path, layers_list[, ...])</code>	Import a DXF file.
<code>import_gds_3d(input_file, mapping_layers[, ...])</code>	Import a GDSII file.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>insert_reduced_matrix(operation_name[, ...])</code>	Insert a new reduced matrix.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>net_sinks(net_name)</code>	Check if a net has sinks and return a list of sink names.
<code>net_sources(net_name)</code>	Check if a net has sources and return a list of source names.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>objects_from_nets(assignment[, materials])</code>	Find the objects that belong to one or more nets.
<code>plot([objects, show, export_path, ...])</code>	Plot the model or a subset of objects.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_material_thresholds([...])</code>	Set material threshold.

continues on next page

Table 3 – continued from previous page

<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_source_context(sources[, number_of_modes])</code>	Set the source context.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>sink([assignment, direction, name, ...])</code>	Generate a sink on a face of an object or a group of faces or face ids.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>source([assignment, direction, name, ...])</code>	Generate a source on a face of an object or a group of faces or face ids.
<code>sources([matrix_index, is_gc_sources])</code>	List of matrix sources.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>components3d</code>	3D components.
<code>configurations</code>	Property to import and export configuration files.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_file</code>	Design file.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>excitation_objects</code>	Get all excitation.
<code>excitations</code>	Get all excitation names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Existing analysis setups.
<code>existing_analysis_sweeps</code>	Existing analysis sweeps.
<code>info</code>	Dictionary of the PyAEDT session information.
<code>layoutheditor</code>	Return the Circuit Layout Editor.
<code>library_list</code>	Library list.
<code>lock_file</code>	Lock file.

continues on next page

Table 4 – continued from previous page

logger	Logger for the design.
materials	Materials in the project.
mesh	Mesh.
modeler	Modeler.
native_components	Native Component dictionary.
nets	Nets in a Q3D project.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.

continues on next page

Table 4 – continued from previous page

solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
working_directory	Path to the working directory.

2.1.4 pyaedt.q3d.Q2d

```
class pyaedt.q3d.Q2d(projectname=None, designname=None, solution_type=None, setup_name=None,  
                      specified_version=None, non_graphical=False, new_desktop_session=False,  
                      close_on_exit=False, student_version=False, machine='', port=0,  
                      aedt_process_id=None)
```

Provides the Q2D app interface.

This class allows you to create an instance of Q2D and link to an existing project or create a new one.

Parameters

projectname

[**str**, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is **None**, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[**str**, optional] Name of the design to select. The default is **None**, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[**str**, optional] Solution type to apply to the design. The default is **None**, in which case the default type is applied.

setup_name

[**str**, optional] Name of the setup to use as the nominal. The default is **None**, in which case the active setup is used or nothing is used.

specified_version

[**str**, **int**, **float**, optional] Version of AEDT to use. The default is **None**, in which case the active version or latest installed version is used. This parameter is ignored when a script is launched within AEDT. Examples of input values are 232, 23.2, ``2023.2``, ``2023.2``.

non_graphical

[**bool**, optional] Whether to launch AEDT in non-graphical mode. The default is **False**, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[**bool**, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the **specified_version** is active on the machine. The default is **False**. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[bool, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[bool, optional] Whether to open the AEDT student version. This parameter is ignored when a script is launched within AEDT.

machine

[str, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[int, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[int, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of Q2D and link to a project named `projectname`. If this project does not exist, create one with this name.

```
>>> from pyaedt import Q2d
>>> app = Q2d(projectname)
```

Create an instance of Q2D and link to a design named `designname` in a project named `projectname`.

```
>>> app = Q2d(projectname, designname)
```

Create an instance of Q2D and open the specified project, which is named `myfile.aedt`.

```
>>> app = Q2d("myfile.aedt")
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,
specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,
student_version=False, machine='', port=0, aedt_process_id=None)
```

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name)</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name)</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.

continues on next page

Table 5 – continued from previous page

<code>add_error_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	mes-	Add a new toolkit to the current application.
<code>add_info_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>		Create a stackup 3D object.
<code>add_warning_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>		Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>		Analyze a design setup.
<code>archive_project([project_file, ...])</code>		Archive the AEDT project and add a message.
<code>assign_huray_finitecond_to_edges(assignment ...)</code>		Assign the Huray surface roughness model to edges.
<code>assign_material(obj, mat)</code>		Assign a material to one or more objects.
<code>assign_single_conductor(assignment[, name, ...])</code>		Assign the conductor type to sheets.
<code>assign_single_signal_line(assignment[, ...])</code>		Assign the conductor type to sheets.
<code>assignmaterial_from_sherlock_files(...)</code>		Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>auto_assign_conductors()</code>		Automatically assign conductors to signal lines.
<code>autosave_disable()</code>		Disable autosave in AEDT.
<code>autosave_enable()</code>		Enable autosave in AEDT.
<code>change Automatically_use_causal_materials</code>		Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>		Set Design Settings.
<code>change_material_override([material_override])</code>		Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>		Change a property.
<code>change_validation_settings([...])</code>		Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>		Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>		Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>		Delete a project folder.
<code>cleanup_solution([variations, ...])</code>		Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>		Close AEDT and release it.
<code>close_project([name, save_project])</code>		Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	de-	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>		Copy the project to another destination.
<code>copy_solid_bodies_from(design[, ...])</code>		Copy a list of objects and user defined models from one design to the active design.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>		Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>		Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>		Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>		Create a 3D dataset.
<code>create_new_project(proj_name)</code>		Create a project within AEDT.
<code>create_output_variable(variable, expression)</code>		Create or modify an output variable.
<code>create_rectangle(origin, sizes[, name, material])</code>		Create a rectangle.
<code>create_setup([name, setup_type])</code>		Create an analysis setup for 2D Extractor.
<code>dataset_exists(name[, is_project_dataset])</code>		Check if a dataset exists.

continues on next page

Table 5 – continued from previous page

deactivate_variable_optimization(variable_name)	Deactivate the optimization analysis for a variable.
deactivate_variable_sensitivity(variable_name)	Deactivate the sensitivity analysis for a variable.
deactivate_variable_statistical(variable_name)	Deactivate the statistical analysis for a variable.
deactivate_variable_tuning(variable_name)	Deactivate the tuning analysis for a variable.
delete_design([name, fallback_design])	Delete a design from the current project.
delete_project(project_name)	Delete a project.
delete_separator(separator_name)	Delete a separator from either the active project or a design.
delete_setup(name)	Delete a setup.
delete_unused_variables()	Delete design and project unused variables.
delete_variable(sVarName)	Delete a variable.
design_variation([variation_string])	Generate a string to specify a desired variation.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
edit_setup(name, properties)	Modify a setup.
edit_sources([cg, acrl, dcrl])	Set up the source loaded for Q3D or Q2D in multiple sources simultaneously.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_3d_model([file_name, file_path, ...])	Export the 3D model.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_equivalent_circuit(output_file[, ...])	Export matrix data.
export_matrix_data(file_name[, ...])	Export matrix data.
export_mesh_stats(setup[, variation_string, ...])	Export mesh statistics to a file.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.
export_rl_matrix(matrix_name, file_path[, ...])	Export R/L matrix after solving.
export_variables_to_csv(filename[, ...])	Export design properties, project variables, or both to a CSV file.
export_w_elements([analyze, export_folder])	Export all W-elements to files.
flatten_3d_components([component_name, ...])	Flatten one or multiple 3d components in the actual layout.
generate_temp_project_directory(subdir_name)	Generate a unique directory string to save a project to.
generate_unique_setup_name([setup_name])	Generate a new setup with an unique name.
get_all_conductors_names()	Retrieve all conductors in the active design.
get_all_dielectrics_names()	Retrieve all dielectrics in the active design.
get_all_sources()	Get all setup sources.
get_components3d_vars(component3dname)	Read the A3DCOMP file and check for variables.
get_dxf_layers(file_path)	Read a DXF file and return all layer names.
get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.
get_monitor_data()	Check and get monitor data of an existing analysis.
get_nominal_variation([with_values])	Retrieve the nominal variation.
get_object_material_properties([assignment, ...])	Retrieve the material properties for a list of objects and return them in a dictionary.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.

continues on next page

Table 5 – continued from previous page

get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_output_variable(variable[, solution])	Retrieve the value of the output variable.
get_property_value(objectname, property[, type])	Retrieve a property value.
get_registry_key_int(key_full_name)	Get the value for the AEDT registry key if one exists.
get_registry_key_string(key_full_name)	Get the value for the AEDT registry key if one exists.
get_setup(name)	Get the setup from the current design.
get_setups()	Retrieve setups.
get_sweeps(name)	Retrieve all sweeps for a setup.
get_traces_for_plot([get_self_terms, ...])	Get a list of traces of specified designs ready to use in plot reports.
hidden_variable(variable_name[, value])	Set the variable to a hidden or unhidden variable.
identify_touching_conductors([object_name])	Identify all touching components and group in a dictionary.
import_dataset1d(filename[, dsname, ...])	Import a 1D dataset.
import_dataset3d(filename[, dsname, ...])	Import a 3D dataset.
import_dxf(file_path, layers_list[, ...])	Import a DXF file.
import_gds_3d(input_file, mapping_layers[, ...])	Import a GDSII file.
insert_design([design_name, solution_type])	Add a design of a specified type.
insert_reduced_matrix(operation_name[, ...])	Insert a new reduced matrix.
list_of_variations([setup_name, sweep_name])	Retrieve a list of active variations for input setup.
load_project(project_file[, design_name, ...])	Open an AEDT project based on a project and optional design.
number_with_units(value[, units])	Convert a number to a string with units.
plot([objects, show, export_path, ...])	Plot the model or a subset of objects.
read_design_data()	Read back the design data as a dictionary.
read_only_variable(variable_name[, value])	Set the variable to a read-only or not read-only variable.
release_desktop([close_projects, close_desktop])	Release AEDT.
remove_all_unused_definitions()	Remove all unused definitions in the project.
rename_design(new_name[, save_after_duplicate])	Rename the active design.
save_project([project_file, overwrite, ...])	Save the project and add a message.
set_active_design(name)	Change the active design to another design.
set_active_dso_config_name([product_name, ...])	Change a specific registry key to a new value.
set_license_type([license_type])	Change the license type between "Pack" and "Pool".
set_oo_property_value(aedt_object, ...)	Change the property value of the object-oriented AEDT object.
set_registry_from_file(registry_file[, ...])	Apply desktop registry settings from an ACT file.
set_registry_key(key_full_name, key_value)	Change a specific registry key to a new value.
set_source_context(sources[, number_of_modes])	Set the source context.
set_temporary_directory(temp_dir_path)	Set temporary directory path.
solve_in_batch([filename, machine, ...])	Analyze a design setup in batch mode.
sources([matrix_index, is_gc_sources])	List of matrix sources.
stop_simulations([clean_stop])	Check if there are simulation running and stops them.
submit_job(clustername[, ...])	Submit a job to be solved on a cluster.
toggle_conductor_type(assignment, new_type)	Change the conductor type.

continues on next page

Table 5 – continued from previous page

<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>components3d</code>	3D components.
<code>configurations</code>	Property to import and export configuration files.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_file</code>	Design file.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>dim</code>	Dimension.
<code>excitation_objects</code>	Get all excitation.
<code>excitations</code>	Get all excitation names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Existing analysis setups.
<code>existing_analysis_sweeps</code>	Existing analysis sweeps.
<code>info</code>	Dictionary of the PyAEDT session information.
<code>layouteditor</code>	Return the Circuit Layout Editor.
<code>library_list</code>	Library list.
<code>lock_file</code>	Lock file.
<code>logger</code>	Logger for the design.
<code>materials</code>	Materials in the project.
<code>mesh</code>	Mesh.
<code>modeler</code>	Modeler.
<code>native_components</code>	Native Component dictionary.
<code>nominal_adaptive</code>	Nominal adaptive sweep.
<code>nominal_sweep</code>	Nominal sweep.
<code>o_component_manager</code>	Component manager object.
<code>o_maxwell_parameters</code>	AEDT Maxwell Parameter Setup Object.
<code>o_model_manager</code>	Model manager object.
<code>o_symbol_manager</code>	Aedt Symbol Manager.
<code>oanalysis</code>	Analysis AEDT Module.
<code>oboundary</code>	Boundary Object.
<code>odefinition_manager</code>	Definition Manager Module.

continues on next page

Table 6 – continued from previous page

odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
working_directory	Path to the working directory.

2.1.5 pyaedt.maxwell.Maxwell2d

```
class pyaedt.maxwell.Maxwell2d(projectname=None, designname=None, solution_type=None,
                               setup_name=None, specified_version=None, non_graphical=False,
                               new_desktop_session=False, close_on_exit=False, student_version=False,
                               machine='', port=0, aedt_process_id=None)
```

Provides the Maxwell 2D app interface.

This class allows you to connect to an existing Maxwell 2D design or create a new Maxwell 2D design if one does not exist.

Parameters

projectname

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

setup_name

[`str`, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

specified_version

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active version or latest installed version is used. This parameter is ignored when a script is launched within AEDT. Examples of input values are `232`, `23.2`, `2023.2`, `2023.2`.

non_graphical

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`. This parameter is ignored when a script is launched within AEDT.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number of which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in

2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of Maxwell 2D and connect to an existing Maxwell 2D design or create a new Maxwell 2D design if one does not exist.

```
>>> from pyaedt import Maxwell2d  
>>> m2d = Maxwell2d()
```

Create an instance of Maxwell 2D and link to a project named `projectname`. If this project does not exist, create one with this name.

```
>>> m2d = Maxwell2d(projectname)
```

Create an instance of Maxwell 2D and link to a design named `designname` in a project named `projectname`.

```
>>> m2d = Maxwell2d(projectname, designname)
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,  
specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,  
student_version=False, machine='', port=0, aedt_process_id=None)
```

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name)</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name)</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.
<code>add_error_message(message_text[, message_type])</code>	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	Add a new toolkit to the current application.
<code>add_info_message(message_text[, message_type])</code>	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>	Create a stackup 3D object.
<code>add_warning_message(message_text[, message_type])</code>	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>add_winding_coils(assignment, coils)</code>	Add coils to the winding.
<code>analyze([name, cores, tasks, gpus, ...])</code>	Solve the active design.

continues on next page

Table 7 – continued from previous page

<code>analyze_from_zero()</code>	Force the next solve to start from time 0 for a given setup.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>	Analyze a design setup.
<code>apply_skew([skew_type, skew_part, ...])</code>	Apply skew to 2D model.
<code>archive_project([project_file, ...])</code>	Archive the AEDT project and add a message.
<code>assign_balloon(assignment[, boundary])</code>	Assign a balloon boundary to a list of edges.
<code>assign_coil(assignment[, conductors_number, ...])</code>	Assign coils to a list of objects or face IDs.
<code>assign_current(assignment[, amplitude, ...])</code>	Assign the source of the current.
<code>assign_current_density(assignment[, ...])</code>	Assign current density to a single or list of entities.
<code>assign_end_connection(assignment[, ...])</code>	Assign an end connection to a list of objects.
<code>assign_force(assignment[, ...])</code>	Assign a force to one or more objects.
<code>assign_master_slave(independent, dependent)</code>	Assign dependent and independent boundary conditions to two edges of the same object.
<code>assign_material(obj, mat)</code>	Assign a material to one or more objects.
<code>assign_matrix(assignment[, matrix_name, ...])</code>	Assign a matrix to the selection.
<code>assign_radiation(assignment[, radiation])</code>	Assign radiation boundary to one or more objects.
<code>assign_rotate_motion(assignment[, ...])</code>	Assign a rotation motion to an object container.
<code>assign_symmetry(assignment[, symmetry_name, ...])</code>	Assign symmetry boundary.
<code>assign_torque(assignment[, ...])</code>	Assign a torque to one or more objects.
<code>assign_translate_motion(assignment[, ...])</code>	Assign a translation motion to an object container.
<code>assign_vector_potential(assignment[, ...])</code>	Assign a vector potential boundary condition to specified edges.
<code>assign_voltage(assignment[, amplitude, name])</code>	Assign a voltage source to a list of faces in Maxwell 3D or a list of objects in Maxwell 2D.
<code>assign_voltage_drop(assignment[, amplitude, ...])</code>	Assign a voltage drop across a list of faces to a specific value.
<code>assign_winding([assignment, winding_type, ...])</code>	Assign a winding to a Maxwell design.
<code>assignmaterial_from_sherlock_files(...)</code>	Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>autosave_disable()</code>	Disable autosave in AEDT.
<code>autosave_enable()</code>	Enable autosave in AEDT.
<code>change_automatically_use_causal_materials()</code>	Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>	Set Design Settings.
<code>change_inductance_computation([...])</code>	Enable the inductance computation for the transient analysis and set the incremental matrix.
<code>change_material_override([material_override])</code>	Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>	Change a property.
<code>change_symmetry_multiplier([value])</code>	Set the design symmetry multiplier to a specified value.
<code>change_validation_settings([...])</code>	Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>	Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>	Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>	Delete a project folder.
<code>cleanup_solution([variations, ...])</code>	Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>	Close AEDT and release it.
<code>close_project([name, save_project])</code>	Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	Copy a design from a project into the active project.

continues on next page

Table 7 – continued from previous page

copy_project(path, dest)	Copy the project to another destination.
copy_solid_bodies_from(design[, ...])	Copy a list of objects and user defined models from one design to the active design.
create_dataset(dsname, xlist, ylist[, ...])	Create a dataset.
create_dataset1d_design(dsname, xlist, ylist)	Create a design dataset.
create_dataset1d_project(dsname, xlist, ylist)	Create a project dataset.
create_dataset3d(dsname, xlist, ylist[, ...])	Create a 3D dataset.
create_new_project(proj_name)	Create a project within AEDT.
create_output_variable(variable, expression)	Create or modify an output variable.
create_setup([name, setup_type])	Create an analysis setup for Maxwell 3D or 2D.
dataset_exists(name[, is_project_dataset])	Check if a dataset exists.
deactivate_variable_optimization(variable_name)	Deactivate the optimization analysis for a variable.
deactivate_variable_sensitivity(variable_name)	Deactivate the sensitivity analysis for a variable.
deactivate_variable_statistical(variable_name)	Deactivate the statistical analysis for a variable.
deactivate_variable_tuning(variable_name)	Deactivate the tuning analysis for a variable.
delete_design([name, fallback_design])	Delete a design from the current project.
delete_project(project_name)	Delete a project.
delete_separator(separator_name)	Delete a separator from either the active project or a design.
delete_setup(name)	Delete a setup.
delete_unused_variables()	Delete design and project unused variables.
delete_variable(sVarName)	Delete a variable.
design_variation([variation_string])	Generate a string to specify a desired variation.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
eddy_effects_on(assignment[, ...])	Assign eddy effects on a list of objects.
edit_external_circuit(netlist_file_path, ...)	Edit the external circuit for the winding and allow editing of the circuit parameters.
edit_setup(name, properties)	Modify a setup.
enable_harmonic_force(assignment[, ...])	Enable the harmonic force calculation for the transient analysis.
enable_harmonic_force_on_layout_component()	Enable the harmonic force calculation for the transient analysis.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_3d_model([file_name, file_path, ...])	Export the 3D model.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_element_based_harmonic_force([...])	Export an element-based harmonic force data to a .csv file.
export_mesh_stats(setup[, variation_string, ...])	Export mesh statistics to a file.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.
export_rl_matrix(matrix_name, file_path[, ...])	Export R/L matrix after solving.
export_variables_to_csv(filename[, ...])	Export design properties, project variables, or both to a CSV file.
flatten_3d_components([component_name, ...])	Flatten one or multiple 3d components in the actual layout.
generate_design_data([line_filter, ...])	Generate a generic set of design data and store it in the extension directory in a <code>design_data.json</code> file.

continues on next page

Table 7 – continued from previous page

generate_temp_project_directory(subdir_name)	Generate a unique directory string to save a project to.
generate_unique_setup_name([setup_name])	Generate a new setup with an unique name.
get_all_conductors_names()	Retrieve all conductors in the active design.
get_all_dielectrics_names()	Retrieve all dielectrics in the active design.
get_all_sources()	Retrieve all setup sources.
get_components3d_vars(component3dname)	Read the A3DCOMP file and check for variables.
get_dxf_layers(file_path)	Read a DXF file and return all layer names.
get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.
get_monitor_data()	Check and get monitor data of an existing analysis.
get_nominal_variation([with_values])	Retrieve the nominal variation.
get_object_material_properties([assignment, ...])	Retrieve the material properties for a list of objects and return them in a dictionary.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.
get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_output_variable(variable[, solution])	Retrieve the value of the output variable.
get_property_value(objectname, property[, type])	Retrieve a property value.
get_registry_key_int(key_full_name)	Get the value for the AEDT registry key if one exists.
get_registry_key_string(key_full_name)	Get the value for the AEDT registry key if one exists.
get_setup(name)	Get the setup from the current design.
get_setups()	Retrieve setups.
get_sweeps(name)	Retrieve all sweeps for a setup.
get_traces_for_plot([get_self_terms, ...])	Retrieve a list of traces of specified designs ready to use in plot reports.
hidden_variable(variable_name[, value])	Set the variable to a hidden or unhidden variable.
identify_touching_conductors([object_name])	Identify all touching components and group in a dictionary.
import_dataset1d(filename[, dsname, ...])	Import a 1D dataset.
import_dataset3d(filename[, dsname, ...])	Import a 3D dataset.
import_dxf(file_path, layers_list[, ...])	Import a DXF file.
import_gds_3d(input_file, mapping_layers[, ...])	Import a GDSII file.
insert_design([design_name, solution_type])	Add a design of a specified type.
list_of_variations([setup_name, sweep_name])	Retrieve a list of active variations for input setup.
load_project(project_file[, design_name, ...])	Open an AEDT project based on a project and optional design.
number_with_units(value[, units])	Convert a number to a string with units.
plot([objects, show, export_path, ...])	Plot the model or a subset of objects.
read_design_data()	Read back the design data as a dictionary.
read_only_variable(variable_name[, value])	Set the variable to a read-only or not read-only variable.
release_desktop([close_projects, close_desktop])	Release AEDT.
remove_all_unused_definitions()	Remove all unused definitions in the project.
rename_design(new_name[, save_after_duplicate])	Rename the active design.
save_project([project_file, overwrite, ...])	Save the project and add a message.
set_active_design(name)	Change the active design to another design.

continues on next page

Table 7 – continued from previous page

<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_core_losses(assignment[, core_loss_on_field])</code>	Whether to enable core losses for a set of objects.
<code>set_initial_angle(motion_setup, angle)</code>	Set the initial angle.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_source_context(sources[, number_of_modes])</code>	Set the source context.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>setup_y_connection([assignment])</code>	Set up the Y connection.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>solve_inside(name[, activate])</code>	Solve inside to generate a solution inside an object.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>components3d</code>	3D components.
<code>configurations</code>	Property to import and export configuration files.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_file</code>	Design file.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>dim</code>	Dimensions.
<code>excitation_objects</code>	Get all excitation.
<code>excitations</code>	Get all excitation names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Existing analysis setups.

continues on next page

Table 8 – continued from previous page

existing_analysis_sweeps	Existing analysis sweeps.
geometry_mode	Geometry mode.
info	Dictionary of the PyAEDT session information.
layoutheditor	Return the Circuit Layout Editor.
library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
materials	Materials in the project.
mesh	Mesh.
model_depth	Model depth.
modeler	Modeler.
native_components	Native Component dictionary.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.

continues on next page

Table 8 – continued from previous page

project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
src_dir	Source directory for Python.
symmetry_multiplier	Symmetry multiplier.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
windings	Windings.
working_directory	Path to the working directory.
xy_plane	Maxwell 2D plane between True and False.

2.1.6 pyaedt.maxwell.Maxwell3d

```
class pyaedt.maxwell.Maxwell3d(projectname=None, designname=None, solution_type=None,
                                setup_name=None, specified_version=None, non_graphical=False,
                                new_desktop_session=False, close_on_exit=False, student_version=False,
                                machine='', port=0, aedt_process_id=None)
```

Provides the Maxwell 3D app interface.

This class allows you to connect to an existing Maxwell 3D design or create a new Maxwell 3D design if one does not exist.

Parameters

projectname

[str, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is None, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[str, optional] Name of the design to select. The default is None, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[str, optional] Solution type to apply to the design. The default is None, in which case the default type is applied.

setup_name

[str, optional] Name of the setup to use as the nominal. The default is None, in which case the active setup is used or nothing is used.

specified_version

[str, int, float, optional] Version of AEDT to use. The default is None, in which case the active version or latest installed version is used. This parameter is ignored when a script is launched within AEDT. Examples of input values are 232, 23.2, ``2023.2`` , ``2023.2`` .

non_graphical

[bool, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[bool, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[bool, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[bool, optional] Whether to open the AEDT student version. The default is `False`. This parameter is ignored when a script is launched within AEDT.

machine

[str, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[int, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when a new server is created. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[int, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of Maxwell 3D and open the specified project, which is named `mymaxwell.aedt`.

```
>>> from pyaedt import Maxwell3d
>>> m3d = Maxwell3d("mymaxwell.aedt")
PyAEDT INFO: Added design ...
```

Create an instance of Maxwell 3D using the 2024 R1 release and open the specified project, which is named `mymaxwell2.aedt`.

```
>>> m3d = Maxwell3d(specified_version="2024.1", projectname="mymaxwell2.aedt")
PyAEDT INFO: Added design ...
```

`__init__(projectname=None, designname=None, solution_type=None, setup_name=None, specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False, student_version=False, machine='', port=0, aedt_process_id=None)`

Initialize the Maxwell class.

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name[, ...])</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name[, ...])</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.
<code>add_error_message(message_text[, message_type])</code>	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	Add a new toolkit to the current application.
<code>add_info_message(message_text[, message_type])</code>	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>	Create a stackup 3D object.
<code>add_warning_message(message_text[, message_type])</code>	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>add_winding_coils(assignment, coils)</code>	Add coils to the winding.
<code>analyze([name, cores, tasks, gpus, ...])</code>	Solve the active design.
<code>analyze_from_zero()</code>	Force the next solve to start from time 0 for a given setup.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>	Analyze a design setup.
<code>apply_skew([skew_type, skew_part, ...])</code>	Apply skew to 2D model.
<code>archive_project([project_file, ...])</code>	Archive the AEDT project and add a message.
<code>assign_coil(assignment[, conductors_number, ...])</code>	Assign coils to a list of objects or face IDs.
<code>assign_current(assignment[, amplitude, ...])</code>	Assign the source of the current.
<code>assign_current_density(assignment[, ...])</code>	Assign current density to a single or list of entities.
<code>assign_current_density_terminal(assignment)</code>	Assign current density terminal to a single or list of entities for an Eddy Current or Magnetostatic solver.
<code>assign_flux_tangential(assignment[, flux_name])</code>	Assign a flux tangential boundary for a transient A-Phi solver.
<code>assign_force(assignment[, ...])</code>	Assign a force to one or more objects.
<code>assign_impedance(assignment[, ...])</code>	Create an impedance boundary condition for Transient or Eddy Current solvers.
<code>assign_insulating(assignment[, insulation])</code>	Create an insulating boundary condition.
<code>assign_layout_force(net_layers, component_name)</code>	Assign the layout force to a component in a Transient A-Phi solver.
<code>assign_master_slave(independent, dependent, ...)</code>	Assign dependent and independent boundary conditions to two faces of the same object.
<code>assign_material(obj, mat)</code>	Assign a material to one or more objects.
<code>assign_matrix(assignment[, matrix_name, ...])</code>	Assign a matrix to the selection.
<code>assign_radiation(assignment[, radiation])</code>	Assign radiation boundary to one or more objects.
<code>assign_rotate_motion(assignment[, ...])</code>	Assign a rotation motion to an object container.
<code>assign_symmetry(assignment[, symmetry_name, ...])</code>	Assign symmetry boundary.
<code>assign_tangential_h_field(assignment[, ...])</code>	Assign a tangential H field boundary to a list of faces.
<code>assign_torque(assignment[, ...])</code>	Assign a torque to one or more objects.

continues on next page

Table 9 – continued from previous page

<code>assign_translate_motion(assignment[, ...])</code>	Assign a translation motion to an object container.
<code>assign_voltage(assignment[, amplitude, name])</code>	Assign a voltage source to a list of faces in Maxwell 3D or a list of objects in Maxwell 2D.
<code>assign_voltage_drop(assignment[, amplitude, ...])</code>	Assign a voltage drop across a list of faces to a specific value.
<code>assign_winding([assignment, winding_type, ...])</code>	Assign a winding to a Maxwell design.
<code>assign_zero_tangential_h_field(assignment[, ...])</code>	Assign a zero tangential H field boundary to a list of faces.
<code>assignmaterial_from_sherlock_files(...)</code>	Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>autosave_disable()</code>	Disable autosave in AEDT.
<code>autosave_enable()</code>	Enable autosave in AEDT.
<code>change Automatically_use_causal_materials()</code>	Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>	Set Design Settings.
<code>change_inductance_computation([...])</code>	Enable the inductance computation for the transient analysis and set the incremental matrix.
<code>change_material_override([material_override])</code>	Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>	Change a property.
<code>change_symmetry_multiplier([value])</code>	Set the design symmetry multiplier to a specified value.
<code>change_validation_settings([...])</code>	Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>	Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>	Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>	Delete a project folder.
<code>cleanup_solution([variations, ...])</code>	Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>	Close AEDT and release it.
<code>close_project([name, save_project])</code>	Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>	Copy the project to another destination.
<code>copy_solid_bodies_from(design[, ...])</code>	Copy a list of objects and user defined models from one design to the active design.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>	Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>	Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>	Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>	Create a 3D dataset.
<code>create_new_project(proj_name)</code>	Create a project within AEDT.
<code>create_output_variable(variable, expression)</code>	Create or modify an output variable.
<code>create_setup([name, setup_type])</code>	Create an analysis setup for Maxwell 3D or 2D.
<code>dataset_exists(name[, is_project_dataset])</code>	Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>	Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>	Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>	Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>	Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>	Delete a design from the current project.
<code>delete_project(project_name)</code>	Delete a project.
<code>delete_separator(separator_name)</code>	Delete a separator from either the active project or a design.
<code>delete_setup(name)</code>	Delete a setup.
<code>delete_unused_variables()</code>	Delete design and project unused variables.

continues on next page

Table 9 – continued from previous page

<code>delete_variable(sVarName)</code>	Delete a variable.
<code>design_variation([variation_string])</code>	Generate a string to specify a desired variation.
<code>duplicate_design(label[, save_after_duplicate])</code>	Copy a design to a new name.
<code>eddy_effects_on(assignment[, ...])</code>	Assign eddy effects on a list of objects.
<code>edit_external_circuit(netlist_file_path, ...)</code>	Edit the external circuit for the winding and allow editing of the circuit parameters.
<code>edit_setup(name, properties)</code>	Modify a setup.
<code>enable_harmonic_force(assignment[, ...])</code>	Enable the harmonic force calculation for the transient analysis.
<code>enable_harmonic_force_on_layout_component()</code>	Enable the harmonic force calculation for the transient analysis.
<code>evaluate_expression(expression_string)</code>	Evaluate a valid string expression and return the numerical value in SI units.
<code>export_3d_model([file_name, file_path, ...])</code>	Export the 3D model.
<code>export_convergence(setup_name[, ...])</code>	Export a solution convergence to a file.
<code>export_design_preview_to_jpg(filename)</code>	Export design preview image to a JPG file.
<code>export_element_based_harmonic_force([...])</code>	Export an element-based harmonic force data to a .csv file.
<code>export_mesh_stats(setup[, variation_string, ...])</code>	Export mesh statistics to a file.
<code>export_parametric_results(sweep, filename[, ...])</code>	Export a list of all parametric variations solved for a sweep to a CSV file.
<code>export_profile(setup_name[, ...])</code>	Export a solution profile to a PROF file.
<code>export_results([analyze, export_folder, ...])</code>	Export all available reports to a file, including profile, and convergence and sNp when applicable.
<code>export_rl_matrix(matrix_name, file_path[, ...])</code>	Export R/L matrix after solving.
<code>export_variables_to_csv(filename[, ...])</code>	Export design properties, project variables, or both to a CSV file.
<code>flatten_3d_components([component_name, ...])</code>	Flatten one or multiple 3d components in the actual layout.
<code>generate_temp_project_directory(subdir_name)</code>	Generate a unique directory string to save a project to.
<code>generate_unique_setup_name([setup_name])</code>	Generate a new setup with an unique name.
<code>get_all_conductors_names()</code>	Retrieve all conductors in the active design.
<code>get_all_dielectrics_names()</code>	Retrieve all dielectrics in the active design.
<code>get_all_sources()</code>	Retrieve all setup sources.
<code>get_components3d_vars(component3dname)</code>	Read the A3DCOMP file and check for variables.
<code>get_conduction_paths()</code>	Get a dictionary of all conduction paths with relative objects.
<code>get_dxf_layers(file_path)</code>	Read a DXF file and return all layer names.
<code>get_evaluated_value(variable_name[, units])</code>	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
<code>get_module(module_name)</code>	Aedt Module object.
<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.
<code>get_object_material_properties([assignment, ...])</code>	Retrieve the material properties for a list of objects and return them in a dictionary.
<code>get_oo_name(aedt_object[, object_name])</code>	Return the object-oriented AEDT property names.
<code>get_oo_object(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object.
<code>get_oo_properties(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_oo_property_value(aedt_object, ...)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_output_variable(variable[, solution])</code>	Retrieve the value of the output variable.

continues on next page

Table 9 – continued from previous page

<code>get_property_value(objectname, property[, type])</code>	Retrieve a property value.
<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_setup(name)</code>	Get the setup from the current design.
<code>get_setups()</code>	Retrieve setups.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Retrieve a list of traces of specified designs ready to use in plot reports.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>identify_touching_conductors([object_name])</code>	Identify all touching components and group in a dictionary.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>import_dxf(file_path, layers_list[, ...])</code>	Import a DXF file.
<code>import_gds_3d(input_file, mapping_layers[, ...])</code>	Import a GDSII file.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>plot([objects, show, export_path, ...])</code>	Plot the model or a subset of objects.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_core_losses(assignment[, core_loss_on_field])</code>	Whether to enable core losses for a set of objects.
<code>set_initial_angle(motion_setup, angle)</code>	Set the initial angle.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_source_context(sources[, number_of_modes])</code>	Set the source context.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>setup_y_connection([assignment])</code>	Set up the Y connection.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>solve_inside(name[, activate])</code>	Solve inside to generate a solution inside an object.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.

continues on next page

Table 9 – continued from previous page

<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".
--	---

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>components3d</code>	3D components.
<code>configurations</code>	Property to import and export configuration files.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_file</code>	Design file.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>dim</code>	Dimensions.
<code>excitation_objects</code>	Get all excitation.
<code>excitations</code>	Get all excitation names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Existing analysis setups.
<code>existing_analysis_sweeps</code>	Existing analysis sweeps.
<code>info</code>	Dictionary of the PyAEDT session information.
<code>layoutheditor</code>	Return the Circuit Layout Editor.
<code>library_list</code>	Library list.
<code>lock_file</code>	Lock file.
<code>logger</code>	Logger for the design.
<code>materials</code>	Materials in the project.
<code>mesh</code>	Mesh.
<code>modeler</code>	Modeler.
<code>native_components</code>	Native Component dictionary.
<code>nominal_adaptive</code>	Nominal adaptive sweep.
<code>nominal_sweep</code>	Nominal sweep.
<code>o_component_manager</code>	Component manager object.
<code>o_maxwell_parameters</code>	AEDT Maxwell Parameter Setup Object.
<code>o_model_manager</code>	Model manager object.
<code>o_symbol_manager</code>	Aedt Symbol Manager.
<code>oanalysis</code>	Analysis AEDT Module.
<code>oboundary</code>	Boundary Object.
<code>odefinition_manager</code>	Definition Manager Module.
<code>odesign</code>	Design.

continues on next page

Table 10 – continued from previous page

odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
src_dir	Source directory for Python.
symmetry_multiplier	Symmetry multiplier.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
windings	Windings.
working_directory	Path to the working directory.

2.1.7 pyaedt.icepak.Icepak

```
class pyaedt.icepak.Icepak(projectname=None, designname=None, solution_type=None, setup_name=None,
                           specified_version=None, non_graphical=False, new_desktop_session=False,
                           close_on_exit=False, student_version=False, machine="", port=0,
                           aedt_process_id=None)
```

Provides the Icepak application interface.

This class allows you to connect to an existing Icepak design or create a new Icepak design if one does not exist.

Parameters

projectname

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

setup_name

[`str`, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

specified_version

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active version or latest installed version is used. This parameter is ignored when Script is launched within AEDT. Examples of input values are `232`, `23.2`, ```2023.2```, ```2023.2```.

non-graphical

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`. This parameter is ignored when a script is launched within AEDT.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number of which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[int, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is None. This parameter is only used when new_desktop_session = False.

Examples

Create an instance of Icepak and connect to an existing Icepak design or create a new Icepak design if one does not exist.

```
>>> from pyaedt import Icepak
>>> icepak = Icepak()
PyAEDT INFO: No project is defined. Project ...
PyAEDT INFO: Active design is set to ...
```

Create an instance of Icepak and link to a project named `IcepakProject`. If this project does not exist, create one with this name.

```
>>> icepak = Icepak("IcepakProject")
PyAEDT INFO: Project ...
PyAEDT INFO: Added design ...
```

Create an instance of Icepak and link to a design named `IcepakDesign1` in a project named `IcepakProject`.

```
>>> icepak = Icepak("IcepakProject", "IcepakDesign1")
PyAEDT INFO: Added design 'IcepakDesign1' of type Icepak.
```

Create an instance of Icepak and open the specified project, which is `myipk.aedt`.

```
>>> icepak = Icepak("myipk.aedt")
PyAEDT INFO: Project myipk has been created.
PyAEDT INFO: No design is present. Inserting a new design.
PyAEDT INFO: Added design ...
```

Create an instance of Icepak using the 2023 R2 release and open the specified project, which is `myipk2.aedt`.

```
>>> icepak = Icepak(specified_version=2023.2, projectname="myipk2.aedt")
PyAEDT INFO: Project...
PyAEDT INFO: No design is present. Inserting a new design.
PyAEDT INFO: Added design...
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,
        specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,
        student_version=False, machine="", port=0, aedt_process_id=None)
```

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name[, ...])</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name[, ...])</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.
<code>add_error_message(message_text[, message_type])</code>	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	Add a new toolkit to the current application.
<code>add_info_message(message_text[, message_type])</code>	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>	Create a stackup 3D object.
<code>add_warning_message(message_text[, message_type])</code>	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>	Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>	Analyze a design setup.
<code>archive_project([project_file, ...])</code>	Archive the AEDT project and add a message.
<code>assign_2way_coupling([setup, ...])</code>	Assign two-way coupling to a setup.
<code>assign_adiabatic_plate(assignment[, ...])</code>	Assign adiabatic plate boundary condition.
<code>assign_block_from_sherlock_file(csv_name)</code>	Assign block power to components based on a CSV file from Sherlock.
<code>assign_blower_type1(faces, inlet_face, ...)</code>	Assign blower type 1.
<code>assign_blower_type2(faces, inlet_face, ...)</code>	Assign blower type 2.
<code>assign_conducting_plate(obj_plate[, ...])</code>	Assign thermal boundary conditions to a conducting plate.
<code>assign_conducting_plate_with_conductance(obj[, conductance])</code>	Assign thermal boundary conditions with conductance specification to a conducting plate.
<code>assign_conducting_plate_with_impedance(obj[, impedance])</code>	Assign thermal boundary conditions with thermal impedance specification to a conducting plate.
<code>assign_conducting_plate_with_resistance(obj[, resistance])</code>	Assign thermal boundary conditions with thermal resistance specification to a conducting plate.
<code>assign_conducting_plate_with_thickness(obj[, thickness])</code>	Assign thermal boundary conditions with thickness specification to a conducting plate.
<code>assign_device_resistance(objects[, ...])</code>	Assign resistance boundary condition using the device/approach model.
<code>assign_em_losses([design, setup, sweep, ...])</code>	Map EM losses to an Icepak design.
<code>assign_free_opening(assignment[, ...])</code>	Assign free opening boundary condition.
<code>assign_grille(air_faces[, free_loss_coeff, ...])</code>	Assign grille to a face or list of faces.
<code>assign_hollow_block(object_name, ..., [...])</code>	Assign block boundary for hollow objects.
<code>assign_loss_curve_resistance(objects[, ...])</code>	Assign resistance boundary condition prescribing a loss curve.
<code>assign_mass_flow_free_opening(assignment[, ...])</code>	Assign free opening boundary condition.
<code>assign_material(obj, mat)</code>	Assign a material to one or more objects.
<code>assign_openings(air_faces)</code>	Assign openings to a list of faces.

continues on next page

Table 11 – continued from previous page

<code>assign_point_monitor(point_position[, ...])</code>	Create and assign a point monitor.
<code>assign_point_monitor_in_object(name[, ...])</code>	Assign a point monitor in the centroid of a specific object.
<code>assign_power_law_resistance(objects[, ...])</code>	Assign resistance boundary condition prescribing a power law.
<code>assign_pressure_free_opening(assignment[, ...])</code>	Assign free opening boundary condition.
<code>assign_priority_on_intersections([...])</code>	Validate an Icepak design.
<code>assign_recirculation_opening(face_list, ...)</code>	Assign recirculation faces.
<code>assign_resistance(objects[, boundary_name, ...])</code>	Assign resistance boundary condition.
<code>assign_solid_block(object_name, power_assignment)</code>	Assign block boundary for solid objects.
<code>assign_source(assignment, thermal_condition, ...)</code>	Create a source power for a face.
<code>assign_stationary_wall(geometry, ..., ...)</code>	Assign surface wall boundary condition.
<code>assign_stationary_wall_with_heat_flux(geom</code>	Assign a surface wall boundary condition with specified heat flux.
<code>assign_stationary_wall_with_htc(geometry[, ...])</code>	Assign a surface wall boundary condition with a given heat transfer coefficient.
<code>assign_stationary_wall_with_temperature(ge</code>	Assign a surface wall boundary condition with specified temperature.
<code>assign_surface_material(obj, mat)</code>	Assign a surface material to one or more objects.
<code>assign_surface_monitor(face_name[, ...])</code>	Assign a surface monitor.
<code>assign_symmetry_wall(geometry[, bound- ary_name])</code>	Assign symmetry wall boundary condition.
<code>assign_velocity_free_opening(assignment[, ...])</code>	Assign free opening boundary condition.
<code>assignmaterial_from_sherlock_files(...)</code>	Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>autosave_disable()</code>	Disable autosave in AEDT.
<code>autosave_enable()</code>	Enable autosave in AEDT.
<code>change_automatically_use_causal_materials</code>	Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>	Set Design Settings.
<code>change_material_override([material_override])</code>	Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>	Change a property.
<code>change_validation_settings([...])</code>	Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>	Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>	Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>	Delete a project folder.
<code>cleanup_solution([variations, ...])</code>	Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>	Close AEDT and release it.
<code>close_project([name, save_project])</code>	Close an AEDT project.
<code>copyGroupFrom(group_name, source_design[, ...])</code>	Copy a group from another design.
<code>copy_design_from(project_fullname, de- sign_name)</code>	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>	Copy the project to another destination.
<code>copy_solid_bodies_from(design[, ...])</code>	Copy a list of objects and user defined models from one design to the active design.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>	Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>	Create a design dataset.

continues on next page

Table 11 – continued from previous page

<code>create_dataset1d_project(dsname, xlist, ylist)</code>	Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>	Create a 3D dataset.
<code>create_dataset_transient_assignment(ds_name)</code>	Create a transient assignment from a dataset.
<code>create_exponential_transient_assignment(...)</code>	Create an object to assign the exponential transient condition to.
<code>create_fan([name, is_2d, shape, ...])</code>	Create a fan component in Icepak that is linked to an HFSS 3D Layout object.
<code>create_ipk_3dcomponent_pcb(compName, ...[, ...])</code>	Create a PCB component in Icepak that is linked to an HFSS 3D Layout object.
<code>create_linear_transient_assignment(...)</code>	Create an object to assign the linear transient condition to.
<code>create_network_blocks(input_list, gravity_dir)</code>	Create network blocks from CSV files.
<code>create_network_object([name, props, create])</code>	Create a thermal network.
<code>create_new_project(proj_name)</code>	Create a project within AEDT.
<code>create_output_variable(variable, expression)</code>	Create or modify an output variable.
<code>create_parametric_fin_heat_sink([hs_height, ...])</code>	Create a parametric heat sink.
<code>create_parametric_heatsink_on_face(top_face)</code>	Create a parametric heat sink.
<code>create_pcb_from_3dlayout(component_name, ...)</code>	Create a PCB component in Icepak that is linked to an HFSS 3DLayout object linking only to the geometry file.
<code>create_powerlaw_transient_assignment(...)</code>	Create an object to assign the power law transient condition to.
<code>create_resistor_network_from_matrix(...[, ...])</code>	Create a thermal network.
<code>create_setup([name, setup_type])</code>	Create an analysis setup for Icepak.
<code>create_sinusoidal_transient_assignment(...)</code>	Create an object to assign the sinusoidal transient condition to.
<code>create_source_blocks_from_list(list_powers)</code>	Assign to a box in Icepak the sources that come from the CSV file.
<code>create_square_wave_transient_assignment(...)</code>	Create an object to assign the square wave transient condition to.
<code>create_temp_dep_assignment(ds_name[, scale])</code>	Create a temperature-dependent assignment from a dataset.
<code>create_two_resistor_network_block(...)</code>	Function to create 2-Resistor network object.
<code>dataset_exists(name[, is_project_dataset])</code>	Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>	Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>	Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>	Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>	Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>	Delete a design from the current project.
<code>delete_em_losses(bound_name)</code>	Delete the EM losses boundary.
<code>delete_pcb_component(comp_name)</code>	Delete a PCB component.
<code>delete_project(project_name)</code>	Delete a project.
<code>delete_separator(separator_name)</code>	Delete a separator from either the active project or a design.
<code>delete_setup(name)</code>	Delete a setup.
<code>delete_unused_variables()</code>	Delete design and project unused variables.
<code>delete_variable(sVarName)</code>	Delete a variable.
<code>design_variation([variation_string])</code>	Generate a string to specify a desired variation.
<code>duplicate_design(label[, save_after_duplicate])</code>	Copy a design to a new name.
<code>edit_design_settings([gravity_dir, ...])</code>	Update the main settings of the design.

continues on next page

Table 11 – continued from previous page

<code>edit_setup(name, properties)</code>	Modify a setup.
<code>eval_surface_quantity_from_field_summary()</code>	Export the field surface output.
<code>eval_volume_quantity_from_field_summary(...)</code>	Export the field volume output.
<code>evaluate_expression(expression_string)</code>	Evaluate a valid string expression and return the numerical value in SI units.
<code>export_3d_model([file_name, file_path, ...])</code>	Export the 3D model.
<code>export_convergence(setup_name[, ...])</code>	Export a solution convergence to a file.
<code>export_design_preview_to_jpg(filename)</code>	Export design preview image to a JPG file.
<code>export_mesh_stats(setup[, variation_string, ...])</code>	Export mesh statistics to a file.
<code>export_parametric_results(sweep, filename[, ...])</code>	Export a list of all parametric variations solved for a sweep to a CSV file.
<code>export_profile(setup_name[, ...])</code>	Export a solution profile to a PROF file.
<code>export_results([analyze, export_folder, ...])</code>	Export all available reports to a file, including profile, and convergence and sNp when applicable.
<code>export_rl_matrix(matrix_name, file_path[, ...])</code>	Export R/L matrix after solving.
<code>export_summary([output_dir, solution_name, ...])</code>	Export a fields summary of all objects.
<code>export_variables_to_csv(filename[, ...])</code>	Export design properties, project variables, or both to a CSV file.
<code>find_top(gravityDir)</code>	Find the top location of the layout given a gravity.
<code>flatten_3d_components([component_name, ...])</code>	Flatten one or multiple 3d components in the actual layout.
<code>generate_fluent_mesh([object_lists, ...])</code>	Generate a Fluent mesh for a list of selected objects and assign the mesh automatically to the objects.
<code>generate_temp_project_directory(subdir_name)</code>	Generate a unique directory string to save a project to.
<code>generate_unique_setup_name([setup_name])</code>	Generate a new setup with an unique name.
<code>get_all_conductors_names()</code>	Retrieve all conductors in the active design.
<code>get_all_dielectrics_names()</code>	Retrieve all dielectrics in the active design.
<code>get_all_sources()</code>	Retrieve all setup sources.
<code>get_components3d_vars(component3dname)</code>	Read the A3DCOMP file and check for variables.
<code>get_dxf_layers(file_path)</code>	Read a DXF file and return all layer names.
<code>get_evaluated_value(variable_name[, units])</code>	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
<code>get_fans_operating_point([export_file, ...])</code>	Get operating point of the fans in the design.
<code>get_gas_objects()</code>	Get gas objects.
<code>get_link_data(links_data, **kwargs)</code>	Get a list of linked data.
<code>get_liquid_objects()</code>	Get liquid material objects.
<code>get_module(module_name)</code>	Aedt Module object.
<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.
<code>get_object_material_properties([assignment, ...])</code>	Retrieve the material properties for a list of objects and return them in a dictionary.
<code>get_oo_name(aedt_object[, object_name])</code>	Return the object-oriented AEDT property names.
<code>get_oo_object(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object.
<code>get_oo_properties(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_oo_property_value(aedt_object, ...)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_output_variable(variable[, solution])</code>	Retrieve the value of the output variable.
<code>get_property_value(objectname, property[, type])</code>	Retrieve a property value.
<code>get_radiation_settings(radiation)</code>	Get radiation settings.
<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.

continues on next page

Table 11 – continued from previous page

<code>get_setup(name)</code>	Get the setup from the current design.
<code>get_setups()</code>	Retrieve setups.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Retrieve a list of traces of specified designs ready to use in plot reports.
<code>globalMeshSettings(mesh_type[, ...])</code>	Create a custom mesh tailored on a PCB design.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>identify_touching_conductors([object_name])</code>	Identify all touching components and group in a dictionary.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>import_dxf(file_path, layers_list[, ...])</code>	Import a DXF file.
<code>import_gds_3d(input_file, mapping_layers[, ...])</code>	Import a GDSII file.
<code>import_idf(board_path[, library_path, ...])</code>	Import an IDF file into an Icepak design.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>plot([objects, show, export_path, ...])</code>	Plot the model or a subset of objects.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_source_context(sources[, number_of_modes])</code>	Set the source context.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

Position	Position of the object.
SimulationSetupTypes	Simulation setup types.
SolutionTypes	Solution types.
active_setup	Get or Set the name of the active setup.
aedt_version_id	AEDT version.
are_there_simulations_running	Check if there are simulation running.
available_variations	Available variation object.
boundaries	Design boundaries and excitations.
boundaries_by_type	Design boundaries by type.
components3d	3D components.
configurations	Property to import and export configuration files.
default_solution_type	Default solution type.
design_datasets	Dictionary of Design Datasets.
design_list	Design list.
design_name	Design name.
design_properties	Design properties.
design_type	Design type.
desktop_class	Desktop class.
desktop_install_dir	AEDT installation directory.
excitation_objects	Get all excitation.
excitations	Get all excitation names.
excitations_by_type	Design excitations by type.
existing_analysis_setups	Existing analysis setups.
existing_analysis_sweeps	Existing analysis setups.
info	Dictionary of the PyAEDT session information.
layoutheditor	Return the Circuit Layout Editor.
library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
materials	Materials in the project.
mesh	Mesh.
modeler	Modeler.
monitor	Property to handle monitor objects.
native_components	Native Component dictionary.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.

continues on next page

Table 12 – continued from previous page

<code>omatrix</code>	Matrix Object.
<code>omeshmodule</code>	Icepak Mesh Module.
<code>omodelsetup</code>	AEDT Model Setup Object.
<code>omonitor</code>	AEDT Monitor Object.
<code>ooptimetrics</code>	AEDT Optimetrics Module.
<code>ooutput_variable</code>	AEDT Output Variable Module.
<code>opadstackmanager</code>	AEDT oPadstackManager.
<code>oproject</code>	Project property.
<code>optimizations</code>	Optimizations in the project.
<code>oradfield</code>	AEDT Radiation Field Object.
<code>oreportsetup</code>	Report setup.
<code>osolution</code>	Solution Module.
<code>output_variables</code>	List of output variables.
<code>parametrics</code>	Setups in the project.
<code>personallib</code>	PersonalLib directory.
<code>ports</code>	Design excitations.
<code>post</code>	PostProcessor.
<code>problem_type</code>	Problem type of the Icepak design.
<code>project_datasets</code>	Dictionary of project datasets.
<code>project_file</code>	Project name and path.
<code>project_list</code>	Project list.
<code>project_name</code>	Project name.
<code>project_path</code>	Project path.
<code>project_properties</code>	Project properties.
<code>project_time_stamp</code>	Return Project time stamp.
<code>project_timestamp_changed</code>	Return a bool if time stamp changed or not.
<code>pyaedt_dir</code>	Pyaedt directory.
<code>results_directory</code>	Results directory.
<code>settings</code>	Settings of the current Python/Pyaedt session.
<code>setup_names</code>	Setup names.
<code>setups</code>	Setups in the project.
<code>solution_type</code>	Solution type.
<code>src_dir</code>	Source directory for Python.
<code>syslib</code>	SysLib directory.
<code>temp_directory</code>	Path to the temporary directory.
<code>toolkit_directory</code>	Path to the toolkit directory.
<code>userlib</code>	UserLib directory.
<code>valid_design</code>	Valid design.
<code>variable_manager</code>	Variable manager for creating and managing project design and postprocessing variables.
<code>working_directory</code>	Path to the working directory.

2.1.8 pyaedt.hfss3dlayout.Hfss3dLayout

```
class pyaedt.hfss3dlayout.Hfss3dLayout(projectname=None, designname=None, solution_type=None,
                                         setup_name=None, specified_version=None,
                                         non_graphical=False, new_desktop_session=False,
                                         close_on_exit=False, student_version=False, machine="",
                                         port=0, aedt_process_id=None, ic_mode=False)
```

Provides the HFSS 3D Layout application interface.

This class inherits all objects that belong to HFSS 3D Layout, including EDB API queries.

Parameters

projectname

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open or the path to the aedb folder or `edb.def` file. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

setup_name

[`str`, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

specified_version

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active version or latest installed version is used. Examples of input values are `232`, `23.2`, ```2023.2```.

non_graphical

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in

2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

ic_mode

[`bool`, optional] Whether to set the design to IC mode or not. The default is `False`.

Examples

Create an `Hfss3dLayout` object and connect to an existing HFSS design or create a new HFSS design if one does not exist.

```
>>> from pyaedt import Hfss3dLayout  
>>> aedtapp = Hfss3dLayout()
```

Create an `Hfss3dLayout` object and link to a project named `projectname`. If this project does not exist, create one with this name.

```
>>> aedtapp = Hfss3dLayout(projectname)
```

Create an `Hfss3dLayout` object and link to a design named `designname` in a project named `projectname`.

```
>>> aedtapp = Hfss3dLayout(projectname, designname)
```

Create an `Hfss3dLayout` object and open the specified project.

```
>>> aedtapp = Hfss3dLayout("myfile.aedt")
```

Create an AEDT 2023 R1 object and then create a `Hfss3dLayout` object and open the specified project.

```
>>> aedtapp = Hfss3dLayout(specified_version="2023.1", projectname="myfile.aedt")
```

Create an instance of `Hfss3dLayout` from an Edb

```
>>> import pyaedt  
>>> edb_path = "/path/to/edbfile.aedb"  
>>> edb = pyaedt.Edb(edb_path, edbversion=231)  
>>> edb.stackup.import_stackup("stackup.xml") # Import stackup. Manipulate edb, ...  
>>> edb.save_edb()  
>>> edb.close_edb()  
>>> aedtapp = pyaedt.Hfss3dLayout(specified_version=231, projectname=edb_path)
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,  
specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,  
student_version=False, machine="", port=0, aedt_process_id=None, ic_mode=False)
```

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name[, ...])</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name[, ...])</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.
<code>add_error_message(message_text[, message_type])</code>	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	Add a new toolkit to the current application.
<code>add_info_message(message_text[, message_type])</code>	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_warning_message(message_text[, message_type])</code>	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>	Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>	Analyze a design setup.
<code>archive_project([project_file, ...])</code>	Archive the AEDT project and add a message.
<code>autosave_disable()</code>	Disable autosave in AEDT.
<code>autosave_enable()</code>	Enable autosave in AEDT.
<code>change_automatically_use_causal_materials()</code>	Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>	Set HFSS 3D Layout Design Settings.
<code>change_material_override([material_override])</code>	Enable or disable the material override in the project.
<code>change_options([color_by_net])</code>	Change options for an existing layout.
<code>change_property(aedt_object, tab_name, ...)</code>	Change a property.
<code>change_validation_settings([...])</code>	Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>	Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>	Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>	Delete a project folder.
<code>close_desktop()</code>	Close AEDT and release it.
<code>close_project([name, save_project])</code>	Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>	Copy the project to another destination.
<code>create_coax_port(via[, radial_extent, ...])</code>	Create a coax port.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>	Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>	Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>	Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>	Create a 3D dataset.
<code>create_differential_port(via_signal, ...[, ...])</code>	Create a differential port.
<code>create_edge_port(assignment, edge_number[, ...])</code>	Create an edge port.
<code>create_linear_count_sweep(setup, unit, ...)</code>	Create a sweep with the specified number of points.
<code>create_linear_step_sweep(setup, unit, ...[, ...])</code>	Create a sweep with the specified frequency step.
<code>create_new_project(proj_name)</code>	Create a project within AEDT.

continues on next page

Table 13 – continued from previous page

<code>create_output_variable(variable, expression)</code>	Create or modify an output variable.
<code>create_pec_on_component_by_nets(component, nets)</code>	Create a PEC connection on a component for a list of nets.
<code>create_pin_port(name[, x, y, rotation, ...])</code>	Create a pin port.
<code>create_ports_on_component_by_nets(component[, ...])</code>	Create the ports on a component for a list of nets.
<code>create_scattering([plot, sweep_name, ...])</code>	Create a scattering report.
<code>create_setup([name, setup_type])</code>	Create a setup.
<code>create_single_point_sweep(setup, unit, freq)</code>	Create a sweep with a single frequency point.
<code>create_wave_port(assignment, edge_number[, ...])</code>	Create a single-ended wave port.
<code>create_wave_port_from_two_conductors([...])</code>	Create a wave port.
<code>dataset_exists(name[, is_project_dataset])</code>	Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>	Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>	Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>	Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>	Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>	Delete a design from the current project.
<code>delete_port(name)</code>	Delete a port.
<code>delete_project(project_name)</code>	Delete a project.
<code>delete_separator(separator_name)</code>	Delete a separator from either the active project or a design.
<code>delete_setup(name)</code>	Delete a setup.
<code>delete_unused_variables()</code>	Delete design and project unused variables.
<code>delete_variable(sVarName)</code>	Delete a variable.
<code>design_variation([variation_string])</code>	Generate a string to specify a desired variation.
<code>dissolve_component(component)</code>	Dissolve a component and remove it from 3D Layout.
<code>duplicate_design(label[, save_after_duplicate])</code>	Copy a design to a new name.
<code>edit_cosim_options([...])</code>	Edit cosimulation options.
<code>edit_hfss_extents([diel_extent_type, ...])</code>	Edit HFSS 3D Layout extents.
<code>edit_setup(name, properties)</code>	Modify a setup.
<code>edit_source_from_file(source, input_file[, ...])</code>	Edit a source from file data.
<code>enable_rigid-flex()</code>	Turn on or off the rigid flex of a board with bending if available.
<code>evaluate_expression(expression_string)</code>	Evaluate a valid string expression and return the numerical value in SI units.
<code>export_3d_model([output_file])</code>	Export the Ecad model to a 3D file.
<code>export_convergence(setup_name[, ...])</code>	Export a solution convergence to a file.
<code>export_design_preview_to_jpg(filename)</code>	Export design preview image to a JPG file.
<code>export_mesh_stats(setup[, variation_string, ...])</code>	Export mesh statistics to a file.
<code>export_parametric_results(sweep, filename[, ...])</code>	Export a list of all parametric variations solved for a sweep to a CSV file.
<code>export_profile(setup_name[, ...])</code>	Export a solution profile to a PROF file.
<code>export_results([analyze, export_folder, ...])</code>	Export all available reports to a file, including profile, and convergence and sNp when applicable.
<code>export_rl_matrix(matrix_name, file_path[, ...])</code>	Export R/L matrix after solving.
<code>export_touchstone([setup_name, sweep_name, ...])</code>	Export a Touchstone file.
<code>export_variables_to_csv(filename[, ...])</code>	Export design properties, project variables, or both to a CSV file.
<code>generate_temp_project_directory(subdir_name)</code>	Generate a unique directory string to save a project to.

continues on next page

Table 13 – continued from previous page

<code>generate_unique_setup_name([setup_name])</code>	Generate a new setup with an unique name.
<code>get_all_insertion_loss_list([trlist, ...])</code>	Get a list of all insertion losses from two lists of excitations (driver and receiver).
<code>get_all_return_loss_list([excitation_names, ...])</code>	Get a list of all return losses for a list of excitations.
<code>get_dcir_element_data_current_source(setup)</code>	Get dcir element data current source.
<code>get_dcir_element_data_loop_resistance(setup)</code>	Get dcir element data loop resistance.
<code>get_dcir_element_data_via(setup)</code>	Get dcir element data via.
<code>get_dcir_solution_data(setup[, show, category])</code>	Retrieve dcir solution data.
<code>get_differential_pairs()</code>	Get the list defined differential pairs.
<code>get_evaluated_value(variable_name[, units])</code>	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
<code>get_fext_xtalk_list([trlist, reclist, ...])</code>	Get a list of all the far end XTalks from two lists of excitations (driver and receiver).
<code>get_model_from_mesh_results([binary])</code>	Get the path for the parasolid file in the result folder.
<code>get_module(module_name)</code>	Aedt Module object.
<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_next_xtalk_list([trlist, tx_prefix, ...])</code>	Get a list of all the near end XTalks from a list of excitations (driver and receiver).
<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.
<code>get_object_material_properties([assignment, ...])</code>	Retrieve the material properties for a list of objects and return them in a dictionary.
<code>get_oo_name(aedt_object[, object_name])</code>	Return the object-oriented AEDT property names.
<code>get_oo_object(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object.
<code>get_oo_properties(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_oo_property_value(aedt_object, ...)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_output_variable(variable[, solution])</code>	Retrieve the value of the output variable.
<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_setup(name[, setup_type])</code>	Retrieve a setup.
<code>get_setups()</code>	Retrieve setups.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_touchstone_data([setup_name, ...])</code>	Return a Touchstone data plot.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Retrieve a list of traces of specified designs ready to use in plot reports.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>import_awr(input_file[, output_dir, ...])</code>	Import an AWR Microwave Office file into HFSS 3D Layout and assign the stackup from an XML file if present.
<code>import_brd(input_file[, output_dir, ...])</code>	Import a board file into HFSS 3D Layout and assign the stackup from an XML file if present.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>import_dxf(input_file[, output_dir, ...])</code>	Import a DXF file into HFSS 3D Layout and assign the stackup from an XML file if present.
<code>import_edb(input_folder)</code>	Import EDB.
<code>import_gds(input_file[, output_dir, ...])</code>	Import a GDS file into HFSS 3D Layout and assign the stackup from an XML file if present.
<code>import_gerber(input_file[, output_dir, ...])</code>	Import a Gerber zip file into HFSS 3D Layout and assign the stackup from an XML file if present.

continues on next page

Table 13 – continued from previous page

<code>import_ipc2581(input_file[, output_dir, ...])</code>	Import an IPC2581 file into HFSS 3D Layout and assign the stackup from an XML file if present.
<code>import_odb(input_file[, output_dir, ...])</code>	Import an ODB++ file into HFSS 3D Layout and assign the stackup from an XML file if present.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_diff_pairs_from_file(input_file)</code>	Load differential pairs definition from a file.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>save_diff_pairs_to_file(output_file)</code>	Save differential pairs definition to a file.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_differential_pair(assignment, reference)</code>	Add a differential pair definition.
<code>set_export_touchstone(activate[, output_dir])</code>	Export the Touchstone file automatically if the simulation is successful.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_meshing_settings([mesh_method, ...])</code>	Define the settings of the mesh.
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>show_extent([show])</code>	Show or hide extent in a HFSS3dLayout design.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_full_design([name, output_dir, ports])</code>	Validate the design based on the expected value and save the information in the log file.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

Position	Position of the object.
SimulationSetupTypes	Simulation setup types.
SolutionTypes	Solution types.
active_setup	Get or Set the name of the active setup.
aedt_version_id	AEDT version.
are_there_simulations_running	Check if there are simulation running.
available_variations	Available variation object.
boundaries	Design boundaries and excitations.
boundaries_by_type	Design boundaries by type.
configurations	Property to import and export configuration files.
default_solution_type	Default solution type.
design_datasets	Dictionary of Design Datasets.
design_list	Design list.
design_name	Design name.
design_properties	Design properties.
design_type	Design type.
desktop_class	Desktop class.
desktop_install_dir	AEDT installation directory.
excitation_objects	Get all excitation.
excitations	Excitation names.
excitations_by_type	Design excitations by type.
existing_analysis_setups	Existing analysis setups in the design.
existing_analysis_sweeps	Existing analysis sweeps.
get_all_sparameter_list	List of all S parameters for a list of excitations.
ic_mode	IC mode of current design.
info	Dictionary of the PyAEDT session information.
layoutheditor	Return the Circuit Layout Editor.
library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
materials	Materials in the project.
mesh	Mesh.
modeler	Modeler object.
native_components	Native Component dictionary.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.

continues on next page

Table 14 – continued from previous page

<code>omatrix</code>	Matrix Object.
<code>omeshmodule</code>	Icepak Mesh Module.
<code>omodelsetup</code>	AEDT Model Setup Object.
<code>omonitor</code>	AEDT Monitor Object.
<code>ooptimetrics</code>	AEDT Optimetrics Module.
<code>ooutput_variable</code>	AEDT Output Variable Module.
<code>opadstackmanager</code>	AEDT oPadstackManager.
<code>oproject</code>	Project property.
<code>optimizations</code>	Optimizations in the project.
<code>oradfield</code>	AEDT Radiation Field Object.
<code>oreportsetup</code>	Report setup.
<code>osolution</code>	Solution Module.
<code>output_variables</code>	List of output variables.
<code>parametrics</code>	Setups in the project.
<code>personallib</code>	PersonalLib directory.
<code>port_list</code>	Port list.
<code>ports</code>	Design excitations.
<code>post</code>	PostProcessor.
<code>project_datasets</code>	Dictionary of project datasets.
<code>project_file</code>	Project name and path.
<code>project_list</code>	Project list.
<code>project_name</code>	Project name.
<code>project_path</code>	Project path.
<code>project_properties</code>	Project properties.
<code>project_time_stamp</code>	Return Project time stamp.
<code>project_timestamp_changed</code>	Return a bool if time stamp changed or not.
<code>pyaedt_dir</code>	Pyaedt directory.
<code>results_directory</code>	Results directory.
<code>settings</code>	Settings of the current Python/Pyaedt session.
<code>setup_names</code>	Setup names.
<code>setups</code>	Setups in the project.
<code>solution_type</code>	Solution type.
<code>src_dir</code>	Source directory for Python.
<code>syslib</code>	SysLib directory.
<code>temp_directory</code>	Path to the temporary directory.
<code>toolkit_directory</code>	Path to the toolkit directory.
<code>userlib</code>	UserLib directory.
<code>valid_design</code>	Valid design.
<code>variable_manager</code>	Variable manager for creating and managing project design and postprocessing variables.
<code>working_directory</code>	Path to the working directory.

2.1.9 pyaedt.mechanical.Mechanical

```
class pyaedt.mechanical.Mechanical(projectname=None, designname=None, solution_type=None,
                                     setup_name=None, specified_version=None, non_graphical=False,
                                     new_desktop_session=False, close_on_exit=False,
                                     student_version=False, machine='', port=0, aedt_process_id=None)
```

Provides the Mechanical application interface.

Parameters

projectname

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

setup_name

[`str`, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

specified_version

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active version or latest installed version is used. This parameter is ignored when a script is launched within AEDT. Examples of input values are `232`, `23.2`, `^2023.2`, `~2023.2`.

non_graphical

[`bool`, optional] Whether to launch AEDT in the non-graphical mode. The default is `False`, in which case AEDT is launched in the graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`. This parameter is ignored when a script is launched within AEDT.

machine

[`str`, optional] Machine name to connect the oDesktop session to. Works only in 2022R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[int, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is None. This parameter is only used when new_desktop_session = False.

Examples

Create an instance of Mechanical and connect to an existing HFSS design or create a new HFSS design if one does not exist.

```
>>> from pyaedt import Mechanical  
>>> aedtapp = Mechanical()
```

Create an instance of Mechanical and link to a project named "projectname". If this project does not exist, create one with this name.

```
>>> aedtapp = Mechanical(projectname)
```

Create an instance of Mechanical and link to a design named "designname" in a project named "projectname".

```
>>> aedtapp = Mechanical(projectname, designname)
```

Create an instance of Mechanical and open the specified project, which is named "myfile.aedt".

```
>>> aedtapp = Mechanical("myfile.aedt")
```

Create a Desktop on 2023 R2 object and then create an Mechanical object and open the specified project, which is named "myfile.aedt".

```
>>> aedtapp = Mechanical(specified_version=23.2, projectname="myfile.aedt")
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,  
specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,  
student_version=False, machine="", port=0, aedt_process_id=None)
```

Methods

activate_variable_optimization(variable_name[, ...])	Activate optimization analysis for a variable and optionally set up ranges.
activate_variable_sensitivity(variable_name)	Activate sensitivity analysis for a variable and optionally set up ranges.
activate_variable_statistical(variable_name)	Activate statistical analysis for a variable and optionally set up ranges.
activate_variable_tuning(variable_name[, ...])	Activate tuning analysis for a variable and optionally set up ranges.
add_error_message(message_text[, message_type])	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
add_from_toolkit(toolkit_object[, draw])	Add a new toolkit to the current application.

continues on next page

Table 15 – continued from previous page

<code>add_info_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_stackup_3d()</code>		Create a stackup 3D object.
<code>add_warning_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>		Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>		Analyze a design setup.
<code>archive_project([project_file, ...])</code>		Archive the AEDT project and add a message.
<code>assign_em_losses([design, setup, sweep, ...])</code>		Map EM losses to a Mechanical design.
<code>assign_fixed_support(assignment[, name])</code>		Assign a Mechanical fixed support.
<code>assign_frictionless_support(assignment[, name])</code>		Assign a Mechanical frictionless support.
<code>assign_heat_flux(assignment, heat_flux_type, ...)</code>		Assign heat flux boundary condition to an object or face list.
<code>assign_heat_generation(assignment, value[, name])</code>		Assign a heat generation boundary condition to an object list.
<code>assign_material(obj, mat)</code>		Assign a material to one or more objects.
<code>assign_thermal_map(object_list[, design, ...])</code>		Map thermal losses to a Mechanical design.
<code>assign_uniform_convection(assignment[, ...])</code>		Assign a uniform convection to the face list.
<code>assign_uniform_temperature(assignment[, ...])</code>		Assign a uniform temperature boundary.
<code>assignmaterial_from_sherlock_files(...)</code>		Assign material to objects in a design based on a CSV file obtained from Sherlock.
<code>autosave_disable()</code>		Disable autosave in AEDT.
<code>autosave_enable()</code>		Enable autosave in AEDT.
<code>change Automatically_use_causal_materials</code>		Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>		Set Design Settings.
<code>change_material_override([material_override])</code>		Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>		Change a property.
<code>change_validation_settings([...])</code>		Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>		Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>		Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>		Delete a project folder.
<code>cleanup_solution([variations, ...])</code>		Delete a set of Solution Variations or part of them.
<code>close_desktop()</code>		Close AEDT and release it.
<code>close_project([name, save_project])</code>		Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	de-	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>		Copy the project to another destination.
<code>copy_solid_bodies_from(design[, ...])</code>		Copy a list of objects and user defined models from one design to the active design.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>		Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>		Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>		Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>		Create a 3D dataset.
<code>create_new_project(proj_name)</code>		Create a project within AEDT.
<code>create_output_variable(variable, expression)</code>		Create or modify an output variable.
<code>create_setup([name, setup_type])</code>		Create an analysis setup for Mechanical.
<code>dataset_exists(name[, is_project_dataset])</code>		Check if a dataset exists.

continues on next page

Table 15 – continued from previous page

deactivate_variable_optimization(variable_name)	Deactivate the optimization analysis for a variable.
deactivate_variable_sensitivity(variable_name)	Deactivate the sensitivity analysis for a variable.
deactivate_variable_statistical(variable_name)	Deactivate the statistical analysis for a variable.
deactivate_variable_tuning(variable_name)	Deactivate the tuning analysis for a variable.
delete_design([name, fallback_design])	Delete a design from the current project.
delete_project(project_name)	Delete a project.
delete_separator(separator_name)	Delete a separator from either the active project or a design.
delete_setup(name)	Delete a setup.
delete_unused_variables()	Delete design and project unused variables.
delete_variable(sVarName)	Delete a variable.
design_variation([variation_string])	Generate a string to specify a desired variation.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
edit_setup(name, properties)	Modify a setup.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_3d_model([file_name, file_path, ...])	Export the 3D model.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_mesh_stats(setup[, variation_string, ...])	Export mesh statistics to a file.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.
export_rl_matrix(matrix_name, file_path[, ...])	Export R/L matrix after solving.
export_variables_to_csv(filename[, ...])	Export design properties, project variables, or both to a CSV file.
flatten_3d_components([component_name, ...])	Flatten one or multiple 3d components in the actual layout.
generate_temp_project_directory(subdir_name)	Generate a unique directory string to save a project to.
generate_unique_setup_name([setup_name])	Generate a new setup with an unique name.
get_all_conductors_names()	Retrieve all conductors in the active design.
get_all_dielectrics_names()	Retrieve all dielectrics in the active design.
get_all_sources()	Retrieve all setup sources.
get_components3d_vars(component3dname)	Read the A3DCOMP file and check for variables.
get_dxf_layers(file_path)	Read a DXF file and return all layer names.
get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.
get_monitor_data()	Check and get monitor data of an existing analysis.
get_nominal_variation([with_values])	Retrieve the nominal variation.
get_object_material_properties([assignment, ...])	Retrieve the material properties for a list of objects and return them in a dictionary.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.
get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_output_variable(variable[, solution])	Retrieve the value of the output variable.
get_property_value(objectname, property[, type])	Retrieve a property value.

continues on next page

Table 15 – continued from previous page

<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_setup(name)</code>	Get the setup from the current design.
<code>get_setups()</code>	Retrieve setups.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Retrieve a list of traces of specified designs ready to use in plot reports.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>identify_touching_conductors([object_name])</code>	Identify all touching components and group in a dictionary.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>import_dxf(file_path, layers_list[, ...])</code>	Import a DXF file.
<code>import_gds_3d(input_file, mapping_layers[, ...])</code>	Import a GDSII file.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>plot([objects, show, export_path, ...])</code>	Plot the model or a subset of objects.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_source_context(sources[, number_of_modes])</code>	Set the source context.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

Position	Position of the object.
SimulationSetupTypes	Simulation setup types.
SolutionTypes	Solution types.
active_setup	Get or Set the name of the active setup.
aedt_version_id	AEDT version.
are_there_simulations_running	Check if there are simulation running.
available_variations	Available variation object.
boundaries	Design boundaries and excitations.
boundaries_by_type	Design boundaries by type.
components3d	3D components.
configurations	Property to import and export configuration files.
default_solution_type	Default solution type.
design_datasets	Dictionary of Design Datasets.
design_list	Design list.
design_name	Design name.
design_properties	Design properties.
design_type	Design type.
desktop_class	Desktop class.
desktop_install_dir	AEDT installation directory.
excitation_objects	Get all excitation.
excitations	Get all excitation names.
excitations_by_type	Design excitations by type.
existing_analysis_setups	Existing analysis setups.
existing_analysis_sweeps	Existing analysis sweeps in the design.
info	Dictionary of the PyAEDT session information.
layoutheditor	Return the Circuit Layout Editor.
library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
materials	Materials in the project.
mesh	Mesh.
modeler	Modeler.
native_components	Native Component dictionary.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.

continues on next page

Table 16 – continued from previous page

omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
working_directory	Path to the working directory.

2.1.10 pyaedt.rmxprt.Rmxprt

```
class pyaedt.rmxprt.Rmxprt(projectname=None, designname=None, solution_type=None, model_units=None,
                           setup_name=None, specified_version=None, non_graphical=False,
                           new_desktop_session=False, close_on_exit=False, student_version=False,
                           machine='', port=0, aedt_process_id=None)
```

Provides the RMxprt app interface.

Parameters

projectname

[str, optional] Name of the project to select or the full path to the project or AEDTZ

archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

model_units

[`str`, optional] Model units.

setup_name

[`str`, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

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. Examples of input values are `232`, `23.2`, ```2023.2```, ```2023.2```.

non_graphical

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to open the AEDT student version. The default is `False`.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of RMxprt and connect to an existing RMxprt design or create a new RMxprt design if one does not exist.

```
>>> from pyaedt import Rmxprt
>>> app = Rmxprt()
```

Create an instance of Rmxprt and link to a project named "projectname". If this project does not exist, create one with this name.

```
>>> app = Rmxprt(projectname)
```

Create an instance of RMxprt and link to a design named "designname" in a project named "projectname".

```
>>> app = Rmxprt(projectname, designname)
```

Create an instance of RMxprt and open the specified project, which is "myfile.aedt".

```
>>> app = Rmxprt("myfile.aedt")
```

```
__init__(projectname=None, designname=None, solution_type=None, model_units=None,
        setup_name=None, specified_version=None, non_graphical=False, new_desktop_session=False,
        close_on_exit=False, student_version=False, machine="", port=0, aedt_process_id=None)
```

Methods

activate_variable_optimization(variable_name)	Activate optimization analysis for a variable and optionally set up ranges.
activate_variable_sensitivity(variable_name)	Activate sensitivity analysis for a variable and optionally set up ranges.
activate_variable_statistical(variable_name)	Activate statistical analysis for a variable and optionally set up ranges.
activate_variable_tuning(variable_name[, ...])	Activate tuning analysis for a variable and optionally set up ranges.
add_error_message(message_text[, sage_type])	mes-Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
add_from_toolkit(toolkit_object[, draw])	Add a new toolkit to the current application.
add_info_message(message_text[, sage_type])	mes-Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
add_warning_message(message_text[, sage_type])	mes-Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
analyze([name, cores, tasks, gpus, ...])	Solve the active design.
analyze_setup([name, cores, tasks, gpus, ...])	Analyze a design setup.
archive_project([project_file, ...])	Archive the AEDT project and add a message.
autosave_disable()	Disable autosave in AEDT.
autosave_enable()	Enable autosave in AEDT.
change Automatically_use_causal_materials	Enable or disable the automatic use of causal materials for lossy dielectrics.

continues on next page

Table 17 – continued from previous page

change_design_settings(settings)	Set Design Settings.
change_material_override([material_override])	Enable or disable the material override in the project.
change_property(aedt_object, tab_name, ...)	Change a property.
change_validation_settings([...])	Update the validation design settings.
check_beta_option_enabled(beta_option_name)	Check if a beta option is enabled.
check_if_project_is_loaded(project_path)	Check if a project path is already loaded in active Desktop.
clean_proj_folder([directory, name])	Delete a project folder.
close_desktop()	Close AEDT and release it.
close_project([name, save_project])	Close an AEDT project.
copy_design_from(project_fullname, sign_name)	Copy a design from a project into the active project.
copy_project(path, dest)	Copy the project to another destination.
create_dataset(dsname, xlist, ylist[, ...])	Create a dataset.
create_dataset1d_design(dsname, xlist, ylist)	Create a design dataset.
create_dataset1d_project(dsname, xlist, ylist)	Create a project dataset.
create_dataset3d(dsname, xlist, ylist[, ...])	Create a 3D dataset.
create_new_project(proj_name)	Create a project within AEDT.
create_output_variable(variable, expression)	Create or modify an output variable.
create_setup([name, setup_type])	Create an analysis setup for RmXprt.
dataset_exists(name[, is_project_dataset])	Check if a dataset exists.
deactivate_variable_optimization(variable_name)	Deactivate the optimization analysis for a variable.
deactivate_variable_sensitivity(variable_name)	Deactivate the sensitivity analysis for a variable.
deactivate_variable_statistical(variable_name)	Deactivate the statistical analysis for a variable.
deactivate_variable_tuning(variable_name)	Deactivate the tuning analysis for a variable.
delete_design([name, fallback_design])	Delete a design from the current project.
delete_project(project_name)	Delete a project.
delete_separator(separator_name)	Delete a separator from either the active project or a design.
delete_setup(name)	Delete a setup.
delete_unused_variables()	Delete design and project unused variables.
delete_variable(sVarName)	Delete a variable.
design_variation([variation_string])	Generate a string to specify a desired variation.
disable_modelcreation([solution_type])	Enable the RMxprt solution.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
edit_setup(name, properties)	Modify a setup.
enable_modelcreation([solution_type])	Enable model creation for the Maxwell model wizard.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.
export_rl_matrix(matrix_name, file_path[, ...])	Export R/L matrix after solving.
export_variables_to_csv(filename[, ...])	Export design properties, project variables, or both to a CSV file.
generate_temp_project_directory(subdir_name)	Generate a unique directory string to save a project to.
generate_unique_setup_name([setup_name])	Generate a new setup with an unique name.

continues on next page

Table 17 – continued from previous page

get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.
get_monitor_data()	Check and get monitor data of an existing analysis.
get_nominal_variation([with_values])	Retrieve the nominal variation.
get_object_material_properties([assignment, ...])	Retrieve the material properties for a list of objects and return them in a dictionary.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.
get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_output_variable(variable[, solution])	Retrieve the value of the output variable.
get_registry_key_int(key_full_name)	Get the value for the AEDT registry key if one exists.
get_registry_key_string(key_full_name)	Get the value for the AEDT registry key if one exists.
get_setup(name)	Get the setup from the current design.
get_setups()	Retrieve setups.
get_sweeps(name)	Retrieve all sweeps for a setup.
get_traces_for_plot([get_self_terms, ...])	Retrieve a list of traces of specified designs ready to use in plot reports.
hidden_variable(variable_name[, value])	Set the variable to a hidden or unhidden variable.
import_dataset1d(filename[, dsname, ...])	Import a 1D dataset.
import_dataset3d(filename[, dsname, ...])	Import a 3D dataset.
insert_design([design_name, solution_type])	Add a design of a specified type.
list_of_variations([setup_name, sweep_name])	Retrieve a list of active variations for input setup.
load_project(project_file[, design_name, ...])	Open an AEDT project based on a project and optional design.
number_with_units(value[, units])	Convert a number to a string with units.
read_design_data()	Read back the design data as a dictionary.
read_only_variable(variable_name[, value])	Set the variable to a read-only or not read-only variable.
release_desktop([close_projects, close_desktop])	Release AEDT.
remove_all_unused_definitions()	Remove all unused definitions in the project.
rename_design(new_name[, save_after_duplicate])	Rename the active design.
save_project([project_file, overwrite, ...])	Save the project and add a message.
set_active_design(name)	Change the active design to another design.
set_active_dso_config_name([product_name, ...])	Change a specific registry key to a new value.
set_license_type([license_type])	Change the license type between "Pack" and "Pool".
set_material_threshold([conductivity, ...])	Set material threshold.
set_oo_property_value(aedt_object, ...)	Change the property value of the object-oriented AEDT object.
set_registry_from_file(registry_file[, ...])	Apply desktop registry settings from an ACT file.
set_registry_key(key_full_name, key_value)	Change a specific registry key to a new value.
set_temporary_directory(temp_dir_path)	Set temporary directory path.
solve_in_batch([filename, machine, ...])	Analyze a design setup in batch mode.
stop_simulations([clean_stop])	Check if there are simulation running and stops them.
submit_job(clustername[, ...])	Submit a job to be solved on a cluster.
validate_simple([logfile])	Validate a design.

continues on next page

Table 17 – continued from previous page

<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".
--	---

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Machine design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>excitation_objects</code>	Get all excitation.
<code>excitations</code>	Get all excitation names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Existing analysis setups.
<code>existing_analysis_sweeps</code>	Existing analysis sweeps.
<code>info</code>	Dictionary of the PyAEDT session information.
<code>layouteditor</code>	Return the Circuit Layout Editor.
<code>library_list</code>	Library list.
<code>lock_file</code>	Lock file.
<code>logger</code>	Logger for the design.
<code>materials</code>	Materials in the project.
<code>modeler</code>	Modeler.
<code>native_components</code>	Native Component dictionary.
<code>nominal_adaptive</code>	Nominal adaptive sweep.
<code>nominal_sweep</code>	Nominal sweep.
<code>o_component_manager</code>	Component manager object.
<code>o_maxwell_parameters</code>	AEDT Maxwell Parameter Setup Object.
<code>o_model_manager</code>	Model manager object.
<code>o_symbol_manager</code>	Aedt Symbol Manager.
<code>oanalysis</code>	Analysis AEDT Module.
<code>oboundary</code>	Boundary Object.
<code>odefinition_manager</code>	Definition Manager Module.
<code>odesign</code>	Design.
<code>odesktop</code>	AEDT instance containing all projects and designs.
<code>oeditor</code>	Oeditor Module.
<code>oexcitation</code>	Solution Module.
<code>ofieldsreporter</code>	Fields reporter.
<code>oimport_export</code>	Import/Export Manager Module.

continues on next page

Table 18 – continued from previous page

omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	Post Object.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyaEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
working_directory	Path to the working directory.

2.1.11 pyaedt.circuit.Circuit

```
class pyaedt.circuit.Circuit(projectname=None, designname=None, solution_type=None,
                             setup_name=None, specified_version=None, non_graphical=False,
                             new_desktop_session=False, close_on_exit=False, student_version=False,
                             machine="", port=0, aedt_process_id=None)
```

Provides the Circuit application interface.

Parameters

projectname

[**str**, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is **None**, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[**str**, optional] Name of the design to select. The default is **None**, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[**str**, optional] Solution type to apply to the design. The default is **None**, in which case the default type is applied.

setup_name

[**str**, optional] Name of the setup to use as the nominal. The default is **None**, in which case the active setup is used or nothing is used.

specified_version

[**str**, **int**, **float**, optional] Version of AEDT to use. The default is **None**, in which case the active version or latest installed version is used. This parameter is ignored when Script is launched within AEDT. Examples of input values are `232`, `23.2`, `^2023.2`, `~2023.2`.

non_graphical

[**bool**, optional] Whether to run AEDT in non-graphical mode. The default is **False**, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[**bool**, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is **False**. This parameter is ignored when a script is launched within AEDT.

close_on_exit

[**bool**, optional] Whether to release AEDT on exit. The default is **False**.

student_version

[**bool**, optional] Whether to open the AEDT student version. The default is **False**. This parameter is ignored when Script is launched within AEDT.

machine

[**str**, optional] Machine name to which connect the oDesktop Session. Works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If a machine is `localhost`, the server also starts if not present.

port

[**int**, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[int, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is None. This parameter is only used when new_desktop_session = False.

Examples

Create an instance of Circuit and connect to an existing HFSS design or create a new HFSS design if one does not exist.

```
>>> from pyaedt import Circuit
>>> aedtapp = Circuit()
```

Create an instance of Circuit and link to a project named "projectname". If this project does not exist, create one with this name.

```
>>> aedtapp = Circuit(projectname)
```

Create an instance of Circuit and link to a design named "designname" in a project named "projectname".

```
>>> aedtapp = Circuit(projectname, designname)
```

Create an instance of Circuit and open the specified project, which is "myfile.aedt".

```
>>> aedtapp = Circuit("myfile.aedt")
```

Create an instance of Circuit using the 2023 R2 version and open the specified project, which is "myfile.aedt".

```
>>> aedtapp = Circuit(specified_version=2023.2, projectname="myfile.aedt")
```

Create an instance of Circuit using the 2023 R2 student version and open the specified project, which is named "myfile.aedt".

```
>>> hfss = Circuit(specified_version="2023.2", projectname="myfile.aedt", student_version=True)
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None, specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False, student_version=False, machine='', port=0, aedt_process_id=None)
```

Methods

activate_variable_optimization(variable_name)	Activate optimization analysis for a variable and optionally set up ranges.
activate_variable_sensitivity(variable_name)	Activate sensitivity analysis for a variable and optionally set up ranges.
activate_variable_statistical(variable_name)	Activate statistical analysis for a variable and optionally set up ranges.
activate_variable_tuning(variable_name[, ...])	Activate tuning analysis for a variable and optionally set up ranges.

continues on next page

Table 19 – continued from previous page

<code>add_error_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	mes-	Add a new toolkit to the current application.
<code>add_info_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_netlist_datablock(input_file[, name])</code>		Add a new netlist data block to the circuit schematic.
<code>add_warning_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>		Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>		Analyze a design setup.
<code>archive_project([project_file, ...])</code>		Archive the AEDT project and add a message.
<code>assign_current_sinusoidal_excitation_to_port(...)</code>		Assign a current sinusoidal excitation to circuit ports.
<code>assign_power_sinusoidal_excitation_to_port(...)</code>		Assign a power sinusoidal excitation to circuit ports.
<code>assign_voltage_frequency_dependent_excitation(...)</code>		Assign a frequency dependent excitation to circuit ports from a frequency dependent source (FDS format).
<code>assign_voltage_sinusoidal_excitation_to_port(...)</code>		Assign a voltage sinusoidal excitation to circuit ports.
<code>autosave_disable()</code>		Disable autosave in AEDT.
<code>autosave_enable()</code>		Enable autosave in AEDT.
<code>browse_log_file([input_file])</code>		Save the most recent log file in a new directory.
<code>change_automatically_use_causal_materials([enable])</code>		Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>		Set Design Settings.
<code>change_material_override([material_override])</code>		Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>		Change a property.
<code>change_validation_settings([...])</code>		Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>		Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>		Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>		Delete a project folder.
<code>close_desktop()</code>		Close AEDT and release it.
<code>close_project([name, save_project])</code>		Close an AEDT project.
<code>connect_circuit_models_from_multi_zone_circuit(...)</code>		Connect circuit model from a multizone clipped project.
<code>copy_design_from(project_fullname, sign_name)</code>	de-	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>		Copy the project to another destination.
<code>create_ami_schematic_from.snp(input_file, ...)</code>		Create a schematic from a Touchstone file and automatically set up an IBIS-AMI analysis.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>		Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>		Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>		Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>		Create a 3D dataset.
<code>create_lna_schematic_from.snp(input_file[, ...])</code>		Create a schematic from a Touchstone file and automatically set up an LNA analysis.
<code>create_new_project(proj_name)</code>		Create a project within AEDT.
<code>create_output_variable(variable, expression)</code>		Create or modify an output variable.
<code>create_schematic_from_mentor_netlist(input_file)</code>		Create a circuit schematic from a Mentor netlist.
<code>create_schematic_from_netlist(input_file)</code>		Create a circuit schematic from an HSpice netlist.

continues on next page

Table 19 – continued from previous page

<code>create_setup([name, setup_type])</code>	Create a setup.
<code>create_source(source_type[, name])</code>	Create a source in Circuit.
<code>create_tdr_schematic_from_snp(input_file, ...)</code>	Create a schematic from a Touchstone file and automatically setup a TDR transient analysis.
<code>create_touchstone_report(name, curves[, ...])</code>	Create a Touchstone plot.
<code>dataset_exists(name[, is_project_dataset])</code>	Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>	Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>	Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>	Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>	Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>	Delete a design from the current project.
<code>delete_project(project_name)</code>	Delete a project.
<code>delete_separator(separator_name)</code>	Delete a separator from either the active project or a design.
<code>delete_setup(name)</code>	Delete a setup.
<code>delete_unused_variables()</code>	Delete design and project unused variables.
<code>delete_variable(sVarName)</code>	Delete a variable.
<code>design_variation([variation_string])</code>	Generate a string to specify a desired variation.
<code>duplicate_design(label[, save_after_duplicate])</code>	Copy a design to a new name.
<code>edit_setup(name, properties)</code>	Modify a setup.
<code>evaluate_expression(expression_string)</code>	Evaluate a valid string expression and return the numerical value in SI units.
<code>export_convergence(setup_name[, ...])</code>	Export a solution convergence to a file.
<code>export_design_preview_to_jpg(filename)</code>	Export design preview image to a JPG file.
<code>export_fullwave_spice([design, setup, ...])</code>	Export a full wave HSpice file using NDE.
<code>export_parametric_results(sweep, filename[, ...])</code>	Export a list of all parametric variations solved for a sweep to a CSV file.
<code>export_profile(setup_name[, ...])</code>	Export a solution profile to a PROF file.
<code>export_results([analyze, export_folder, ...])</code>	Export all available reports to a file, including profile, and convergence and sNp when applicable.
<code>export_rl_matrix(matrix_name, file_path[, ...])</code>	Export R/L matrix after solving.
<code>export_touchstone([setup_name, sweep_name, ...])</code>	Export a Touchstone file.
<code>export_variables_to_csv(filename[, ...])</code>	Export design properties, project variables, or both to a CSV file.
<code>generate_temp_project_directory(subdir_name)</code>	Generate a unique directory string to save a project to.
<code>generate_unique_setup_name([setup_name])</code>	Generate a new setup with an unique name.
<code>get_all_insertion_loss_list([trlist, ...])</code>	Get a list of all insertion losses from two lists of excitations (driver and receiver).
<code>get_all_return_loss_list([excitation_names, ...])</code>	Get a list of all return losses for a list of excitations.
<code>get_evaluated_value(variable_name[, units])</code>	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
<code>get_fext_xtalk_list([trlist, reclist, ...])</code>	Get a list of all the far end XTalks from two lists of excitations (driver and receiver).
<code>get_ibis_model_from_file(input_file[, is_ami])</code>	Create an IBIS model based on the data contained in an IBIS file.
<code>get_module(module_name)</code>	Aedt Module object.
<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_next_xtalk_list([trlist, tx_prefix, ...])</code>	Get a list of all the near end XTalks from a list of excitations (driver and receiver).

continues on next page

Table 19 – continued from previous page

<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.
<code>get_object_material_properties([assignment, ...])</code>	Retrieve the material properties for a list of objects and return them in a dictionary.
<code>get_oo_name(aedt_object[, object_name])</code>	Return the object-oriented AEDT property names.
<code>get_oo_object(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object.
<code>get_oo_properties(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_oo_property_value(aedt_object, ...)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_output_variable(variable[, solution])</code>	Retrieve the value of the output variable.
<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_setup(name)</code>	Retrieve the setup from the current design.
<code>get_setups()</code>	Retrieve setups.
<code>get_source_pin_names(source_design_name[, ...])</code>	Retrieve pin names.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_touchstone_data([setup_name, ...])</code>	Return a Touchstone data plot.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Retrieve a list of traces of specified designs ready to use in plot reports.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>import_edb_in_circuit(input_dir)</code>	Import an EDB design inside a Circuit project.
<code>import_touchstone_solution(input_file[, ...])</code>	Import a Touchstone file as the solution.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_diff_pairs_from_file(input_file)</code>	Load differential pairs definition from a file.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>pop_up()</code>	Pop-up to parent Circuit design and reinitialize Circuit object.
<code>push_down(component_name)</code>	Push-down to the child component and reinitialize the Circuit object.
<code>push_exitations(instance[, ...])</code>	Push excitations for a linear frequency setup.
<code>push_time_exitations(instance[, start, ...])</code>	Push excitations for a transient setup.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>retrieve_mentor_comp(reference_id)</code>	Retrieve the type of the Mentor netlist component for a given reference ID.
<code>save_diff_pairs_to_file(output_file)</code>	Save differential pairs definition to a file.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_differential_pair(assignment, reference)</code>	Add a differential pair definition.

continues on next page

Table 19 – continued from previous page

<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.
<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>excitation_objects</code>	List of port objects.
<code>excitations</code>	List of port names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Analysis setups.
<code>existing_analysis_sweeps</code>	Analysis setups.
<code>get_all_sparameter_list</code>	List of all S parameters for a list of excitations.
<code>info</code>	Dictionary of the PyAEDT session information.
<code>layoutheditor</code>	Return the Circuit Layout Editor.
<code>library_list</code>	Library list.
<code>lock_file</code>	Lock file.
<code>logger</code>	Logger for the design.
<code>materials</code>	Materials in the project.
<code>modeler</code>	Modeler object.
<code>native_components</code>	Native Component dictionary.
<code>nominal_adaptive</code>	Nominal adaptive sweep.
<code>nominal_sweep</code>	Nominal sweep.
<code>o_component_manager</code>	Component manager object.

continues on next page

Table 20 – continued from previous page

o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	PostProcessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyaEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.
solution_type	Solution type.
source_names	Get all source names.
source_objects	Get all source objects.
sources	Get all sources.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.

continues on next page

Table 20 – continued from previous page

<code>valid_design</code>	Valid design.
<code>variable_manager</code>	Variable manager for creating and managing project design and postprocessing variables.
<code>working_directory</code>	Path to the working directory.

2.1.12 `pyaedt.maxwellcircuit.MaxwellCircuit`

```
class pyaedt.maxwellcircuit.MaxwellCircuit(projectname=None, designname=None,
                                             solution_type=None, specified_version=None,
                                             non_graphical=False, new_desktop_session=False,
                                             close_on_exit=False, student_version=False, machine="",
                                             port=0, aedt_process_id=None)
```

Provide the Maxwell Circuit application interface.

Parameters

`projectname`

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

`designname`

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

`specified_version`

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`. If `None`, the active setup is used or the latest installed version is used. Examples of input values are `232`, `23.2`, ```2023.2```, `2023.2`.

`non-graphical`

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

`new_desktop_session`

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`.

`close_on_exit`

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

`student_version`

[`bool`, optional] Whether to open the AEDT student version. The default is `False`.

`machine`

[`str`, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server is also started if not present.

`port`

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[int, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is None. This parameter is only used when new_desktop_session = False.

Examples

Create an instance of Maxwell Circuit and connect to an existing Maxwell circuit design or create a new Maxwell circuit design if one does not exist.

```
>>> from pyaedt import MaxwellCircuit  
>>> app = MaxwellCircuit()
```

Create an instance of Maxwell Circuit and link to a project named "projectname". If this project does not exist, create one with this name.

```
>>> app = MaxwellCircuit(projectname)
```

Create an instance of Maxwell Circuit and link to a design named "designname" in a project named "projectname".

```
>>> app = MaxwellCircuit(projectname, designname)
```

Create an instance of Maxwell Circuit and open the specified project, which is named "myfile.aedt".

```
>>> app = MaxwellCircuit("myfile.aedt")
```

```
__init__(projectname=None, designname=None, solution_type=None, specified_version=None,  
non_graphical=False, new_desktop_session=False, close_on_exit=False, student_version=False,  
machine='', port=0, aedt_process_id=None)
```

Constructor.

Methods

activate_variable_optimization(variable_name[, ...])	Activate optimization analysis for a variable and optionally set up ranges.
activate_variable_sensitivity(variable_name)	Activate sensitivity analysis for a variable and optionally set up ranges.
activate_variable_statistical(variable_name)	Activate statistical analysis for a variable and optionally set up ranges.
activate_variable_tuning(variable_name[, ...])	Activate tuning analysis for a variable and optionally set up ranges.
add_error_message(message_text[, sage_type])	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
add_from_toolkit(toolkit_object[, draw])	Add a new toolkit to the current application.
add_info_message(message_text[, sage_type])	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
add_warning_message(message_text[, sage_type])	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.

continues on next page

Table 21 – continued from previous page

analyze([name, cores, tasks, gpus, ...])	Solve the active design.
analyze_setup([name, cores, tasks, gpus, ...])	Analyze a design setup.
archive_project([project_file, ...])	Archive the AEDT project and add a message.
autosave_disable()	Disable autosave in AEDT.
autosave_enable()	Enable autosave in AEDT.
change Automatically_use_causal_materials([value])	Enable or disable the automatic use of causal materials for lossy dielectrics.
change_design_settings(settings)	Set Design Settings.
change_material_override([material_override])	Enable or disable the material override in the project.
change_property(aedt_object, tab_name, ...)	Change a property.
change_validation_settings([...])	Update the validation design settings.
check_beta_option_enabled(beta_option_name)	Check if a beta option is enabled.
check_if_project_is_loaded(project_path)	Check if a project path is already loaded in active Desktop.
clean_proj_folder([directory, name])	Delete a project folder.
close_desktop()	Close AEDT and release it.
close_project([name, save_project])	Close an AEDT project.
copy_design_from(project_fullname, sign_name)	Copy a design from a project into the active project.
copy_project(path, dest)	Copy the project to another destination.
create_dataset(dsname, xlist, ylist[, ...])	Create a dataset.
create_dataset1d_design(dsname, xlist, ylist)	Create a design dataset.
create_dataset1d_project(dsname, xlist, ylist)	Create a project dataset.
create_dataset3d(dsname, xlist, ylist[, ...])	Create a 3D dataset.
create_new_project(proj_name)	Create a project within AEDT.
create_output_variable(variable, expression)	Create or modify an output variable.
create_schematic_from_netlist(file_to_import)	Create a circuit schematic from an HSpice net list.
dataset_exists(name[, is_project_dataset])	Check if a dataset exists.
deactivate_variable_optimization(variable_name)	Deactivate the optimization analysis for a variable.
deactivate_variable_sensitivity(variable_name)	Deactivate the sensitivity analysis for a variable.
deactivate_variable_statistical(variable_name)	Deactivate the statistical analysis for a variable.
deactivate_variable_tuning(variable_name)	Deactivate the tuning analysis for a variable.
delete_design([name, fallback_design])	Delete a design from the current project.
delete_project(project_name)	Delete a project.
delete_separator(separator_name)	Delete a separator from either the active project or a design.
delete_setup(name)	Delete a setup.
delete_unused_variables()	Delete design and project unused variables.
delete_variable(sVarName)	Delete a variable.
design_variation([variation_string])	Generate a string to specify a desired variation.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
edit_setup(name, properties)	Modify a setup.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_netlist_from_schematic(file_to_export)	Create netlist from schematic circuit.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.

continues on next page

Table 21 – continued from previous page

<code>export_rl_matrix(matrix_name, file_path[, ...])</code>	Export R/L matrix after solving.
<code>export_variables_to_csv(filename[, ...])</code>	Export design properties, project variables, or both to a CSV file.
<code>generate_temp_project_directory(subdir_name)</code>	Generate a unique directory string to save a project to.
<code>generate_unique_setup_name([setup_name])</code>	Generate a new setup with an unique name.
<code>get_evaluated_value(variable_name[, units])</code>	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
<code>get_module(module_name)</code>	Aedt Module object.
<code>get_monitor_data()</code>	Check and get monitor data of an existing analysis.
<code>get_nominal_variation([with_values])</code>	Retrieve the nominal variation.
<code>get_object_material_properties([assignment, ...])</code>	Retrieve the material properties for a list of objects and return them in a dictionary.
<code>get_oo_name(aedt_object[, object_name])</code>	Return the object-oriented AEDT property names.
<code>get_oo_object(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object.
<code>get_oo_properties(aedt_object, object_name)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_oo_property_value(aedt_object, ...)</code>	Return the Object Oriented AEDT Object Properties.
<code>get_output_variable(variable[, solution])</code>	Retrieve the value of the output variable.
<code>get_registry_key_int(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_registry_key_string(key_full_name)</code>	Get the value for the AEDT registry key if one exists.
<code>get_setup(name)</code>	Get the setup from the current design.
<code>get_setups()</code>	Retrieve setups.
<code>get_sweeps(name)</code>	Retrieve all sweeps for a setup.
<code>get_traces_for_plot([get_self_terms, ...])</code>	Retrieve a list of traces of specified designs ready to use in plot reports.
<code>hidden_variable(variable_name[, value])</code>	Set the variable to a hidden or unhidden variable.
<code>import_dataset1d(filename[, dsname, ...])</code>	Import a 1D dataset.
<code>import_dataset3d(filename[, dsname, ...])</code>	Import a 3D dataset.
<code>insert_design([design_name, solution_type])</code>	Add a design of a specified type.
<code>list_of_variations([setup_name, sweep_name])</code>	Retrieve a list of active variations for input setup.
<code>load_project(project_file[, design_name, ...])</code>	Open an AEDT project based on a project and optional design.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>read_design_data()</code>	Read back the design data as a dictionary.
<code>read_only_variable(variable_name[, value])</code>	Set the variable to a read-only or not read-only variable.
<code>release_desktop([close_projects, close_desktop])</code>	Release AEDT.
<code>remove_all_unused_definitions()</code>	Remove all unused definitions in the project.
<code>rename_design(new_name[, save_after_duplicate])</code>	Rename the active design.
<code>save_project([project_file, overwrite, ...])</code>	Save the project and add a message.
<code>set_active_design(name)</code>	Change the active design to another design.
<code>set_active_dso_config_name([product_name, ...])</code>	Change a specific registry key to a new value.
<code>set_license_type([license_type])</code>	Change the license type between "Pack" and "Pool".
<code>set_oo_property_value(aedt_object, ...)</code>	Change the property value of the object-oriented AEDT object.
<code>set_registry_from_file(registry_file[, ...])</code>	Apply desktop registry settings from an ACT file.
<code>set_registry_key(key_full_name, key_value)</code>	Change a specific registry key to a new value.
<code>set_temporary_directory(temp_dir_path)</code>	Set temporary directory path.

continues on next page

Table 21 – continued from previous page

<code>solve_in_batch([filename, machine, ...])</code>	Analyze a design setup in batch mode.
<code>stop_simulations([clean_stop])</code>	Check if there are simulation running and stops them.
<code>submit_job(clustername[, ...])</code>	Submit a job to be solved on a cluster.
<code>validate_simple([logfile])</code>	Validate a design.
<code>value_with_units(value[, units, unit_system])</code>	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

<code>Position</code>	Position of the object.
<code>SimulationSetupTypes</code>	Simulation setup types.
<code>SolutionTypes</code>	Solution types.
<code>active_setup</code>	Get or Set the name of the active setup.
<code>aedt_version_id</code>	AEDT version.
<code>are_there_simulations_running</code>	Check if there are simulation running.
<code>available_variations</code>	Available variation object.
<code>boundaries</code>	Design boundaries and excitations.
<code>boundaries_by_type</code>	Design boundaries by type.
<code>default_solution_type</code>	Default solution type.
<code>design_datasets</code>	Dictionary of Design Datasets.
<code>design_list</code>	Design list.
<code>design_name</code>	Design name.
<code>design_properties</code>	Design properties.
<code>design_type</code>	Design type.
<code>desktop_class</code>	Desktop class.
<code>desktop_install_dir</code>	AEDT installation directory.
<code>excitation_objects</code>	Get all excitation.
<code>excitations</code>	Get all excitation names.
<code>excitations_by_type</code>	Design excitations by type.
<code>existing_analysis_setups</code>	Existing analysis setups.
<code>existing_analysis_sweeps</code>	Existing analysis sweeps.
<code>info</code>	Dictionary of the PyAEDT session information.
<code>layoutheditor</code>	Return the Circuit Layout Editor.
<code>library_list</code>	Library list.
<code>lock_file</code>	Lock file.
<code>logger</code>	Logger for the design.
<code>materials</code>	Materials in the project.
<code>modeler</code>	Design oModeler.
<code>native_components</code>	Native Component dictionary.
<code>nominal_adaptive</code>	Nominal adaptive sweep.
<code>nominal_sweep</code>	Nominal sweep.
<code>o_component_manager</code>	Component manager object.
<code>o_maxwell_parameters</code>	AEDT Maxwell Parameter Setup Object.
<code>o_model_manager</code>	Model manager object.
<code>o_symbol_manager</code>	Aedt Symbol Manager.
<code>oanalysis</code>	Analysis AEDT Module.
<code>oboundary</code>	Boundary Object.
<code>odefinition_manager</code>	Definition Manager Module.
<code>odesign</code>	Design.
<code>odesktop</code>	AEDT instance containing all projects and designs.

continues on next page

Table 22 – continued from previous page

<code>oeditor</code>	Oeditor Module.
<code>oexcitation</code>	Solution Module.
<code>ofieldsreporter</code>	Fields reporter.
<code>oimport_export</code>	Import/Export Manager Module.
<code>omaterial_manager</code>	Material Manager Module.
<code>omatrix</code>	Matrix Object.
<code>omeshmodule</code>	Icepak Mesh Module.
<code>omodelsetup</code>	AEDT Model Setup Object.
<code>omonitor</code>	AEDT Monitor Object.
<code>ooptimetrics</code>	AEDT Optimetrics Module.
<code>ooutput_variable</code>	AEDT Output Variable Module.
<code>opadstackmanager</code>	AEDT oPadstackManager.
<code>oproject</code>	Project property.
<code>optimizations</code>	Optimizations in the project.
<code>oradfield</code>	AEDT Radiation Field Object.
<code>oreportsetup</code>	Report setup.
<code>osolution</code>	Solution Module.
<code>output_variables</code>	List of output variables.
<code>parametrics</code>	Setups in the project.
<code>personallib</code>	PersonalLib directory.
<code>ports</code>	Design excitations.
<code>project_datasets</code>	Dictionary of project datasets.
<code>project_file</code>	Project name and path.
<code>project_list</code>	Project list.
<code>project_name</code>	Project name.
<code>project_path</code>	Project path.
<code>project_properties</code>	Project properties.
<code>project_time_stamp</code>	Return Project time stamp.
<code>project_timestamp_changed</code>	Return a bool if time stamp changed or not.
<code>pyaedt_dir</code>	PyAEDT directory.
<code>results_directory</code>	Results directory.
<code>settings</code>	Settings of the current Python/Pyaedt session.
<code>setup_names</code>	Setup names.
<code>setups</code>	Setups in the project.
<code>solution_type</code>	Solution type.
<code>src_dir</code>	Source directory for Python.
<code>syslib</code>	SysLib directory.
<code>temp_directory</code>	Path to the temporary directory.
<code>toolkit_directory</code>	Path to the toolkit directory.
<code>userlib</code>	UserLib directory.
<code>valid_design</code>	Valid design.
<code>variable_manager</code>	Variable manager for creating and managing project design and postprocessing variables.
<code>working_directory</code>	Path to the working directory.

2.1.13 pyaedt.emit.Emit

```
class pyaedt.emit.Emit(projectname=None, designname=None, solution_type=None, specified_version=None,
                      non_graphical=False, new_desktop_session=True, close_on_exit=True,
                      student_version=False, machine='', port=0, aedt_process_id=None)
```

Provides the EMIT application interface.

Parameters

projectname

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

designname

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

solution_type

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

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. Examples of input values are `232`, `23.2`, `2023.2`, `2023.2`.

non_graphical

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

new_desktop_session

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`.

close_on_exit

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

student_version

[`bool`, optional] Whether to start the AEDT student version. The default is `False`.

port

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a server. This parameter works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. The default is `0`.

machine

[`str`, optional] Machine name that the Desktop session is to connect to. This parameter works only in 2022 R2 and later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server starts if it is not present.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an `Emit` instance. You can also choose to define parameters for this instance here.

```
>>> from pyaedt import Emit  
>>> aedtapp = Emit()
```

Typically, it is desirable to specify a project name, design name, and other parameters.

```
>>> aedtapp = Emit(projectname, designname, specified_version=232)
```

Once an `Emit` instance is initialized, you can edit the schematic:

```
>>> rad1 = aedtapp.modeler.components.create_component("Bluetooth")  
>>> ant1 = aedtapp.modeler.components.create_component("Antenna")  
>>> if rad1 and ant1:  
>>>     ant1.move_and_connect_to(rad1)
```

Once the schematic is generated, the `Emit` object can be analyzed to generate a revision. Each revision is added as an element of the `Emit` object members `Results.revisions` list.

```
>>> revision = aedtapp.results.analyze()
```

A revision within PyAEDT is analogous to a revision in AEDT. An interaction domain must be defined and then used as the input to the run command used on that revision.

```
>>> domain = aedtapp.results.interaction_domain()  
>>> domain.rx_radio_name = "UE - HandHeld"  
>>> interaction = revision.run(domain)
```

The output of the run command is an `interaction` object. This object summarizes the interaction data that is defined in the interaction domain.

```
>>> instance = interaction.worst_instance(ResultType.SENSITIVITY)  
>>> val = instance.value(ResultType.SENSITIVITY)  
>>> print("Worst-case sensitivity for Rx '{}' is {}dB.".format(domain.rx_radio_name,  
    val))
```

```
__init__(projectname=None, designname=None, solution_type=None, specified_version=None,  
non_graphical=False, new_desktop_session=True, close_on_exit=True, student_version=False,  
machine='', port=0, aedt_process_id=None)
```

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name)</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name)</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.

continues on next page

Table 23 – continued from previous page

<code>add_error_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	mes-	Add a new toolkit to the current application.
<code>add_info_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_warning_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>archive_project([project_file, ...])</code>		Archive the AEDT project and add a message.
<code>autosave_disable()</code>		Disable autosave in AEDT.
<code>autosave_enable()</code>		Enable autosave in AEDT.
<code>change Automatically_use_causal_materials()</code>		Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>		Set Design Settings.
<code>change_material_override([material_override])</code>		Enable or disable the material override in the project.
<code>change_validation_settings([...])</code>		Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>		Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>		Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>		Delete a project folder.
<code>close_desktop()</code>		Close AEDT and release it.
<code>close_project([name, save_project])</code>		Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	de-	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>		Copy the project to another destination.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>		Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>		Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>		Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>		Create a 3D dataset.
<code>create_new_project(proj_name)</code>		Create a project within AEDT.
<code>dataset_exists(name[, is_project_dataset])</code>		Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>		Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>		Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>		Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>		Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>		Delete a design from the current project.
<code>delete_project(project_name)</code>		Delete a project.
<code>delete_separator(separator_name)</code>		Delete a separator from either the active project or a design.
<code>delete_unused_variables()</code>		Delete design and project unused variables.
<code>delete_variable(sVarName)</code>		Delete a variable.
<code>design_variation([variation_string])</code>		Generate a string to specify a desired variation.
<code>duplicate_design(label[, save_after_duplicate])</code>		Copy a design to a new name.
<code>evaluate_expression(expression_string)</code>		Evaluate a valid string expression and return the numerical value in SI units.
<code>export_design_preview_to_jpg(filename)</code>		Export design preview image to a JPG file.
<code>export_profile(setup_name[, ...])</code>		Export a solution profile to a PROF file.
<code>export_variables_to_csv(filename[, ...])</code>		Export design properties, project variables, or both to a CSV file.

continues on next page

Table 23 – continued from previous page

generate_temp_project_directory(subdir_name)	Generate a unique directory string to save a project to.
get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.
get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_registry_key_int(key_full_name)	Get the value for the AEDT registry key if one exists.
get_registry_key_string(key_full_name)	Get the value for the AEDT registry key if one exists.
get_units([unit_type])	Get units for the EMIT design.
hidden_variable(variable_name[, value])	Set the variable to a hidden or unhidden variable.
import_dataset1d(filename[, dsname, ...])	Import a 1D dataset.
import_dataset3d(filename[, dsname, ...])	Import a 3D dataset.
insert_design([design_name, solution_type])	Add a design of a specified type.
load_project(project_file[, design_name, ...])	Open an AEDT project based on a project and optional design.
read_design_data()	Read back the design data as a dictionary.
read_only_variable(variable_name[, value])	Set the variable to a read-only or not read-only variable.
release_desktop([close_projects, close_desktop])	Release AEDT.
remove_all_unused_definitions()	Remove all unused definitions in the project.
rename_design(new_name[, save_after_duplicate])	Rename the active design.
save_project([project_file, overwrite, ...])	Save the project and add a message.
set_active_design(name)	Change the active design to another design.
set_active_dso_config_name([product_name, ...])	Change a specific registry key to a new value.
set_license_type([license_type])	Change the license type between "Pack" and "Pool".
set_oo_property_value(aedt_object, ...)	Change the property value of the object-oriented AEDT object.
set_registry_from_file(registry_file[, ...])	Apply desktop registry settings from an ACT file.
set_registry_key(key_full_name, key_value)	Change a specific registry key to a new value.
set_temporary_directory(temp_dir_path)	Set temporary directory path.
set_units(unit_type, unit_value)	Set units for the EMIT design.
validate_simple([logfile])	Validate a design.
version([detailed])	Get version information.

Attributes

aedt_version_id	AEDT version.
boundaries	Design boundaries and excitations.
boundaries_by_type	Design boundaries by type.
couplings	EMIT Couplings.
default_solution_type	Default solution type.
design_datasets	Dictionary of Design Datasets.
design_list	Design list.
design_name	Design name.

continues on next page

Table 24 – continued from previous page

design_properties	Design properties.
design_type	Design type.
desktop_class	Desktop class.
desktop_install_dir	AEDT installation directory.
info	Dictionary of the PyAEDT session information.
layoutheditor	Return the Circuit Layout Editor.
library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
modeler	Modeler.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
personallib	PersonalLib directory.
ports	Design excitations.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	Pyaedt directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.

continues on next page

Table 24 – continued from previous page

<code>toolkit_directory</code>	Path to the toolkit directory.
<code>userlib</code>	UserLib directory.
<code>valid_design</code>	Valid design.
<code>variable_manager</code>	Variable manager for creating and managing project design and postprocessing variables.
<code>working_directory</code>	Path to the working directory.
<code>results</code>	"Result" object for the selected design.

2.1.14 `pyaedt.twinbuilder.TwinBuilder`

```
class pyaedt.twinbuilder.TwinBuilder(projectname=None, designname=None, solution_type=None,
                                      setup_name=None, specified_version=None, non_graphical=False,
                                      new_desktop_session=False, close_on_exit=False,
                                      student_version=False, machine="", port=0,
                                      aedt_process_id=None)
```

Provides the Twin Builder application interface.

Parameters

`projectname`

[`str`, optional] Name of the project to select or the full path to the project or AEDTZ archive to open. The default is `None`, in which case an attempt is made to get an active project. If no projects are present, an empty project is created.

`designname`

[`str`, optional] Name of the design to select. The default is `None`, in which case an attempt is made to get an active design. If no designs are present, an empty design is created.

`solution_type`

[`str`, optional] Solution type to apply to the design. The default is `None`, in which case the default type is applied.

`setup_name`

[`str`, optional] Name of the setup to use as the nominal. The default is `None`, in which case the active setup is used or nothing is used.

`specified_version`

[`str`, `int`, `float`, optional] Version of AEDT to use. The default is `None`, in which case the active setup or latest installed version is used. Examples of input values are 232, 23.2, ``2023.2``, ``2023.2``.

`non_graphical`

[`bool`, optional] Whether to launch AEDT in non-graphical mode. The default is `False`, in which case AEDT is launched in graphical mode. This parameter is ignored when a script is launched within AEDT.

`new_desktop_session`

[`bool`, optional] Whether to launch an instance of AEDT in a new thread, even if another instance of the `specified_version` is active on the machine. The default is `False`.

`close_on_exit`

[`bool`, optional] Whether to release AEDT on exit. The default is `False`.

`student_version`

[`bool`, optional] Whether to open the AEDT student version. The default is `False`.

machine

[`str`, optional] Machine name to connect the oDesktop session to. This works only in 2022 R2 or later. The remote server must be up and running with the command `ansysedt.exe -grpcsrv portnum`. If the machine is `localhost`, the server also starts if not present.

port

[`int`, optional] Port number on which to start the oDesktop communication on an already existing server. This parameter is ignored when creating a new server. It works only in 2022 R2 or later. The remote server must be up and running with command `ansysedt.exe -grpcsrv portnum`.

aedt_process_id

[`int`, optional] Process ID for the instance of AEDT to point PyAEDT at. The default is `None`. This parameter is only used when `new_desktop_session = False`.

Examples

Create an instance of Twin Builder and connect to an existing Maxwell design or create a new Maxwell design if one does not exist.

```
>>> from pyaedt import TwinBuilder
>>> app = TwinBuilder()
```

Create a instance of Twin Builder and link to a project named "projectname". If this project does not exist, create one with this name.

```
>>> app = TwinBuilder(projectname)
```

Create an instance of Twin Builder and link to a design named "designname" in a project named "projectname".

```
>>> app = TwinBuilder(projectname, designname)
```

Create an instance of Twin Builder and open the specified project, which is named "myfile.aedt".

```
>>> app = TwinBuilder("myfile.aedt")
```

```
__init__(projectname=None, designname=None, solution_type=None, setup_name=None,
        specified_version=None, non_graphical=False, new_desktop_session=False, close_on_exit=False,
        student_version=False, machine='', port=0, aedt_process_id=None)
```

Constructor.

Methods

<code>activate_variable_optimization(variable_name[, ...])</code>	Activate optimization analysis for a variable and optionally set up ranges.
<code>activate_variable_sensitivity(variable_name)</code>	Activate sensitivity analysis for a variable and optionally set up ranges.
<code>activate_variable_statistical(variable_name)</code>	Activate statistical analysis for a variable and optionally set up ranges.
<code>activate_variable_tuning(variable_name[, ...])</code>	Activate tuning analysis for a variable and optionally set up ranges.

continues on next page

Table 25 – continued from previous page

<code>add_error_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Error" message to either the global, active project, or active design level of the message manager tree.
<code>add_from_toolkit(toolkit_object[, draw])</code>	mes-	Add a new toolkit to the current application.
<code>add_info_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Info" message to either the global, active project, or active design level of the message manager tree.
<code>add_q3d_dynamic_component(source_project, ...)</code>		Add a Q2D or Q3D dynamic component to a Twin Builder design.
<code>add_warning_message(message_text[, sage_type])</code>	mes-	Add a type 0 "Warning" message to either the global, active project, or active design level of the message manager tree.
<code>analyze([name, cores, tasks, gpus, ...])</code>		Solve the active design.
<code>analyze_setup([name, cores, tasks, gpus, ...])</code>		Analyze a design setup.
<code>archive_project([project_file, ...])</code>		Archive the AEDT project and add a message.
<code>autosave_disable()</code>		Disable autosave in AEDT.
<code>autosave_enable()</code>		Enable autosave in AEDT.
<code>change Automatically_use_causal_materials</code>		Enable or disable the automatic use of causal materials for lossy dielectrics.
<code>change_design_settings(settings)</code>		Set Design Settings.
<code>change_material_override([material_override])</code>		Enable or disable the material override in the project.
<code>change_property(aedt_object, tab_name, ...)</code>		Change a property.
<code>change_validation_settings([...])</code>		Update the validation design settings.
<code>check_beta_option_enabled(beta_option_name)</code>		Check if a beta option is enabled.
<code>check_if_project_is_loaded(project_path)</code>		Check if a project path is already loaded in active Desktop.
<code>clean_proj_folder([directory, name])</code>		Delete a project folder.
<code>close_desktop()</code>		Close AEDT and release it.
<code>close_project([name, save_project])</code>		Close an AEDT project.
<code>copy_design_from(project_fullname, sign_name)</code>	de-	Copy a design from a project into the active project.
<code>copy_project(path, dest)</code>		Copy the project to another destination.
<code>create_dataset(dsname, xlist, ylist[, ...])</code>		Create a dataset.
<code>create_dataset1d_design(dsname, xlist, ylist)</code>		Create a design dataset.
<code>create_dataset1d_project(dsname, xlist, ylist)</code>		Create a project dataset.
<code>create_dataset3d(dsname, xlist, ylist[, ...])</code>		Create a 3D dataset.
<code>create_new_project(proj_name)</code>		Create a project within AEDT.
<code>create_output_variable(variable, expression)</code>		Create or modify an output variable.
<code>create_schematic_from_netlist(input_file)</code>		Create a circuit schematic from an HSpice net list.
<code>create_setup([name, setup_type])</code>		Create a setup.
<code>dataset_exists(name[, is_project_dataset])</code>		Check if a dataset exists.
<code>deactivate_variable_optimization(variable_name)</code>		Deactivate the optimization analysis for a variable.
<code>deactivate_variable_sensitivity(variable_name)</code>		Deactivate the sensitivity analysis for a variable.
<code>deactivate_variable_statistical(variable_name)</code>		Deactivate the statistical analysis for a variable.
<code>deactivate_variable_tuning(variable_name)</code>		Deactivate the tuning analysis for a variable.
<code>delete_design([name, fallback_design])</code>		Delete a design from the current project.
<code>delete_project(project_name)</code>		Delete a project.
<code>delete_separator(separator_name)</code>		Delete a separator from either the active project or a design.
<code>delete_setup(name)</code>		Delete a setup.
<code>delete_unused_variables()</code>		Delete design and project unused variables.
<code>delete_variable(sVarName)</code>		Delete a variable.

continues on next page

Table 25 – continued from previous page

design_variation([variation_string])	Generate a string to specify a desired variation.
duplicate_design(label[, save_after_duplicate])	Copy a design to a new name.
edit_setup(name, properties)	Modify a setup.
evaluate_expression(expression_string)	Evaluate a valid string expression and return the numerical value in SI units.
export_convergence(setup_name[, ...])	Export a solution convergence to a file.
export_design_preview_to_jpg(filename)	Export design preview image to a JPG file.
export_parametric_results(sweep, filename[, ...])	Export a list of all parametric variations solved for a sweep to a CSV file.
export_profile(setup_name[, ...])	Export a solution profile to a PROF file.
export_results([analyze, export_folder, ...])	Export all available reports to a file, including profile, and convergence and sNp when applicable.
export_rl_matrix(matrix_name, file_path[, ...])	Export R/L matrix after solving.
export_variables_to_csv(filename[, ...])	Export design properties, project variables, or both to a CSV file.
generate_temp_project_directory(subdir_name)	Generate a unique directory string to save a project to.
generate_unique_setup_name([setup_name])	Generate a new setup with an unique name.
get_evaluated_value(variable_name[, units])	Retrieve the evaluated value of a design property or project variable in SI units if no unit is provided.
get_module(module_name)	Aedt Module object.
get_monitor_data()	Check and get monitor data of an existing analysis.
get_nominal_variation([with_values])	Retrieve the nominal variation.
get_object_material_properties([assignment, ...])	Retrieve the material properties for a list of objects and return them in a dictionary.
get_oo_name(aedt_object[, object_name])	Return the object-oriented AEDT property names.
get_oo_object(aedt_object, object_name)	Return the Object Oriented AEDT Object.
get_oo_properties(aedt_object, object_name)	Return the Object Oriented AEDT Object Properties.
get_oo_property_value(aedt_object, ...)	Return the Object Oriented AEDT Object Properties.
get_output_variable(variable[, solution])	Retrieve the value of the output variable.
get_registry_key_int(key_full_name)	Get the value for the AEDT registry key if one exists.
get_registry_key_string(key_full_name)	Get the value for the AEDT registry key if one exists.
get_setup(name)	Get the setup from the current design.
get_setups()	Retrieve setups.
get_sweeps(name)	Retrieve all sweeps for a setup.
get_traces_for_plot([get_self_terms, ...])	Retrieve a list of traces of specified designs ready to use in plot reports.
hidden_variable(variable_name[, value])	Set the variable to a hidden or unhidden variable.
import_dataset1d(filename[, dsname, ...])	Import a 1D dataset.
import_dataset3d(filename[, dsname, ...])	Import a 3D dataset.
insert_design([design_name, solution_type])	Add a design of a specified type.
list_of_variations([setup_name, sweep_name])	Retrieve a list of active variations for input setup.
load_project(project_file[, design_name, ...])	Open an AEDT project based on a project and optional design.
number_with_units(value[, units])	Convert a number to a string with units.
read_design_data()	Read back the design data as a dictionary.
read_only_variable(variable_name[, value])	Set the variable to a read-only or not read-only variable.
release_desktop([close_projects, close_desktop])	Release AEDT.
remove_all_unused_definitions()	Remove all unused definitions in the project.

continues on next page

Table 25 – continued from previous page

rename_design(new_name[, save_after_duplicate])	Rename the active design.
save_project([project_file, overwrite, ...])	Save the project and add a message.
set_active_design(name)	Change the active design to another design.
set_active_dso_config_name([product_name, ...])	Change a specific registry key to a new value.
set_end_time(expression)	Set the end time.
set_hmax(expression)	Set hmax.
set_hmin(expression)	Set hmin.
set_license_type([license_type])	Change the license type between "Pack" and "Pool".
set_oo_property_value(aedt_object, ...)	Change the property value of the object-oriented AEDT object.
set_registry_from_file(registry_file[, ...])	Apply desktop registry settings from an ACT file.
set_registry_key(key_full_name, key_value)	Change a specific registry key to a new value.
set_sim_setup_parameter(variable, expression)	Set simulation setup parameters.
set_temporary_directory(temp_dir_path)	Set temporary directory path.
solve_in_batch([filename, machine, ...])	Analyze a design setup in batch mode.
stop_simulations([clean_stop])	Check if there are simulation running and stops them.
submit_job(clustername[, ...])	Submit a job to be solved on a cluster.
validate_simple([logfile])	Validate a design.
value_with_units(value[, units, unit_system])	Combine a number and a string containing the modeler length unit in a single string e.g. "1.2mm".

Attributes

Position	Position of the object.
SimulationSetupTypes	Simulation setup types.
SolutionTypes	Solution types.
active_setup	Get or Set the name of the active setup.
aedt_version_id	AEDT version.
are_there_simulations_running	Check if there are simulation running.
available_variations	Available variation object.
boundaries	Design boundaries and excitations.
boundaries_by_type	Design boundaries by type.
default_solution_type	Default solution type.
design_datasets	Dictionary of Design Datasets.
design_list	Design list.
design_name	Design name.
design_properties	Design properties.
design_type	Design type.
desktop_class	Desktop class.
desktop_install_dir	AEDT installation directory.
excitation_objects	Get all excitation.
excitations	Get all excitation names.
excitations_by_type	Design excitations by type.
existing_analysis_setups	Get all analysis solution setups.
existing_analysis_sweeps	Get all existing analysis setups.
info	Dictionary of the PyAEDT session information.
layouteditor	Return the Circuit Layout Editor.

continues on next page

Table 26 – continued from previous page

library_list	Library list.
lock_file	Lock file.
logger	Logger for the design.
materials	Materials in the project.
modeler	Design Modeler.
native_components	Native Component dictionary.
nominal_adaptive	Nominal adaptive sweep.
nominal_sweep	Nominal sweep.
o_component_manager	Component manager object.
o_maxwell_parameters	AEDT Maxwell Parameter Setup Object.
o_model_manager	Model manager object.
o_symbol_manager	Aedt Symbol Manager.
oanalysis	Analysis AEDT Module.
oboundary	Boundary Object.
odefinition_manager	Definition Manager Module.
odesign	Design.
odesktop	AEDT instance containing all projects and designs.
oeditor	Oeditor Module.
oexcitation	Solution Module.
ofieldsreporter	Fields reporter.
oimport_export	Import/Export Manager Module.
omaterial_manager	Material Manager Module.
omatrix	Matrix Object.
omeshmodule	Icepak Mesh Module.
omodelsetup	AEDT Model Setup Object.
omonitor	AEDT Monitor Object.
ooptimetrics	AEDT Optimetrics Module.
ooutput_variable	AEDT Output Variable Module.
opadstackmanager	AEDT oPadstackManager.
oproject	Project property.
optimizations	Optimizations in the project.
oradfield	AEDT Radiation Field Object.
oreportsetup	Report setup.
osolution	Solution Module.
output_variables	List of output variables.
parametrics	Setups in the project.
personallib	PersonalLib directory.
ports	Design excitations.
post	Design Postprocessor.
project_datasets	Dictionary of project datasets.
project_file	Project name and path.
project_list	Project list.
project_name	Project name.
project_path	Project path.
project_properties	Project properties.
project_time_stamp	Return Project time stamp.
project_timestamp_changed	Return a bool if time stamp changed or not.
pyaedt_dir	PyAEDT directory.
results_directory	Results directory.
settings	Settings of the current Python/Pyaedt session.
setup_names	Setup names.
setups	Setups in the project.

continues on next page

Table 26 – continued from previous page

solution_type	Solution type.
src_dir	Source directory for Python.
syslib	SysLib directory.
temp_directory	Path to the temporary directory.
toolkit_directory	Path to the toolkit directory.
userlib	UserLib directory.
valid_design	Valid design.
variable_manager	Variable manager for creating and managing project design and postprocessing variables.
working_directory	Path to the working directory.

All other classes and methods are inherited into the app class. AEDT, which is also referred to as the desktop app, is implicitly launched in any PyAEDT app. Before accessing a PyAEDT app, the desktop app must be launched and initialized. The desktop app can be explicitly or implicitly initialized as in the following examples.

Example with Desktop class explicit initialization:

```
from pyaedt import launch_desktop, Circuit
d = launch_desktop(specified_version="2023.1",
                     non_graphical=False,
                     new_desktop_session=True,
                     close_on_exit=True,
                     student_version=False):
    circuit = Circuit()
    ...
    # Any error here will be caught by Desktop.
    ...
d.release_desktop()
```

Example with Desktop class implicit initialization:

```
from pyaedt import Circuit
circuit = Circuit(specified_version="2023.1",
                  non_graphical=False,
                  new_desktop_session=True,
                  close_on_exit=True,
                  student_version=False):
    circuit = Circuit()
    ...
    # Any error here will be caught by Desktop.
    ...
circuit.release_desktop()
```

2.2 Material and stackup

This section lists material and stackup modules. These classes cannot be used directly but can be accessed through an app. Example:

2.2.1 Material management

This section describes all material-related classes and methods.

<i>MaterialLib.Materials</i>	Contains the AEDT materials database and all methods for creating and editing materials.
<i>Material.Material</i>	Manages material properties.
<i>Material.SurfaceMaterial</i>	Manages surface material properties for Icepak only.
<i>Material.MatProperties</i>	Contains a list of constant names for all materials with mappings to their internal XML names.
<i>Material.SurfMatProperties</i>	Contains a list of constant names for all surface materials with mappings to their internal XML names.
<i>Material.MatProperty</i>	Manages simple, anisotropic, tensor, and non-linear properties.

pyaedt.modules.MaterialLib.Materials

class pyaedt.modules.MaterialLib.**Materials**(app)

Contains the AEDT materials database and all methods for creating and editing materials.

Parameters

app

[pyaedt.application.Analysis3D.FieldAnalysis3D] Inherited parent object.

Examples

```
>>> from pyaedt import Hfss
>>> app = Hfss()
>>> materials = app.materials
```

__init__(app)

Methods

add_material(name[, properties])	Add a material with default values.
add_material_sweep(assignment, name)	Create a sweep material made of an array of materials.
add_surface_material(name[, emissivity])	Add a surface material.
check_thermal_modifier(material)	Check a material to see if it has any thermal modifiers.
checkifmaterialexists(material)	Check if a material exists in AEDT or PyAEDT Definitions.
duplicate_material(material[, name, properties])	Duplicate a material.
duplicate_surface_material(material, name)	Duplicate a surface material.
export_materials_to_file(output_file)	Export all materials to a JSON or TOML file.
get_used_project_material_names()	Get list of material names in current project.
import_materials_from_excel(input_file)	Import and create materials from a csv or excel file.
import_materials_from_file([input_file])	Import and create materials from a JSON or AMAT file.
remove_material(material[, library])	Remove a material.

Attributes

conductors	Conductors in the material database.
dielectrics	Dielectrics in the material database.
gases	Return the gas materials.
liquids	Return the liquids materials.
mat_names_aedt	List material names.
mat_names_aedt_lower	List material names with lower case.
odefinition_manager	Definition Manager from AEDT.
omaterial_manager	Material Manager from AEDT.
surface_material_keys	Dictionary of Surface Material in the project.

pyaedt.modules.Material.Material

class pyaedt.modules.Material.Material(materiallib, name, props=None, material_update=True)

Manages material properties.

Parameters

materiallib

[pyaedt.modules.MaterialLib.Materials] Inherited parent object.

name

[str] Name of the material.

props

The default is None.

material_update

[bool, optional] The default is True.

Examples

```
>>> from pyaedt import Hfss
>>> app = Hfss()
>>> material = app.materials["copper"]
```

```
__init__(materiallib, name, props=None, material_update=True)
```

Methods

get_core_loss_coefficients(points_at_frequency)	Get electrical steel or power ferrite core loss coefficients at a given frequency.
get_curve_coreloss_type()	Return the curve core loss type assigned to material.
get_curve_coreloss_values()	Return the curve core values type assigned to material.
get_magnetic_coercivity()	Get the magnetic coercivity values.
is_conductor([threshold])	Check if the material is a conductor.
is_dielectric([threshold])	Check if the material is dielectric.
set_bp_curve_coreloss(points[, kdc, ...])	Set B-P Type Core Loss.
set_coreloss_at_frequency(points_at_frequency)	Set electrical steel or power ferrite core loss model at one single frequency or at multiple frequencies.
set_djordjevic_sarkar_model([dk, df, ...])	Set Djordjevic-Sarkar model.
set_electrical_steel_coreloss([kh, kc, ke, ...])	Set electrical steel core loss.
set_hysteresis_coreloss([kdc, hci, br, hkc, ...])	Set Hysteresis Type Core Loss.
set_magnetic_coercivity([value, x, y, z])	Set magnetic coercivity for material.
set_power_ferrite_coreloss([cm, x, y, kdc, ...])	Set Power Ferrite Type Core Loss.
update()	Update the material in AEDT.

Attributes

conductivity	Conductivity.
coordinate_system	Material coordinate system.
dielectric_loss_tangent	Dielectric loss tangent.
diffusivity	Diffusivity.
is_used	Checks if a project material is in use.
magnetic_coercivity	Magnetic coercivity.
magnetic_loss_tangent	Magnetic loss tangent.
mass_density	Mass density.
material_appearance	Material appearance specified as a list where the first three items are RGB color and the fourth one is transparency.
molecular_mass	Molecular mass.
permeability	Permeability.
permittivity	Permittivity.
poissons_ratio	Poisson's ratio.
specific_heat	Specific heat.
stacking_direction	Stacking direction for the lamination can either be "V(1)", "V(2)" or "V(3)".
stacking_factor	Stacking factor for lamination.
stacking_type	Composition of the wire can either be "Solid", "Lamination" or "Litz Wire".
strand_number	Strand number for litz wire.
thermal_conductivity	Thermal conductivity.
thermal_expansion_coefficient	Thermal expansion coefficient.
viscosity	Viscosity.
wire_diameter	Diameter of the round litz wire.
wire_thickness	Thickness of rectangular litz wire.
wire_thickness_direction	Thickness direction of the wire can either be "V(1)", "V(2)" or "V(3)".
wire_type	The type of the wire can either be "Round", "Square" or "Rectangular".
wire_width	Width of the rectangular or square litz wire.
wire_width_direction	Width direction of the wire can either be "V(1)", "V(2)" or "V(3)".
youngs_modulus	Young's modulus.

pyaedt.modules.Material.SurfaceMaterial

```
class pyaedt.modules.Material.SurfaceMaterial(materiallib, name, props=None, material_update=True)
```

Manages surface material properties for Icepak only.

Parameters

materiallib

[*pyaedt.modules.MaterialLib.Materials*] Inherited parent object.

name

[*str*] Name of the surface material

props

The default is None.

material_update

[bool, optional] The default is True.

__init__(materiallib, name, props=None, material_update=True)

Methods

update()	Update the surface material in AEDT.
----------	--------------------------------------

Attributes

coordinate_system	Material coordinate system.
emissivity	Emissivity.
is_used	Checks if a project material is in use.
surface_diffuse_absorptance	Surface diffuse absorptance.
surface_incident_absorptance	Surface incident absorptance.
surface_roughness	Surface roughness.

pyaedt.modules.Material.MatProperties

class pyaedt.modules.Material.MatProperties

Contains a list of constant names for all materials with mappings to their internal XML names.

Internal names are used in scripts, and XML names are used in the XML syntax.

__init__(*args, **kwargs)

Methods

get_defaultunit([aedtname])	Retrieve the default unit for a full name or a category name.
get_defaultvalue(aedtname)	Retrieve the default value for a full name or a category name.

Attributes

aedtname

| cond_order |
| defaultunit |
| defaultvalue |
| diel_order |

pyaedt.modules.Material.SurfMatProperties**class pyaedt.modules.Material.SurfMatProperties**

Contains a list of constant names for all surface materials with mappings to their internal XML names.

Internal names are used in scripts, and XML names are used in the XML syntax.

`__init__(*args, **kwargs)`

Methods

<code>get_defaultunit([aedtname])</code>	Retrieve the default unit for a full name or a category name.
<code>get_defaultvalue([aedtname])</code>	Get the default value for a full name or a category name.

Attributes

`aedtname`

`defaultunit`

`defaultvalue`

pyaedt.modules.Material.MatProperty**class pyaedt.modules.Material.MatProperty(material, name, val=None, thermalmodifier=None, spatialmodifier=None)**

Manages simple, anisotropic, tensor, and non-linear properties.

Parameters**material**

[*pyaedt.modules.Material.Material*] Inherited parent object.

name

[*str*] Name of the material property.

val

The default is None.

thermalmodifier

The default is None.

spatialmodifier

The default is None.

Examples

```
>>> from pyaedt import Hfss
>>> app = Hfss()
>>> matproperty = app.materials["copper"].conductivity
```

```
__init__(material, name, val=None, thermalmodifier=None, spatialmodifier=None)
```

Methods

add_spatial_modifier_dataset(dataset[, index])	Add a spatial modifier to a material property using an existing dataset.
add_spatial_modifier_free_form(formula[, index])	Add a spatial modifier to a material property using a free-form formula.
add_thermal_modifier_closed_form([tref, c1, ...])	Add a thermal modifier to a material property using a closed-form formula.
add_thermal_modifier_dataset(dataset[, index])	Add a thermal modifier to a material property using an existing dataset.
add_thermal_modifier_free_form(formula[, index])	Add a thermal modifier to a material property using a free-form formula.

Attributes

data_set	Dataset.
evaluated_value	Evaluated value.
spatialmodifier	Spatial modifier.
thermalmodifier	Thermal modifier.
type	Type of the material property.
unit	Units for a material property value.
value	Value for a material property.

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call returns the Materials class
my_materials = app.materials
# This call returns the Material class
copper = my_materials["copper"]
# This property is from the MatProperty class
copper.conductivity
...
```

2.2.2 Stackup management

This section describes all layer-related classes and methods used in HFSS 3D Layout (and indirectly in Circuit).

<i>LayerStackup.Layers</i>	Manages stackup for the Circuit and HFSS 3D Layout tools.
<i>LayerStackup.Layer</i>	Manages the stackup layer for the Circuit and HFSS 3D Layout tools.

pyaedt.modules.LayerStackup.Layers

class `pyaedt.modules.LayerStackup.Layers(modeler, roughnessunits='um')`

Manages stackup for the Circuit and HFSS 3D Layout tools.

Parameters

modeler
[*pyaedt.modeler.modelerpcb.Modeler3DLayout*]

roughnessunits
[*str*, optional] Units for the roughness of layers. The default is "um".

Examples

```
>>> from pyaedt import Hfss3dLayout  
>>> app = Hfss3dLayout()  
>>> layers = app.modeler.layers
```

```
__init__(modeler, roughnessunits='um')
```

Methods

<code>add_layer(layer[, layer_type, thickness, ...])</code>	Add a layer.
<code>change_stackup_type([mode, number_zones])</code>	Change the stackup type between Multizone, Overlap and Laminate.
<code>layer_id(name)</code>	Retrieve a layer ID.

Attributes

LengthUnit	Length units.
all_diel_layers	All dielectric layers.
all_layers	All stackup layers.
all_signal_layers	All signal layers.
dielectrics	All dielectric layers.
drawing_layers	All drawing layers.
drawings	All stackup layers.
layers	Refresh all layers in the current stackup.
oeditor	Editor.
signals	All signal layers.
stackup_layers	All stackup layers.
zones	List of all available zones.

pyaedt.modules.LayerStackup.Layer

```
class pyaedt.modules.LayerStackup.Layer(app, layertype='signal', negative=False)
```

Manages the stackup layer for the Circuit and HFSS 3D Layout tools.

Parameters

app	[<i>pyaedt.modules.LayerStackup.Layers</i>]
layertype	[<i>str</i> , optional] The default is "signal".
negative	[<i>bool</i> , optional] Whether the geometry on the layer is cut away from the layer. The default is False.

Examples

```
>>> from pyaedt import Hfss3dLayout
>>> app = Hfss3dLayout()
>>> layers = app.modeler.layers["Top"]
```

```
__init__(app, layertype='signal', negative=False)
```

Methods

create_stackup_layer()	Create a stackup layer.
remove_stackup_layer()	Remove the stackup layer.
set_layer_color(r, g, b)	Update the color of the layer.
update_stackup_layer()	Update the stackup layer.

Attributes

bottom_huray_ratio	Get/Set the active layer bottom roughness ratio.
bottom_nodule_radius	Get/Set the active layer bottom roughness radius.
bottom_roughness	Return or set the active layer bottom roughness (with units).
bottom_roughness_model	Get/Set the active layer bottom roughness model.
color	Return or set the property of the active layer.
draw_override	Get/Set the active layer draw override value.
etch	Get/Set the active layer etch value.
fill_material	Get/Set the active layer filling material.
hfss_solver_settings	Get/Set the active layer hfss solver settings.
index	Get/Set the active layer index.
is_mesh_background	Get/Set the active layer mesh backgraound.
is_mesh_overlay	Get/Set the active layer mesh overlay.
is_negative	Get/Set the active layer negative flag.
is_visible	Get/Set the active layer visibility.
is_visible_component	Get/Set the active layer component visibility.
is_visible_hole	Get/Set the active layer hole visibility.
is_visible_pad	Get/Set the active layer pad visibility.
is_visible_path	Get/Set the active layer paths visibility.
is_visible_shape	Get/Set the active layer shape visibility.
locked	Get/Set the active layer lock flag.
lower_elevation	Get/Set the active layer lower elevation.
material	Get/Set the active layer material name.
oeditor	Oeditor Module.
pattern	Get/Set the active layer pattern.
planar_em_solver_settings	Get/Set the active layer PlanarEm solver settings.
roughness	Return or set the active layer roughness (with units).
side_huray_ratio	Get/Set the active layer bottom roughness ratio.
side_model	Get/Set the active layer side roughness model.
side_nodule_radius	Get/Set the active layer side roughness radius.
side_roughness	Get/Set the active layer side roughness (with units).
thickness	Get/Set the active layer thickness value.
thickness_units	Get the active layer thickness units value.
top_bottom	Get/Set the active layer top bottom alignment.
top_huray_ratio	Get/Set the active layer top roughness ratio.
top_nodule_radius	Get/Set the active layer top roughness radius.
top_roughness	Get/Set the active layer top roughness (with units).
top_roughness_model	Get/Set the active layer top roughness model.
transparency	Get/Set the property to the active layer.
upper_elevation	Get the active layer upper elevation value with units.
use_etch	Get/Set the active layer etching flag.
user	Get/Set the active layer user flag.
usp	Get/Set the active layer usp flag.
visflag	Visibility flag for objects on the layer.
zones	Get/Set the active layer zones.

2.3 3D modeler

This section lists the core AEDT Modeler modules with 3D solvers (HFSS, Maxwell, Icepak, Q3D, and Mechanical):

- Modeler
- Primitives
- Objects

They are accessible through the `modeler` property:

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call return the Modeler3D class
modeler = app.modeler

# This call returns a Primitives3D object
primitives = modeler

# This call return an Object3d object
my_box = primitives.create_box([0,0,0],[10,10,10])
my_box = primitives.objects[my_box.id]

# This call return a FacePrimitive object list
my_box.faces
# This call returns an EdgePrimitive object list
my_box.edges
my_box.faces[0].edges

# This call returns a VertexPrimitive object list
my_box.vertices
my_box.faces[0].vertices
my_box.faces[0].edges[0].vertices

...
```

2.3.1 Modeler

The Modeler module contains all properties and methods needed to edit a modeler, including all primitives methods and properties for HFSS, Maxwell 3D, Q3D Extractor, and Icepak:

<code>modeler3d.Modeler3D</code>	Provides the Modeler 3D application interface.
----------------------------------	--

pyaedt.modeler.modeler3d.Modeler3D

class `pyaedt.modeler.modeler3d.Modeler3D(application)`

Provides the Modeler 3D application interface.

This class is inherited in the caller application and is accessible through the modeler variable object. For example, `hfss.modeler`.

Parameters

`application`

`[pyaedt.application.Analysis3D.FieldAnalysis3D]`

Examples

```
>>> from pyaedt import Hfss  
>>> hfss = Hfss()  
>>> my_modeler = hfss.modeler
```

`__init__(application)`

Methods

<code>add_bird(input_dir[, speed, global_offset, ...])</code>	Add a Bird Multipart from 3D Components.
<code>add_environment(input_dir[, global_offset, ...])</code>	Add an Environment Multipart Component from JSON file.
<code>add_new_objects()</code>	Add objects that have been created in the modeler by previous operations.
<code>add_new_points()</code>	Add objects that have been created in the modeler by previous operations.
<code>add_new_solids()</code>	Add objects that have been created in the modeler by previous operations.
<code>add_new_user_defined_component()</code>	Add 3D components and user-defined models that have been created in the modeler by previous operations.
<code>add_person(input_dir[, speed, ...])</code>	Add a Walking Person Multipart from 3D Components.
<code>add_vehicle(input_dir[, speed, ...])</code>	Add a Moving Vehicle Multipart from 3D Components.
<code>automatic_thicken_sheets(assignment, value)</code>	Create thickened sheets for a list of input faces.
<code>break_spaceclaim_connection()</code>	Disconnect from the running SpaceClaim instance.
<code>change_region_coordinate_system([region_cs, ...])</code>	Change region coordinate system.
<code>change_region_padding(padding_data, padding_type)</code>	Change region padding settings.
<code>chassis_subtraction(chassis_part)</code>	Subtract all non-vacuum objects from the main chassis object.
<code>check_choke_values(input_dir[, ...])</code>	Verify the values in the json file and create another one with corrected values next to the first one.
<code>check_plane(assignment, face_location[, offset])</code>	Check for the plane that is defined as the face for an object.

continues on next page

Table 28 – continued from previous page

<code>clean_objects_name(main_part_name)</code>	Clean the names of the objects for a main part.
<code>cleanup_objects()</code>	Clean up objects that no longer exist in the modeler because they were removed by previous operations.
<code>cleanup_points()</code>	Clean up points that no longer exist in the modeler because they were removed by previous operations.
<code>cleanup_solids()</code>	Clean up solids that no longer exist in the modeler because they were removed by previous operations.
<code>clone(assignment)</code>	Clone objects from a list of object IDs.
<code>connect(assignment)</code>	Connect objects from a list.
<code>convert_segments_to_line(assignment)</code>	Convert a CreatePolyline list of segments to lines.
<code>convert_to_selections(assignment[, turn_list])</code>	Convert modeler objects.
<code>copy(assignment)</code>	Copy objects to the clipboard.
<code>cover_faces(assignment)</code>	Cover a face.
<code>cover_lines(assignment)</code>	Cover closed lines and transform them to a sheet.
<code>create_3dcomponent(component_file[, ...])</code>	Create a 3D component file.
<code>create_air_region([x_pos, y_pos, z_pos, ...])</code>	Create an air region.
<code>create_airbox([offset, offset_type, name])</code>	Create an airbox that is as big as the bounding extension of the project.
<code>create_bondwire(start, end[, h1, h2, alpha, ...])</code>	Create a bondwire.
<code>create_box(origin, sizes[, name, material])</code>	Create a box.
<code>create_choke(input_file)</code>	Create a choke from a JSON setting file.
<code>create_circle(orientation, origin, radius[, ...])</code>	Create a circle.
<code>create_coaxial(startingposition, axis[, ...])</code>	Create a coaxial.
<code>create_cone(orientation, origin, ...[, ...])</code>	Create a cone.
<code>create_coordinate_system([origin, ...])</code>	Create a coordinate system.
<code>create_cylinder(orientation, origin, radius, ...)</code>	Create a cylinder.
<code>create_ellipse(orientation, origin, ...[, ...])</code>	Create an ellipse.
<code>create_equationbased_curve([x_t, y_t, z_t, ...])</code>	Create an equation-based curve.
<code>create_face_coordinate_system(face, origin, ...)</code>	Create a face coordinate system.
<code>create_face_list(assignment[, name])</code>	Create a list of faces given a list of face ID or a list of objects.
<code>create_faceted_bondwire_from_true_surface([face, ...])</code>	Create a faceted bondwire from an existing true surface bondwire.
<code>create_group([objects, components, groups, ...])</code>	Group objects or groups into one group.
<code>create_helix(assignment, origin, ...[, ...])</code>	Create an helix from a polyline.
<code>create_object_coordinate_system(assignment, ...)</code>	Create an object coordinate system.
<code>create_object_from_edge(assignment[, non_model])</code>	Create an object from one or multiple edges.
<code>create_object_from_face(assignment[, non_model])</code>	Create an object from one or multiple face.
<code>create_object_list(assignment[, name])</code>	Create an object list given a list of object names.
<code>create_outer_facelist(assignment[, name])</code>	Create a face list from a list of outer objects.
<code>create_plane([name, plane_base_x, ...])</code>	Create a plane.
<code>create_point(position[, name, color])</code>	Create a point.
<code>create_polyhedron([orientation, center, ...])</code>	Create a regular polyhedron.
<code>create_polyline(points[, segment_type, ...])</code>	Draw a polyline object in the 3D modeler.
<code>create_rectangle(orientation, origin, sizes)</code>	Create a rectangle.
<code>create_region([pad_value, pad_type, name])</code>	Create an air region.
<code>create_sheet_to_ground(assignment[, ...])</code>	Create a sheet between an object and a ground plane.

continues on next page

Table 28 – continued from previous page

<code>create_sphere(origin, radius[, name, material])</code>	Create a sphere.
<code>create_spiral([internal_radius, spacing, ...])</code>	Create a spiral inductor from a polyline.
<code>create_spiral_on_face(assignment, width[, ...])</code>	Create a Spiral Polyline inside a face.
<code>create_subregion(padding_values, ...[, name])</code>	Create a subregion.
<code>create_torus(origin, major_radius, minor_radius)</code>	Create a torus.
<code>create_udm(udm_full_name, parameters[, ...])</code>	Create a user-defined model.
<code>create_udp(dll, parameters[, library, name])</code>	Create a user-defined primitive (UDP).
<code>create_waveguide(origin, wg_direction_axis)</code>	Create a standard waveguide and optionally parametrize W and H .
<code>delete([assignment])</code>	Delete objects or groups.
<code>delete_objects_containing(contained_string)</code>	Delete all objects with a given prefix.
<code>detach_faces(assignment, faces)</code>	Section the object.
<code>does_object_exists(assignment)</code>	Check to see if an object exists.
<code>duplicate_along_line(assignment, vector[, ...])</code>	Duplicate a selection along a line.
<code>duplicate_and_mirror(assignment, origin, vector)</code>	Duplicate and mirror a selection.
<code>duplicate_around_axis(assignment, axis[, ...])</code>	Duplicate a selection around an axis.
<code>duplicate_coordinate_system_to_global(...)</code>	Create a duplicate coordinate system referenced to the global coordinate system.
<code>edit_region_dimensions(values)</code>	Modify the dimensions of the region.
<code>explicitly_subtract(tool_parts, blank_parts)</code>	Explicitly subtract all elements in a SolveInside list and a SolveSurface list.
<code>find_closest_edges(start_object, end_object)</code>	Retrieve the two closest edges that are not perpendicular for two objects.
<code>find_new_objects()</code>	Find any new objects in the modeler that were created by previous operations.
<code>find_point_around(assignment, origin, ...)</code>	Find the point around an object.
<code>find_port_faces(assignment)</code>	Find the vacuums given a list of input sheets.
<code>fit_all()</code>	Fit all.
<code>flatten_assembly()</code>	Flatten the assembly, removing all group trees.
<code>generate_object_history(assignment)</code>	Generate history for the object.
<code>get_3d_component_object_list(name)</code>	Retrieve all objects belonging to a 3D component.
<code>get_bodynames_from_position(position[, ...])</code>	Retrieve the names of the objects that are in contact with a given point.
<code>get_boundaries_name()</code>	Retrieve all boundary names.
<code>get_bounding_dimension()</code>	Retrieve the dimension array of the bounding box.
<code>get_closest_edgeid_to_position(position[, units])</code>	Get the edge ID closest to a given position.
<code>get_edge_length(assignment)</code>	Get the length of an edge.
<code>get_edge_midpoint(assignment)</code>	Retrieve the midpoint coordinates of a given edge ID or edge name.
<code>get_edge_vertices(assignment)</code>	Retrieve the vertex IDs of a given edge ID or edge name.
<code>get_edgeid_from_position(position[, ...])</code>	Get an edge ID from a position.
<code>get_edgeids_from_vertexid(vertex, assignment)</code>	Retrieve edge IDs for a vertex ID.
<code>get_edges_for_circuit_port(assignment[, ...])</code>	Retrieve two edge IDs suitable for the circuit port.
<code>get_edges_for_circuit_port_from_sheet(assig</code>	Retrieve two edge IDs that are suitable for a circuit port from a sheet.
<code>get_edges_on_bounding_box(assignment[, ...])</code>	Retrieve the edges of the sheets passed in the input that are lying on the bounding box.
<code>get_entitylist_id(name)</code>	Retrieve the ID of an entity list.

continues on next page

Table 28 – continued from previous page

<code>get_equivalent_parallel_edges(assignment[, ...])</code>	Create two new edges that are parallel and equal to the smallest edge given a parallel couple of edges.
<code>get_existing_polyline(assignment)</code>	Retrieve a polyline object to manipulate it.
<code>get_face_area(assignment)</code>	Retrieve the area of a given face ID.
<code>get_face_by_id(assignment)</code>	Get the face object given its ID.
<code>get_face_center(assignment)</code>	Retrieve the center position for a given planar face ID.
<code>get_face_edges(assignment)</code>	Retrieve the edge IDs of a given face name or face ID.
<code>get_face_vertices(assignment)</code>	Retrieve the vertex IDs of a given face ID or face name.
<code>get_faceid_from_position(position[, ...])</code>	Retrieve a face ID from a position.
<code>get_faces_from_materials(filter_materials)</code>	Select all outer faces given a list of materials.
<code>get_group_bounding_box(group)</code>	Retrieve the bounding box of a group.
<code>get_line_ids()</code>	Create a dictionary of object IDs for the lines in the design with the line name as the key.
<code>get_matched_object_name(search_string)</code>	Retrieve the name of the matched object.
<code>get_mid_points_on_dir(assignment, axis)</code>	Retrieve midpoints on a given axis direction.
<code>get_model_bounding_box()</code>	Retrieve the model bounding box.
<code>get_obj_id(assignment)</code>	Return the object ID from an object name.
<code>get_object_edges(assignment)</code>	Retrieve the edge IDs of a given object ID or object name.
<code>get_object_faces(assignment)</code>	Retrieve the face IDs of a given object ID or object name.
<code>get_object_from_name(assignment)</code>	Return the object from an object name.
<code>get_object_name_from_edge_id(assignment)</code>	Retrieve the object name for a predefined edge ID.
<code>get_object_vertices(assignment)</code>	Retrieve the vertex IDs of a given object name or object ID.
<code>get_objects_by_material([material])</code>	Get a list of objects either of a specified material or classified by material.
<code>get_objects_in_group(group)</code>	Retrieve a list of objects belonging to a group.
<code>get_objects_w_string(string_name[, ...])</code>	Retrieve all objects with a given string in their names.
<code>get_solving_volume()</code>	Generate a mesh for a setup.
<code>get_vertex_position(assignment)</code>	Retrieve a vector of vertex coordinates.
<code>get_vertices_of_line(assignment)</code>	Generate a list of vertex positions for a line object from AEDT in model units.
<code>global_to_cs(point, coordinate_system)</code>	Transform a point from the global coordinate system to another coordinate system.
<code>heal_objects(assignment[, auto_heal, ...])</code>	Repair invalid geometry entities for the selected objects within the specified tolerance settings.
<code>import_3d_cad(input_file[, healing, ...])</code>	Import a CAD model.
<code>import_from_openstreet_map(latitude_longitude)</code>	Import OpenStreet Maps into AEDT.
<code>import_nastran(file_path[, import_lines, ...])</code>	Import Nastran file into 3D Modeler by converting the faces to stl and reading it.
<code>import_primitives_from_file([input_file, ...])</code>	Import and create primitives from a JSON file or dictionary of properties.
<code>import_spaceclaim_document(input_file)</code>	Import a SpaceClaim document.
<code>imprint(blank_list, tool_list[, keep_originals])</code>	Imprint an object list on another object list.
<code>imprint_normal_projection(assignment[, ...])</code>	Imprint the normal projection of objects over a sheet.
<code>imprint_vector_projection(assignment, ..., ...)</code>	Imprint the projection of objects over a sheet with a specified vector and distance.
<code>insert_3d_component(input_file[, ...])</code>	Insert a new 3D component.
<code>insert_layout_component(input_file[, ...])</code>	Insert a new layout component.
<code>intersect(assignment[, keep_originals])</code>	Intersect objects from a list.

continues on next page

Table 28 – continued from previous page

<code>invert_cs(coordinate_system[, to_global])</code>	Get the inverse translation and the conjugate quaternion of the input coordinate system.
<code>load_scdm_in_hfss(input_file)</code>	Load a SpaceClaim file in HFSS.
<code>mirror(assignment, origin, vector[, ...])</code>	Mirror a selection.
<code>modeler_variable(value)</code>	Modeler variable.
<code>move(assignment, vector)</code>	Move objects from a list.
<code>move_edge(assignment[, offset])</code>	Move an input edge or a list of input edges of a specific object.
<code>move_face(assignment[, offset])</code>	Move an input face or a list of input faces of a specific object.
<code>objects_in_bounding_box(bounding_box[, ...])</code>	Given a bounding box checks if objects, sheets and lines are inside it.
<code>objects_segmentation(objects_list[, ...])</code>	Get segmentation of an object given the segmentation thickness or number of segments.
<code>paste()</code>	Paste objects from the clipboard.
<code>polyline_segment(type[, num_seg, ...])</code>	New segment of a polyline.
<code>purge_history(assignment)</code>	Purge history objects from object names.
<code>reassign_subregion(region, parts)</code>	Modify parts in the subregion.
<code>reference_cs_to_global(coordinate_system)</code>	Get the origin and quaternion defining the coordinate system in the global coordinates.
<code>refresh()</code>	Refresh this object.
<code>refresh_all_ids()</code>	Refresh all IDs.
<code>replace_3dcomponent([component_name, ...])</code>	Replace with 3D component.
<code>rotate(assignment, axis[, angle, units])</code>	Rotate the selection.
<code>scale(assignment[, x, y, z])</code>	Scale a list of objects.
<code>section(assignment, plane[, create_new, ...])</code>	Section the selection.
<code>select_allfaces_fromobjects(assignment)</code>	Select all outer faces given a list of objects.
<code>separate_bodies(assignment[, create_group])</code>	Separate bodies of the selection.
<code>set_object_model_state(assignment[, model])</code>	Set a list of objects to either models or non-models.
<code>set_objects_deformation(assignment)</code>	Assign deformation objects to a Workbench link.
<code>set_objects_temperature(assignment[, ...])</code>	Assign temperatures to objects.
<code>set_working_coordinate_system(name)</code>	Set the working coordinate system to another coordinate system.
<code>setunassigned_mats()</code>	Find unassigned objects and set them to non-model.
<code>simplify_objects(assignment[, ...])</code>	Simplify command to converts complex objects into simpler primitives which are easy to mesh and solve.
<code>split(assignment[, plane, sides, tool, ...])</code>	Split a list of objects.
<code>subtract(blank_list, tool_list[, keep_originals])</code>	Subtract objects.
<code>sweep_along_normal(assignment, faces[, ...])</code>	Sweep the selection along the vector.
<code>sweep_along_path(assignment, sweep_object[, ...])</code>	Sweep the selection along a path.
<code>sweep_along_vector(assignment, sweep_vector)</code>	Sweep the selection along a vector.
<code>sweep_around_axis(assignment, axis[, ...])</code>	Sweep the selection around the axis.
<code>thicken_sheet(assignment, thickness[, ...])</code>	Thicken the sheet of the selection.
<code>ungroup(groups)</code>	Ungroup one or more groups.
<code>unite(assignment[, purge, keep_originals])</code>	Unite objects from a list.
<code>update_object(assignment)</code>	Update any <code>pyaedt.modeler.Object3d</code> . <code>Object3d</code> derivatives that have potentially been modified by a modeler operation.
<code>update_udp(assignment, operation, parameters)</code>	Update an existing geometrical object that was originally created using a user-defined primitive (UDP).

continues on next page

Table 28 – continued from previous page

<code>value_in_object_units(value)</code>	Convert one or more strings for numerical lengths to floating point values.
<code>vertex_data_of_lines([text_filter])</code>	Generate a dictionary of line vertex data for all lines contained within the design.
<code>wrap_sheet(sheet, object[, imprinted])</code>	Execute the sheet wrapping around an object.

Attributes

<code>coordinate_systems</code>	Coordinate systems.
<code>defaultmaterial</code>	Default material.
<code>design_type</code>	Design type.
<code>dimension</code>	Dimensions.
<code>geometry_mode</code>	Geometry mode.
<code>layout_component_names</code>	List of the names of all Layout component objects.
<code>line_names</code>	List of the names of all line objects.
<code>line_objects</code>	List of all line objects.
<code>logger</code>	Logger.
<code>materials</code>	Material library used in the project.
<code>model_consistency_report</code>	Summary of detected inconsistencies between the AEDT modeler and PyAEDT structures.
<code>model_objects</code>	List of the names of all model objects.
<code>model_units</code>	Model units as a string.
<code>non_model_objects</code>	List of objects of all non-model objects.
<code>object_list</code>	List of all objects.
<code>object_names</code>	List of the names of all objects.
<code>objects_by_name</code>	Object dictionary organized by name.
<code>obounding_box</code>	Bounding box.
<code>oeditor</code>	AEDT oEditor module.
<code>planes</code>	Planes.
<code>point_names</code>	List of the names of all points.
<code>point_objects</code>	List of points objects.
<code>projdir</code>	Project directory.
<code>selections</code>	Selections.
<code>sheet_names</code>	List of the names of all sheet objects.
<code>sheet_objects</code>	List of all sheet objects.
<code>solid_bodies</code>	List of object names.
<code>solid_names</code>	List of the names of all solid objects.
<code>solid_objects</code>	List of all solid objects.
<code>unclassified_names</code>	List of the names of all unclassified objects.
<code>unclassified_objects</code>	List of all unclassified objects.
<code>user_defined_component_names</code>	List of the names of all 3D component objects.
<code>user_lists</code>	User lists.
<code>version</code>	Version.

```
from pyaedt import Circuit
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)
```

(continues on next page)

(continued from previous page)

```
# This call returns the NexximComponents class
origin = [0,0,0]
sizes = [10,5,20]
#Material and name are not mandatory fields
box_object = app.modeler.primivites.create_box(origin, sizes, name="mybox", material=
    ↪"copper")

...
```

2.4 2D modeler

This section lists the core AEDT Modeler modules for 2D and 3D solvers (Maxwell 2D, 2D Extractor).

They are accessible through the `modeler` property:

```
from pyaedt import Maxwell2d
app = Maxwell2d(specified_version="2023.1",
                 non_graphical=False, new_desktop_session=True,
                 close_on_exit=True, student_version=False)

# This call return the Modeler2D class
modeler = app.modeler

...
```

The `Modeler` module contains all properties and methods needed to edit a modeler, including all primitives methods and properties:

<code>modeler2d.Modeler2D</code>	Provides the Modeler 2D application interface.
----------------------------------	--

2.4.1 pyaedt.modeler.modeler2d.Modeler2D

```
class pyaedt.modeler.modeler2d.Modeler2D(application)
```

Provides the Modeler 2D application interface.

This class is inherited in the caller application and is accessible through the `modeler` variable object(eg. `maxwell2d.modeler`).

Parameters

`application`

[`pyaedt.application.Analysis2D.FieldAnalysis2D`]

Examples

```
>>> from pyaedt import Maxwell2d
>>> app = Maxwell2d()
>>> my_modeler = app.modeler
```

`__init__(application)`

Methods

<code>add_new_objects()</code>	Add objects that have been created in the modeler by previous operations.
<code>add_new_points()</code>	Add objects that have been created in the modeler by previous operations.
<code>add_new_solids()</code>	Add objects that have been created in the modeler by previous operations.
<code>add_new_user_defined_component()</code>	Add 3D components and user-defined models that have been created in the modeler by previous operations.
<code>automatic_thicken_sheets(assignment, value)</code>	Create thickened sheets for a list of input faces.
<code>break_spaceclaim_connection()</code>	Disconnect from the running SpaceClaim instance.
<code>calculate_radius_2D(assignment[, inner])</code>	Calculate the extremity of an object in the radial direction.
<code>chassis_subtraction(chassis_part)</code>	Subtract all non-vacuum objects from the main chassis object.
<code>check_plane(assignment, face_location[, offset])</code>	Check for the plane that is defined as the face for an object.
<code>clean_objects_name(main_part_name)</code>	Clean the names of the objects for a main part.
<code>cleanup_objects()</code>	Clean up objects that no longer exist in the modeler because they were removed by previous operations.
<code>cleanup_points()</code>	Clean up points that no longer exist in the modeler because they were removed by previous operations.
<code>cleanup_solids()</code>	Clean up solids that no longer exist in the modeler because they were removed by previous operations.
<code>clone(assignment)</code>	Clone objects from a list of object IDs.
<code>connect(assignment)</code>	Connect objects from a list.
<code>convert_segments_to_line(assignment)</code>	Convert a CreatePolyline list of segments to lines.
<code>convert_to_selections(assignment[, turn_list])</code>	Convert modeler objects.
<code>copy(assignment)</code>	Copy objects to the clipboard.
<code>cover_faces(assignment)</code>	Cover a face.
<code>cover_lines(assignment)</code>	Cover closed lines and transform them to a sheet.
<code>create_air_region([x_pos, y_pos, z_pos, ...])</code>	Create an air region.
<code>create_airbox([offset, offset_type, name])</code>	Create an airbox that is as big as the bounding extension of the project.
<code>create_circle(origin, radius[, num_sides, ...])</code>	Create a circle.
<code>create_coordinate_system([origin, ...])</code>	Create a coordinate system.
<code>create_ellipse(origin, major_radius, ratio)</code>	Create an ellipse.
<code>create_face_coordinate_system(face, origin, ...)</code>	Create a face coordinate system.

continues on next page

Table 30 – continued from previous page

<code>create_face_list(assignment[, name])</code>	Create a list of faces given a list of face ID or a list of objects.
<code>create_faceted_bondwire_from_true_surface([name, bondwire_type])</code>	Create a faceted bondwire from an existing true surface bondwire.
<code>create_group([objects, components, groups, ...])</code>	Group objects or groups into one group.
<code>create_object_coordinate_system(assignment, ...)</code>	Create an object coordinate system.
<code>create_object_from_edge(assignment[, non_model])</code>	Create an object from one or multiple edges.
<code>create_object_from_face(assignment[, non_model])</code>	Create an object from one or multiple face.
<code>create_object_list(assignment[, name])</code>	Create an object list given a list of object names.
<code>create_outer_facelist(assignment[, name])</code>	Create a face list from a list of outer objects.
<code>create_plane([name, plane_base_x, ...])</code>	Create a plane.
<code>create_point(position[, name, color])</code>	Create a point.
<code>create_polyline(points[, segment_type, ...])</code>	Draw a polyline object in the 3D modeler.
<code>create_rectangle(origin, sizes[, ...])</code>	Create a rectangle.
<code>create_region([pad_value, pad_type, name])</code>	Create an air region.
<code>create_regular_polygon(origin, start_point)</code>	Create a rectangle.
<code>create_sheet_to_ground(assignment[, ...])</code>	Create a sheet between an object and a ground plane.
<code>create_spiral_on_face(assignment, width[, ...])</code>	Create a Spiral Polyline inside a face.
<code>create_subregion(padding_values, ...[, name])</code>	Create a subregion.
<code>create_udp(dll, parameters[, library, name])</code>	Create a user-defined primitive (UDP).
<code>delete([assignment])</code>	Delete objects or groups.
<code>delete_objects_containing(contained_string)</code>	Delete all objects with a given prefix.
<code>detach_faces(assignment, faces)</code>	Section the object.
<code>does_object_exists(assignment)</code>	Check to see if an object exists.
<code>duplicate_along_line(assignment, vector[, ...])</code>	Duplicate a selection along a line.
<code>duplicate_and_mirror(assignment, origin, vector)</code>	Duplicate and mirror a selection.
<code>duplicate_around_axis(assignment, axis[, ...])</code>	Duplicate a selection around an axis.
<code>duplicate_coordinate_system_to_global(...)</code>	Create a duplicate coordinate system referenced to the global coordinate system.
<code>edit_region_dimensions(values)</code>	Modify the dimensions of the region.
<code>explicitly_subtract(tool_parts, blank_parts)</code>	Explicitly subtract all elements in a SolveInside list and a SolveSurface list.
<code>find_closest_edges(start_object, end_object)</code>	Retrieve the two closest edges that are not perpendicular for two objects.
<code>find_new_objects()</code>	Find any new objects in the modeler that were created by previous operations.
<code>find_point_around(assignment, origin, ...)</code>	Find the point around an object.
<code>find_port_faces(assignment)</code>	Find the vacuums given a list of input sheets.
<code>fit_all()</code>	Fit all.
<code>flatten_assembly()</code>	Flatten the assembly, removing all group trees.
<code>generate_object_history(assignment)</code>	Generate history for the object.
<code>get_bodynames_from_position(position[, ...])</code>	Retrieve the names of the objects that are in contact with a given point.
<code>get_boundaries_name()</code>	Retrieve all boundary names.
<code>get_bounding_dimension()</code>	Retrieve the dimension array of the bounding box.
<code>get_closest_edgeid_to_position(position[, units])</code>	Get the edge ID closest to a given position.
<code>get_edge_length(assignment)</code>	Get the length of an edge.

continues on next page

Table 30 – continued from previous page

<code>get_edge_midpoint(assignment)</code>	Retrieve the midpoint coordinates of a given edge ID or edge name.
<code>get_edge_vertices(assignment)</code>	Retrieve the vertex IDs of a given edge ID or edge name.
<code>get_edgeid_from_position(position[, ...])</code>	Get an edge ID from a position.
<code>get_edgeids_from_vertexid(vertex, assignment)</code>	Retrieve edge IDs for a vertex ID.
<code>get_edges_for_circuit_port(assignment[, ...])</code>	Retrieve two edge IDs suitable for the circuit port.
<code>get_edges_for_circuit_port_from_sheet(assignment[, ...])</code>	Retrieve two edge IDs that are suitable for a circuit port from a sheet.
<code>get_edges_on_bounding_box(assignment[, ...])</code>	Retrieve the edges of the sheets passed in the input that are lying on the bounding box.
<code>get_entitylist_id(name)</code>	Retrieve the ID of an entity list.
<code>get_equivalent_parallel_edges(assignment[, ...])</code>	Create two new edges that are parallel and equal to the smallest edge given a parallel couple of edges.
<code>get_existing_polyline(assignment)</code>	Retrieve a polyline object to manipulate it.
<code>get_face_area(assignment)</code>	Retrieve the area of a given face ID.
<code>get_face_by_id(assignment)</code>	Get the face object given its ID.
<code>get_face_center(assignment)</code>	Retrieve the center position for a given planar face ID.
<code>get_face_edges(assignment)</code>	Retrieve the edge IDs of a given face name or face ID.
<code>get_face_vertices(assignment)</code>	Retrieve the vertex IDs of a given face ID or face name.
<code>get_faceid_from_position(position[, ...])</code>	Retrieve a face ID from a position.
<code>get_faces_from_materials(filter_materials)</code>	Select all outer faces given a list of materials.
<code>get_group_bounding_box(group)</code>	Retrieve the bounding box of a group.
<code>get_line_ids()</code>	Create a dictionary of object IDs for the lines in the design with the line name as the key.
<code>get_matched_object_name(search_string)</code>	Retrieve the name of the matched object.
<code>get_mid_points_on_dir(assignment, axis)</code>	Retrieve midpoints on a given axis direction.
<code>get_model_bounding_box()</code>	Retrieve the model bounding box.
<code>get_obj_id(assignment)</code>	Return the object ID from an object name.
<code>get_object_edges(assignment)</code>	Retrieve the edge IDs of a given object ID or object name.
<code>get_object_faces(assignment)</code>	Retrieve the face IDs of a given object ID or object name.
<code>get_object_from_name(assignment)</code>	Return the object from an object name.
<code>get_object_name_from_edge_id(assignment)</code>	Retrieve the object name for a predefined edge ID.
<code>get_object_vertices(assignment)</code>	Retrieve the vertex IDs of a given object name or object ID.
<code>get_objects_by_material([material])</code>	Get a list of objects either of a specified material or classified by material.
<code>get_objects_in_group(group)</code>	Retrieve a list of objects belonging to a group.
<code>get_objects_w_string(string_name[, ...])</code>	Retrieve all objects with a given string in their names.
<code>get_solving_volume()</code>	Generate a mesh for a setup.
<code>get_vertex_position(assignment)</code>	Retrieve a vector of vertex coordinates.
<code>get_vertices_of_line(assignment)</code>	Generate a list of vertex positions for a line object from AEDT in model units.
<code>global_to_cs(point, coordinate_system)</code>	Transform a point from the global coordinate system to another coordinate system.
<code>heal_objects(assignment[, auto_heal, ...])</code>	Repair invalid geometry entities for the selected objects within the specified tolerance settings.
<code>import_3d_cad(input_file[, healing, ...])</code>	Import a CAD model.

continues on next page

Table 30 – continued from previous page

<code>import_primitives_from_file([input_file, ...])</code>	Import and create primitives from a JSON file or dictionary of properties.
<code>import_spaceclaim_document(input_file)</code>	Import a SpaceClaim document.
<code>imprint(blank_list, tool_list[, keep_originals])</code>	Imprint an object list on another object list.
<code>imprint_normal_projection(assignment[, ...])</code>	Imprint the normal projection of objects over a sheet.
<code>imprint_vector_projection(assignment, ...[, ...])</code>	Imprint the projection of objects over a sheet with a specified vector and distance.
<code>intersect(assignment[, keep_originals])</code>	Intersect objects from a list.
<code>invert_cs(coordinate_system[, to_global])</code>	Get the inverse translation and the conjugate quaternion of the input coordinate system.
<code>load_scdm_in_hfss(input_file)</code>	Load a SpaceClaim file in HFSS.
<code>mirror(assignment, origin, vector[, ...])</code>	Mirror a selection.
<code>modeler_variable(value)</code>	Modeler variable.
<code>move(assignment, vector)</code>	Move objects from a list.
<code>move_edge(assignment[, offset])</code>	Move an input edge or a list of input edges of a specific object.
<code>move_face(assignment[, offset])</code>	Move an input face or a list of input faces of a specific object.
<code>objects_in_bounding_box(bounding_box[, ...])</code>	Given a 2D bounding box, check if sheets and lines are inside it.
<code>paste()</code>	Paste objects from the clipboard.
<code>polyline_segment(type[, num_seg, ...])</code>	New segment of a polyline.
<code>purge_history(assignment)</code>	Purge history objects from object names.
<code>radial_split_2D(radius, name)</code>	Split the stator and rotor for mesh refinement.
<code>reassign_subregion(region, parts)</code>	Modify parts in the subregion.
<code>reference_cs_to_global(coordinate_system)</code>	Get the origin and quaternion defining the coordinate system in the global coordinates.
<code>refresh()</code>	Refresh this object.
<code>refresh_all_ids()</code>	Refresh all IDs.
<code>rotate(assignment, axis[, angle, units])</code>	Rotate the selection.
<code>scale(assignment[, x, y, z])</code>	Scale a list of objects.
<code>section(assignment, plane[, create_new, ...])</code>	Section the selection.
<code>select_allfaces_fromobjects(assignment)</code>	Select all outer faces given a list of objects.
<code>separate_bodies(assignment[, create_group])</code>	Separate bodies of the selection.
<code>set_object_model_state(assignment[, model])</code>	Set a list of objects to either models or non-models.
<code>set_objects_deformation(assignment)</code>	Assign deformation objects to a Workbench link.
<code>set_objects_temperature(assignment[, ...])</code>	Assign temperatures to objects.
<code>set_working_coordinate_system(name)</code>	Set the working coordinate system to another coordinate system.
<code>setunassigned_mats()</code>	Find unassigned objects and set them to non-model.
<code>simplify_objects(assignment[, ...])</code>	Simplify command to converts complex objects into simpler primitives which are easy to mesh and solve.
<code>split(assignment[, plane, sides, tool, ...])</code>	Split a list of objects.
<code>subtract(blank_list, tool_list[, keep_originals])</code>	Subtract objects.
<code>sweep_along_normal(assignment, faces[, ...])</code>	Sweep the selection along the vector.
<code>sweep_along_path(assignment, sweep_object[, ...])</code>	Sweep the selection along a path.
<code>sweep_along_vector(assignment, sweep_vector)</code>	Sweep the selection along a vector.
<code>sweep_around_axis(assignment, axis[, ...])</code>	Sweep the selection around the axis.
<code>thicken_sheet(assignment, thickness[, ...])</code>	Thicken the sheet of the selection.
<code>ungroup(groups)</code>	Ungroup one or more groups.
<code>unite(assignment[, purge, keep_originals])</code>	Unite objects from a list.

continues on next page

Table 30 – continued from previous page

<code>update_object(assignment)</code>	Update any <code>pyaedt.modeler.Object3d</code> . <code>Object3d</code> derivatives that have potentially been modified by a modeler operation.
<code>update_udp(assignment, operation, parameters)</code>	Update an existing geometrical object that was originally created using a user-defined primitive (UDP).
<code>value_in_object_units(value)</code>	Convert one or more strings for numerical lengths to floating point values.
<code>vertex_data_of_lines([text_filter])</code>	Generate a dictionary of line vertex data for all lines contained within the design.
<code>wrap_sheet(sheet, object[, imprinted])</code>	Execute the sheet wrapping around an object.

Attributes

<code>coordinate_systems</code>	Coordinate systems.
<code>defaultmaterial</code>	Default material.
<code>design_type</code>	Design type.
<code>dimension</code>	Dimensions.
<code>geometry_mode</code>	Geometry mode.
<code>layout_component_names</code>	List of the names of all Layout component objects.
<code>line_names</code>	List of the names of all line objects.
<code>line_objects</code>	List of all line objects.
<code>logger</code>	Logger.
<code>materials</code>	Material library used in the project.
<code>model_consistency_report</code>	Summary of detected inconsistencies between the AEDT modeler and PyAEDT structures.
<code>model_objects</code>	List of the names of all model objects.
<code>model_units</code>	Model units as a string.
<code>non_model_objects</code>	List of objects of all non-model objects.
<code>object_list</code>	List of all objects.
<code>object_names</code>	List of the names of all objects.
<code>objects_by_name</code>	Object dictionary organized by name.
<code>obounding_box</code>	Bounding box.
<code>oeditor</code>	AEDT <code>oEditor</code> module.
<code>plane2d</code>	Create a 2D plane.
<code>planes</code>	Planes.
<code>point_names</code>	List of the names of all points.
<code>point_objects</code>	List of points objects.
<code>projdir</code>	Project directory.
<code>selections</code>	Selections.
<code>sheet_names</code>	List of the names of all sheet objects.
<code>sheet_objects</code>	List of all sheet objects.
<code>solid_bodies</code>	List of object names.
<code>solid_names</code>	List of the names of all solid objects.
<code>solid_objects</code>	List of all solid objects.
<code>unclassified_names</code>	List of the names of all unclassified objects.
<code>unclassified_objects</code>	List of all unclassified objects.
<code>user_defined_component_names</code>	List of the names of all 3D component objects.
<code>user_lists</code>	User lists.
<code>version</code>	Version.

```
from pyaedt import Maxwell2d
app = Maxwell2d(specified_version="2023.1",
                 non_graphical=False, new_desktop_session=True,
                 close_on_exit=True, student_version=False)

# This call returns the NexximComponents class
origin = [0,0,0]
dimensions = [10,5,20]
#Material and name are not mandatory fields
box_object = app.modeler.primitives.create_rectangle([15, 20, 0], [5, 5], material=
    ↪"aluminum")

...
```

2.5 Primitives

This section lists the core AEDT Modeler primitives that are supported both in 2D and 3D solvers (HFSS, Maxwell, Icepak, Q3D, and Mechanical):

- Primitives
- Objects

They are accessible through the `modeler.objects` property:

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call return the Modeler3D class
modeler = app.modeler

# This call returns a Primitives3D object
primitives = modeler

# This call return an Object3d object
my_box = primitives.create_box([0,0,0],[10,10,10])
my_box = primitives.objects[my_box.id]

# This call return a FacePrimitive object list
my_box.faces
# This call returns an EdgePrimitive object list
my_box.edges
my_box.faces[0].edges

# This call returns a VertexPrimitive object list
my_box.vertices
my_box.faces[0].vertices
my_box.faces[0].edges[0].vertices

...
```

2.5.1 Objects

The following classes define objects properties for 3D and 2D Solvers (excluding HFSS 3D Layout). They contain all getters and setters to simplify object manipulation.

<code>cad.object3d.Object3d</code>	Manages object attributes for the AEDT 3D Modeler.
<code>cad.elements3d.FacePrimitive</code>	Contains the face object within the AEDT Desktop Modeler.
<code>cad.elements3d.EdgePrimitive</code>	Contains the edge object within the AEDT Desktop Modeler.
<code>cad.elements3d.VertexPrimitive</code>	Contains the vertex object within the AEDT Desktop Modeler.
<code>cad.polylines.PolylineSegment</code>	Creates and manipulates a segment of a polyline.
<code>cad.polylines.Polyline</code>	Creates and manipulates a polyline.
<code>cad.component_array.ComponentArray</code>	Manages object attributes for a 3D component array.
<code>cad.components_3d.UserDefinedComponent</code>	Manages object attributes for 3DComponent and User Defined Model.
<code>cad.elements3d.Point</code>	Manages point attributes for the AEDT 3D Modeler.
<code>cad.elements3d.Plane</code>	Manages plane attributes for the AEDT 3D Modeler.
<code>cad.elements3d.HistoryProps</code>	Manages an object's history properties.
<code>cad.elements3d.BinaryTreeNode</code>	Manages an object's history structure.

pyaedt.modeler.cad.object3d.Object3d

```
class pyaedt.modeler.cad.object3d.Object3d(primitives, name=None)
```

Manages object attributes for the AEDT 3D Modeler.

Parameters

primitives

[pyaedt.modeler.Primitives3D.Primitives3D] Inherited parent object.

name

[str]

Examples

Basic usage demonstrated with an HFSS design:

```
>>> from pyaedt import Hfss
>>> aedtapp = Hfss()
>>> prim = aedtapp.modeler
```

Create a part, such as box, to return an `pyaedt.modeler.Object3d.Object3d`.

```
>>> id = prim.create_box([0, 0, 0],[10, 10, 5],"Mybox","Copper")
>>> part = prim[id]
```

```
__init__(primitives, name=None)
```

Parameters

primitives

[pyaedt.modeler.Primitives3D.Primitives3D] Inherited parent object.

name
[str]

Methods

clone()	Clone the object and return the new 3D object.
delete()	Delete the object.
detach_faces(faces)	Section the object.
duplicate_along_line(vector[, clones, attach])	Duplicate the object along a line.
duplicate_around_axis(axis[, angle, clones, ...])	Duplicate the object around the axis.
edges_by_length(length[, length_filter, ...])	Filter edges by length.
export_image([output_file])	Export the current object to a specified file path.
faces_by_area(area[, area_filter, tolerance])	Filter faces by area.
get_touching_faces(assignment)	Get the objects that touch one of the face center of each face of the object.
history()	Object history.
intersect(assignment[, keep_originals])	Intersect the active object with a given list.
largest_face([n])	Return only the face with the greatest area.
longest_edge([n])	Return only the edge with the greatest length.
mirror(origin, vector[, duplicate])	Mirror a selection.
move(vector)	Move objects from a list.
plot([show])	Plot model with PyVista.
rotate(axis[, angle, units])	Rotate the selection.
section(plane[, create_new, ...])	Section the object.
shortest_edge([n])	Return only the edge with the smallest length.
smallest_face([n])	Return only the face with the smallest area.
split(plane[, sides])	Split the active object.
subtract(tool_list[, keep_originals])	Subtract one or more parts from the object.
sweep_along_path(sweep_object[, ...])	Sweep the selection along a vector.
sweep_along_vector(sweep_vector[, ...])	Sweep the selection along a vector.
sweep_around_axis(axis[, sweep_angle, ...])	Sweep around an axis.
touching_conductors()	Get the conductors of given object.
unite(assignment)	Unite a list of objects with this object.
wrap_sheet(object_name[, imprinted])	Execute the sheet wrapping around an object.

Attributes

bottom_edge_x	Bottom edge in the X direction of the object.
bottom_edge_y	Bottom edge in the Y direction of the object.
bottom_edge_z	Bottom edge in the Z direction of the object.
bottom_face_x	Bottom face in the X direction of the object.
bottom_face_y	Bottom face in the X direction of the object.
bottom_face_z	Bottom face in the Z direction of the object.
bounding_box	Bounding box of a part.
bounding_dimension	Retrieve the dimension array of the bounding box.
color	Part color as a tuple of integer values for (<i>Red</i> , <i>Green</i> , <i>Blue</i>) color values.
color_string	Color tuple as a string in the format '(Red, Green, Blue)'.

continues on next page

Table 32 – continued from previous page

display_wireframe	Wireframe property of the part.
edges	Information for each edge in the given part.
face_closest_to_bounding_box	Return only the face ids of the face closest to the bounding box.
faces	Information for each face in the given part.
faces_on_bounding_box	Return only the face ids of the faces touching the bounding box.
group_name	Group the object belongs to.
id	ID of the object.
is3d	Check for if the object is 3D.
is_conductor	Check if the object is a conductor.
logger	Logger.
mass	Object mass.
material_name	Material name of the object.
model	Part model or non-model property.
name	Name of the object as a string value.
object_type	Type of the object.
object_units	Object units.
part_coordinate_system	Part coordinate system.
solve_inside	Part solve inside flag.
surface_material_name	Surface material name of the object.
top_edge_x	Top edge in the X direction of the object.
top_edge_y	Top edge in the Y direction of the object.
top_edge_z	Top edge in the Z direction of the object.
top_face_x	Top face in the X direction of the object.
top_face_y	Top face in the Y direction of the object.
top_face_z	Top face in the Z direction of the object.
touching_objects	Get the objects that touch a vertex, edge midpoint, or face of the object.
transparency	Part transparency as a value between 0.0 and 1.0.
valid_properties	Valid properties.
vertices	Information for each vertex in the given part.
volume	Object volume.

pyaedt.modeler.cad.elements3d.FacePrimitive

```
class pyaedt.modeler.cad.elements3d.FacePrimitive(object3d, obj_id)
```

Contains the face object within the AEDT Desktop Modeler.

```
__init__(object3d, obj_id)
```

Parameters

object3d	
	[pyaedt.modeler.cad.object3d.Object3d]
obj_id	
	[int]

Methods

<code>create_object([non_model])</code>	Return a new object from the selected face.
<code>is_on_bounding([tolerance])</code>	Check if the face is on bounding box or Not.
<code>move_with_offset([offset])</code>	Move the face along the normal.
<code>move_with_vector(vector)</code>	Move the face along a vector.

Attributes

<code>area</code>	Face area.
<code>bottom_edge_x</code>	Bottom edge in the X direction of the object.
<code>bottom_edge_y</code>	Bottom edge in the X direction of the object.
<code>bottom_edge_z</code>	Bottom edge in the Z direction of the object.
<code>center</code>	Face center in model units.
<code>center_from_aedt</code>	Face center for a planar face in model units.
<code>edges</code>	Edges lists.
<code>id</code>	Face ID.
<code>is_planar</code>	Check if a face is planar or not.
<code>logger</code>	Logger.
<code>normal</code>	Face normal.
<code>oeditor</code>	Oeditor Module.
<code>top_edge_x</code>	Top edge in the X direction of the object.
<code>top_edge_y</code>	Top edge in the Y direction of the object.
<code>top_edge_z</code>	Top edge in the Z direction of the object.
<code>touching_objects</code>	Get the objects that touch one of the vertex, edge mid-point or the actual face.
<code>vertices</code>	Vertices lists.

`pyaedt.modeler.cad.elements3d.EdgePrimitive`

```
class pyaedt.modeler.cad.elements3d.EdgePrimitive(object3d, edge_id)
```

Contains the edge object within the AEDT Desktop Modeler.

Parameters

<code>object3d</code>	
	[<code>pyaedt.modeler.cad.object3d.Object3d</code>] Pointer to the calling object that provides additional functionality.
<code>edge_id</code>	
	[<code>int</code>] Object ID as determined by the parent object.

```
__init__(object3d, edge_id)
```

Methods

<code>chamfer([left_distance, right_distance, ...])</code>	Add a chamfer to the selected edges in 3D/vertices in 2D.
<code>create_object([non_model])</code>	Return a new object from the selected edge.
<code>fillet([radius, setback])</code>	Add a fillet to the selected edges in 3D/vertices in 2D.
<code>move_along_normal([offset])</code>	Move this edge.

Attributes

<code>length</code>	Length of the edge.
<code>midpoint</code>	Midpoint coordinates of the edge.
<code>segment_info</code>	Compute segment information using the object-oriented method (from AEDT 2021 R2 with beta options).
<code>vertices</code>	Vertices list.

`pyaedt.modeler.cad.elements3d.VertexPrimitive`

`class pyaedt.modeler.cad.elements3d.VertexPrimitive(object3d, objid, position=None)`

Contains the vertex object within the AEDT Desktop Modeler.

Parameters

object3d
`[pyaedt.modeler.cad.object3d.Object3d]` Pointer to the calling object that provides additional functionality.

objid
`[int]` Object ID as determined by the parent object.

`__init__(object3d, objid, position=None)`

Methods

<code>chamfer([left_distance, right_distance, ...])</code>	Add a chamfer to the selected edges in 3D/vertices in 2D.
<code>fillet([radius, setback])</code>	Add a fillet to the selected edges in 3D/vertices in 2D.

Attributes

position	Position of the vertex.
----------	-------------------------

pyaedt.modeler.cad.polylines.PolylineSegment

```
class pyaedt.modeler.cad.polylines.PolylineSegment(segment_type, num_seg=0, num_points=0,  
arc_angle=0, arc_center=None,  
arc_plane=None)
```

Creates and manipulates a segment of a polyline.

Parameters

segment_type

[str] Type of the object. Choices are "Line", "Arc", "Spline", and "AngularArc".

num_seg

[int, optional] Number of segments for the types "Arc", "Spline", and "AngularArc".
The default is 0. For the type Line, this parameter is ignored.

num_points

[int, optional] Number of control points for the type Spline. For other types, this parameter is defined automatically.

arc_angle

[float or str, optional] Sweep angle in radians or a valid value string. For example, "35deg" or 0.25. This argument is Specific to type AngularArc.

arc_center

[list or str, optional] List of values in model units or a valid value string. For example, a list of [x, y, z] coordinates. This argument is Specific to type AngularArc.

arc_plane

[str, int optional] Plane in which the arc sweep is performed in the active coordinate system "XY", "YZ" or "ZX". The default is None, in which case the plane is determined automatically by the first coordinate for which the starting point and center point have the same value. This argument is Specific to type AngularArc.

Examples

See `pyaedt.Primitives.Polyline`.

```
__init__(segment_type, num_seg=0, num_points=0, arc_angle=0, arc_center=None, arc_plane=None)
```

`pyaedt.modeler.cad.polyline.Polyline`

```
class pyaedt.modeler.cad.polyline.Polyline(primitives, src_object=None, position_list=None,
                                             segment_type=None, cover_surface=False,
                                             close_surface=False, name=None, matname=None,
                                             xsection_type=None, xsection_orient=None,
                                             xsection_width=1, xsection_topwidth=1,
                                             xsection_height=1, xsection_num_seg=0,
                                             xsection_bend_type=None, non_model=False)
```

Creates and manipulates a polyline.

The constructor for this class is intended to be called from the `pyaedt.modeler.Primitives.Primitives.create_polyline()` method. The documentation is provided there.

The returned Polyline object exposes the methods for manipulating the polyline.

Parameters

primitives
`[pyaedt.modeler.Primitives3D.Primitives3D]` Pointer to the parent Primitives object.

src_object
`[optional]` The default is `None`. If specified, all other arguments are ignored.

position_list
`[list, optional]` List of positions in the `[x, y, z]` format. The default is `None`. It is mandatory if `scr_object` is not specified.

segment_type
`[str or PolylineSegment or list, optional]` Define the list of segment types. For a string, "Line" or "Arc" is valid. Use a "PolylineSegment", for "Line", "Arc", "Spline", or "AngularArc". A list of segment types (str or `pyaedt.modeler.Primitives.PolylineSegment`) is valid for a compound polyline. The default is `None`.

cover_surface
`[bool, optional]` The default is `False`.

close_surface
`[bool, optional]` The default is `False`.

name
`[str, optional]` The default is `None`.

matname
`[str, optional]` Name of the material. The default is `None`.

xsection_type
`[str, optional]` Type of the cross-section. Options are "Line", "Circle", "Rectangle", and "Isosceles Trapezoid". The default is `None`.

xsection_orient
`[str, optional]` Direction of the normal vector to the width of the cross-section. Options are "X", "Y", "Z", and "Auto". The default is `None`.

xsection_width
`[float or str, optional]` Width or diameter of the cross-section for all types. The default is `1`.

xsection_topwidth

[`float` or `str`, optional] Top width of the cross-section for the type "Isosceles Trapezoid" only. The default is 1.

xsection_height

[`float` or `str`, optional] Height of the cross-section for the types "Rectangle" and "Isosceles Trapezoid" only. The default is 1.

xsection_num_seg

[`int`, optional] Number of segments in the cross-section surface for the types "Circle", "Rectangle" and "Isosceles Trapezoid". The default is 0. The value must be 0 or greater than 2.

xsection_bend_type

[`str`, optional] Type of the bend. The default is `None`, in which case the bend type is set to "Corner". For the type "Circle", the bend type should be set to "Curved".

`__init__(primitives, src_object=None, position_list=None, segment_type=None, cover_surface=False, close_surface=False, name=None, matname=None, xsection_type=None, xsection_orient=None, xsection_width=1, xsection_topwidth=1, xsection_height=1, xsection_num_seg=0, xsection_bend_type=None, non_model=False)`

Parameters**primitives**

[`pyaedt.modeler.Primitives3D.Primitives3D`] Inherited parent object.

name

[`str`]

Methods

<code>clone()</code>	Clone a polyline object.
<code>delete()</code>	Delete the object.
<code>detach_faces(faces)</code>	Section the object.
<code>duplicate_along_line(vector[, clones, attach])</code>	Duplicate the object along a line.
<code>duplicate_around_axis(axis[, angle, clones, ...])</code>	Duplicate the object around the axis.
<code>edges_by_length(length[, length_filter, ...])</code>	Filter edges by length.
<code>export_image([output_file])</code>	Export the current object to a specified file path.
<code>faces_by_area(area[, area_filter, tolerance])</code>	Filter faces by area.
<code>get_touching_faces(assignment)</code>	Get the objects that touch one of the face center of each face of the object.
<code>history()</code>	Object history.
<code>insert_segment(points[, segment])</code>	Add a segment to an existing polyline.
<code>intersect(assignment[, keep_originals])</code>	Intersect the active object with a given list.
<code>largest_face([n])</code>	Return only the face with the greatest area.
<code>longest_edge([n])</code>	Return only the edge with the greatest length.
<code>mirror(origin, vector[, duplicate])</code>	Mirror a selection.
<code>move(vector)</code>	Move objects from a list.
<code>plot([show])</code>	Plot model with PyVista.
<code>remove_point(position[, tolerance])</code>	Remove a point from an existing polyline by position.
<code>remove_segments(assignment)</code>	Remove a segment from an existing polyline by segment id.
<code>rotate(axis[, angle, units])</code>	Rotate the selection.

continues on next page

Table 33 – continued from previous page

<code>section(plane[, create_new, ...])</code>	Section the object.
<code>set_crosssection_properties([type, orient, ...])</code>	Set the properties of an existing polyline object.
<code>shortest_edge([n])</code>	Return only the edge with the smallest length.
<code>smallest_face([n])</code>	Return only the face with the smallest area.
<code>split(plane[, sides])</code>	Split the active object.
<code>subtract(tool_list[, keep_originals])</code>	Subtract one or more parts from the object.
<code>sweep_along_path(sweep_object[, ...])</code>	Sweep the selection along a vector.
<code>sweep_along_vector(sweep_vector[, ...])</code>	Sweep the selection along a vector.
<code>sweep_around_axis(axis[, sweep_angle, ...])</code>	Sweep around an axis.
<code>touching_conductors()</code>	Get the conductors of given object.
<code>unite(assignment)</code>	Unite a list of objects with this object.
<code>wrap_sheet(object_name[, imprinted])</code>	Execute the sheet wrapping around an object.

Attributes

<code>bottom_edge_x</code>	Bottom edge in the X direction of the object.
<code>bottom_edge_y</code>	Bottom edge in the Y direction of the object.
<code>bottom_edge_z</code>	Bottom edge in the Z direction of the object.
<code>bottom_face_x</code>	Bottom face in the X direction of the object.
<code>bottom_face_y</code>	Bottom face in the X direction of the object.
<code>bottom_face_z</code>	Bottom face in the Z direction of the object.
<code>bounding_box</code>	Bounding box of a part.
<code>bounding_dimension</code>	Retrieve the dimension array of the bounding box.
<code>color</code>	Part color as a tuple of integer values for (<i>Red</i> , <i>Green</i> , <i>Blue</i>) color values.
<code>color_string</code>	Color tuple as a string in the format '(Red, Green, Blue)'.
<code>display_wireframe</code>	Wireframe property of the part.
<code>edges</code>	Information for each edge in the given part.
<code>end_point</code>	List of the [x, y, z] coordinates for the ending point in the polyline object in the object's coordinate system.
<code>face_closest_to_bounding_box</code>	Return only the face ids of the face closest to the bounding box.
<code>faces</code>	Information for each face in the given part.
<code>faces_on_bounding_box</code>	Return only the face ids of the faces touching the bounding box.
<code>group_name</code>	Group the object belongs to.
<code>id</code>	ID of the object.
<code>is3d</code>	Check for if the object is 3D.
<code>is_conductor</code>	Check if the object is a conductor.
<code>logger</code>	Logger.
<code>mass</code>	Object mass.
<code>material_name</code>	Material name of the object.
<code>model</code>	Part model or non-model property.
<code>name</code>	Name of the object as a string value.
<code>object_type</code>	Type of the object.
<code>object_units</code>	Object units.
<code>part_coordinate_system</code>	Part coordinate system.

continues on next page

Table 34 – continued from previous page

<code>points</code>	Polyline Points.
<code>segment_types</code>	List of the segment types of the polyline.
<code>solve_inside</code>	Part solve inside flag.
<code>start_point</code>	List of the [x, y, z] coordinates for the starting point in the polyline object in the object's coordinate system.
<code>surface_material_name</code>	Surface material name of the object.
<code>top_edge_x</code>	Top edge in the X direction of the object.
<code>top_edge_y</code>	Top edge in the Y direction of the object.
<code>top_edge_z</code>	Top edge in the Z direction of the object.
<code>top_face_x</code>	Top face in the X direction of the object.
<code>top_face_y</code>	Top face in the Y direction of the object.
<code>top_face_z</code>	Top face in the Z direction of the object.
<code>touching_objects</code>	Get the objects that touch a vertex, edge midpoint, or face of the object.
<code>transparency</code>	Part transparency as a value between 0.0 and 1.0.
<code>valid_properties</code>	Valid properties.
<code>vertex_positions</code>	List of the [x, y, z] coordinates for all vertex positions in the polyline object in the object's coordinate system.
<code>vertices</code>	Information for each vertex in the given part.
<code>volume</code>	Object volume.

pyaedt.modeler.cad.component_array.ComponentArray

```
class pyaedt.modeler.cad.component_array.ComponentArray(app, name=None)
```

Manages object attributes for a 3D component array.

Parameters

- app**
[pyaedt.Hfss] HFSS PyAEDT object.
- name**
[str, optional] Array name. The default is `None`, in which case a random name is assigned.

Examples

Basic usage demonstrated with an HFSS design with an existing array:

```
>>> from pyaedt import Hfss
>>> aedtapp = Hfss(projectname="Array.aedt")
>>> array_names = aedtapp.component_array_names[0]
>>> array = aedtapp.component_array[array_names[0]]
```

```
__init__(app, name=None)
```

Methods

<code>delete()</code>	Delete the component array.
<code>edit_array()</code>	Edit component array.
<code>export_array_info([output_file])</code>	Export array information to a CSV file.
<code>get_cell(row, col)</code>	Get cell object corresponding to a row and column.
<code>get_cell_position()</code>	Get cell position.
<code>get_component_objects()</code>	Get 3D component center.
<code>lattice_vector()</code>	Get model lattice vector.
<code>parse_array_info_from_csv(input_file)</code>	Parse component array information from the CSV file.
<code>update_properties()</code>	Update component array properties.

Attributes

<code>a_length</code>	Length of the array in A direction.
<code>a_size</code>	Number of cells in the vector A direction.
<code>a_vector_choices</code>	List of name choices for vector A.
<code>a_vector_name</code>	Name of vector A.
<code>b_length</code>	Length of the array in B direction.
<code>b_size</code>	Number of cells in the vector B direction.
<code>b_vector_choices</code>	List of name choices for vector B.
<code>b_vector_name</code>	Name of vector B.
<code>cells</code>	List of pyaedt.modeler.cad.component_array.CellArray objects.
<code>component_names</code>	List of component names.
<code>coordinate_system</code>	Coordinate system name.
<code>name</code>	Name of the array.
<code>padding_cells</code>	Number of padding cells.
<code>post_processing_cells</code>	Dictionary of each component's postprocessing cells.
<code>properties</code>	Ordered dictionary of the properties of the component array.
<code>render</code>	Array rendering.
<code>render_choices</code>	List of rendered name choices.
<code>render_id</code>	Array rendering ID.
<code>show_cell_number</code>	Flag indicating if the array cell number is shown.
<code>visible</code>	Flag indicating if the array is visible.

pyaedt.modeler.cad.components_3d.UserDefinedComponent

```
class pyaedt.modeler.cad.components_3d.UserDefinedComponent(primitives, name=None, props=None, component_type=None)
```

Manages object attributes for 3DComponent and User Defined Model.

Parameters

primitives

[pyaedt.modeler.Primitives3D.Primitives3D] Inherited parent object.

name

[str, optional] Name of the component. The default value is None.

props

[**dict**, optional] Dictionary of properties. The default value is `None`.

component_type

[**str**, optional] Type of the component. The default value is `None`.

Examples

Basic usage demonstrated with an HFSS design:

```
>>> from pyaedt import Hfss
>>> aedtapp = Hfss()
>>> prim = aedtapp.modeler.user_defined_components
```

Obtain user defined component names, to return a `pyaedt.modeler.cad.components_3d.UserDefinedComponent`.

```
>>> component_names = aedtapp.modeler.user_defined_components
>>> component = aedtapp.modeler[component_names["3DC_Cell_Radome_In1"]]
```

`__init__(primitives, name=None, props=None, component_type=None)`

Methods

<code>delete()</code>	Delete the object.
<code>duplicate_along_line(vector[, clones, attach])</code>	Duplicate the object along a line.
<code>duplicate_and_mirror(origin, vector)</code>	Duplicate and mirror a selection.
<code>duplicate_around_axis(axis[, angle, clones, ...])</code>	Duplicate the component around the axis.
<code>edit_definition([password])</code>	Edit 3d Definition.
<code>get_component_filepath()</code>	Get 3d component file path.
<code>history()</code>	Component history.
<code>mirror(origin, vector)</code>	Mirror a selection.
<code>move(vector)</code>	Move component from a list.
<code>rotate(axis[, angle, units])</code>	Rotate the selection.
<code>update_definition([password, output_file, ...])</code>	Update 3d component definition.
<code>update_native()</code>	Update the Native Component in AEDT.

Attributes

<code>bounding_box</code>	Get bounding dimension of a user defined model.
<code>center</code>	Get center coordinates of a user defined model.
<code>group_name</code>	Group the component belongs to.
<code>is3dcomponent</code>	3DComponent flag.
<code>layout_component</code>	Layout component object.
<code>mesh_assembly</code>	Mesh assembly flag.
<code>name</code>	Name of the object.
<code>parameters</code>	Component parameters.
<code>parts</code>	Dictionary of objects that belong to the user-defined component.
<code>target_coordinate_system</code>	Target coordinate system.

pyaedt.modeler.cad.elements3d.Point

```
class pyaedt.modeler.cad.elements3d.Point(primitives, name)
```

Manages point attributes for the AEDT 3D Modeler.

Parameters**primitives**

[pyaedt.modeler.Primitives3D.Primitives3D] Inherited parent object.

name

[str] Name of the point.

Examples

Basic usage demonstrated with an HFSS design:

```
>>> from pyaedt import Hfss
>>> aedtapp = Hfss()
>>> primitives = aedtapp.modeler
```

Create a point, to return an pyaedt.modeler.Object3d.Point.

```
>>> point = primitives.create_point([30, 30, 0], "my_point", (0, 195, 255))
>>> my_point = primitives.points[point.name]
```

__init__(primitives, name)**Methods**

<code>delete()</code>	Delete the point.
<code>set_color(color_value)</code>	Set symbol color.

Attributes

<code>coordinate_system</code>	Coordinate system of the point.
<code>logger</code>	Logger.
<code>name</code>	Name of the point as a string value.
<code>valid_properties</code>	Valid properties.

pyaedt.modeler.cad.elements3d.Plane

```
class pyaedt.modeler.cad.elements3d.Plane(primitives, name)
```

Manages plane attributes for the AEDT 3D Modeler.

Parameters**primitives**

[pyaedt.modeler.Primitives3D.Primitives3D] Inherited parent object.

name

[str] Name of the point.

Examples

Basic usage demonstrated with an HFSS design:

```
>>> from pyaedt import Hfss
>>> aedtapp = Hfss()
>>> primitives = aedtapp.modeler
```

Create a plane, to return an pyaedt.modeler.Object3d.Plane.

```
>>> plane = primitives.create_plane("my_plane", "-0.7mm", "0.3mm", "0mm", "0.7mm",
    ↪ "-0.3mm", "0mm", "(0, 195, 255)")
>>> my_plane = primitives.planes[plane.name]
```

`__init__(primitives, name)`

Methods

<code>delete()</code>	Delete the plane.
<code>set_color(color_value)</code>	Set symbol color.

Attributes

<code>coordinate_system</code>	Coordinate system of the plane.
<code>logger</code>	Logger.
<code>name</code>	Name of the plane as a string value.
<code>valid_properties</code>	Valid properties.

pyaedt.modeler.cad.elements3d.HistoryProps

`class pyaedt.modeler.cad.elements3d.HistoryProps(child_object, props)`

Manages an objects history properties.

`__init__(child_object, props)`

Methods

<code>clear()</code>	
<code>copy()</code>	
<code>fromkeys(/, iterable[, value])</code>	Create a new ordered dictionary with keys from iterable and values set to value.
<code>get(key[, default])</code>	Return the value for key if key is in the dictionary, else default.
<code>items()</code>	
<code>keys()</code>	
<code>move_to_end(/, key[, last])</code>	Move an existing element to the end (or beginning if last is false).
<code>pop(/, key[, default])</code>	If the key is not found, return the default if given; otherwise, raise a KeyError.
<code>popitem(/[, last])</code>	Remove and return a (key, value) pair from the dictionary.
<code>setdefault(/, key[, default])</code>	Insert key with a value of default if key is not in the dictionary.
<code>update([E,]**F)</code>	If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]
<code>values()</code>	

pyaedt.modeler.cad.elements3d.BinaryTreeNode

```
class pyaedt.modeler.cad.elements3d.BinaryTreeNode(node, child_object, first_level=False,
                                                 get_child_obj_arg=None, root_name=None)
```

Manages an objects history structure.

```
__init__(node, child_object, first_level=False, get_child_obj_arg=None, root_name=None)
```

Methods

<code>jsonalize_tree()</code>	Create dictionary from the Binary Tree.
<code>suppress_all(app)</code>	Activate suppress option for all the operations contained in the binary tree node.
<code>unsuppress_all(app)</code>	Disable suppress option for all the operations contained in the binary tree node.
<code>update_property(prop_name, prop_value)</code>	Update the property of the binary tree node.

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
```

(continues on next page)

(continued from previous page)

```

non_graphical=False, new_desktop_session=True,
close_on_exit=True, student_version=False)

# This call returns the Modeler3D class
modeler = app.modeler

# This call returns a Primitives3D object
primitives = modeler

# This call returns an Object3d object
my_box = primitives.create_box([0,0,0],[10,10,10])

# Getter and setter
my_box.material_name
my_box.material_name = "copper"

my_box.faces[0].center

...

```

2.5.2 Coordinate systems and geometry operators

This module contains all properties and methods needed to edit a coordinate system and a set of useful geometry operators. The `CoordinateSystem` class is accessible through the `create_coordinate_system` method or the `coordinate_systems` list. The `GeometryOperators` class can be imported and used because it is made by static methods.

<code>cad.Modeler.CoordinateSystem</code>	Manages coordinate system data and execution.
<code>geometry_operators.GeometryOperators</code>	Manages geometry operators.

pyaedt.modeler.cad.Modeler.CoordinateSystem

`class pyaedt.modeler.cad.Modeler.CoordinateSystem(modeler, props=None, name=None)`

Manages coordinate system data and execution.

Parameters

`modeler`

Inherited parent object.

`props`

[`dict`, optional] Dictionary of properties. The default is None.

`name`

[optional] The default is None.

`__init__(modeler, props=None, name=None)`

Methods

<code>change_cs_mode([mode_type])</code>	Change the mode of the coordinate system.
<code>create([origin, reference_cs, name, mode, ...])</code>	Create a coordinate system.
<code>delete()</code>	Delete the coordinate system.
<code>rename(name)</code>	Rename the coordinate system.
<code>set_as_working_cs()</code>	Set the coordinate system as the working coordinate system.
<code>update()</code>	Update the coordinate system.

Attributes

<code>available_properties</code>	Available properties.
<code>mode</code>	Coordinate System mode.
<code>origin</code>	Coordinate system origin in model units.
<code>props</code>	Coordinate System Properties.
<code>quaternion</code>	Quaternion computed based on specific axis mode.
<code>ref_cs</code>	Reference coordinate system getter and setter.

pyaedt.modeler.geometry_operators.GeometryOperators

```
class pyaedt.modeler.geometry_operators.GeometryOperators
    Manages geometry operators.

    __init__(*args, **kwargs)
```

Methods

<code>List2list(input_list)</code>	Convert a C# list object to a Python list.
<code>are_segments_intersecting(a1, a2, b1, b2[, ...])</code>	Determine if the two segments a and b are intersecting.
<code>arrays_positions_sum(vertlist1, vertlist2)</code>	Return the sum of two vertices lists.
<code>atan2(y, x)</code>	Implementation of atan2 that does not suffer from the following issues: $\text{math.atan2}(0.0, 0.0) = 0.0$ $\text{math.atan2}(-0.0, 0.0) = -0.0$ $\text{math.atan2}(0.0, -0.0) = 3.141592653589793$ $\text{math.atan2}(-0.0, -0.0) = -3.141592653589793$ and returns always 0.0.
<code>axis_angle_to_quaternion(u, theta)</code>	Convert the axis angle rotation formulation to a quaternion.
<code>axis_to_euler_zxz(x, y, z)</code>	Retrieve Euler angles of a frame following the rotation sequence ZXZ.
<code>axis_to_euler_zyz(x, y, z)</code>	Retrieve Euler angles of a frame following the rotation sequence ZYZ.
<code>cs_axis_str(val)</code>	Retrieve a string for a coordinate system axis.
<code>cs_plane_to_axis_str(val)</code>	Retrieve a string for a coordinate system plane.
<code>cs_plane_to_plane_str(val)</code>	Retrieve a string for a coordinate system plane.

continues on next page

Table 35 – continued from previous page

<code>cs_xy_pointing_expression(yaw, pitch, roll)</code>	Return x_pointing and y_pointing vectors as expressions from the yaw, ptich, and roll input (as strings).
<code>deg2rad(angle)</code>	Convert the angle from degrees to radians.
<code>degrees_default_rounded(angle, digits)</code>	Convert angle to degree with given digits rounding.
<code>degrees_over_rounded(angle, digits)</code>	Ceil of angle.
<code>distance_vector(p, a, b)</code>	Evaluate the vector distance between point p and a line defined by two points, a and b.
<code>draft_type_str(val)</code>	Retrieve the draft type.
<code>euler_zxz_to_quaternion(phi, theta, psi)</code>	Convert the Euler angles following rotation sequence ZXZ to a quaternion.
<code>euler_zyz_to_quaternion(phi, theta, psi)</code>	Convert the Euler angles following rotation sequence ZYZ to a quaternion.
<code>find_closest_points(points_list, reference_point)</code>	Given a list of points, finds the closest points to a reference point.
<code>find_largest_rectangle_inside_polygon(poly)</code>	Find the largest area rectangles of arbitrary orientation in a polygon.
<code>find_point_on_plane(pointlists[, direction])</code>	Find a point on a plane.
<code>get_mid_point(v1, v2)</code>	Evaluate the midpoint between two points.
<code>get_numeric(s)</code>	Convert a string to a numeric value.
<code>get_polygon_centroid(pts)</code>	Evaluate the centroid of a polygon defined by its points.
<code>get_triangle_area(v1, v2, v3)</code>	Evaluate the area of a triangle defined by its three vertices.
<code>is_between_points(p, a, b[, tol])</code>	Check if a point lies on the segment defined by two points.
<code>is_collinear(a, b[, tol])</code>	Check if two vectors are collinear (parallel or anti-parallel).
<code>is_parallel(a1, a2, b1, b2[, tol])</code>	Check if a segment defined by two points is parallel to a segment defined by two other points.
<code>is_perpendicular(a, b[, tol])</code>	Check if two vectors are perpendicular.
<code>is_point_in_polygon(point, polygon)</code>	Determine if a point is inside or outside a polygon, both located on the same plane.
<code>is_point_projection_in_segment(p, a, b)</code>	Check if a point projection lies on the segment defined by two points.
<code>is_projection_inside(a1, a2, b1, b2)</code>	Project a segment onto another segment and check if the projected segment is inside it.
<code>is_segment_intersecting_polygon(a, b, polygon)</code>	Determine if a segment defined by two points a and b intersects a polygon.
<code>is_small(s)</code>	Return True if the number represented by s is zero (i.e very small).
<code>mirror_point(start, reference, vector)</code>	Mirror point about a plane defining by a point on the plane and a normal point.
<code>normalize_vector(v)</code>	Normalize a geometry vector.
<code>numeric_cs(cs_in)</code>	Return a list of [x,y,z] numeric values given a coordinate system as input.
<code>orient_polygon(x, y[, clockwise])</code>	Orient a polygon clockwise or counterclockwise.
<code>parallel_coeff(a1, a2, b1, b2)</code>	ADD DESCRIPTION.
<code>parse_dim_arg(string[, scale_to_unit, ...])</code>	Convert a number and unit to a float.
<code>point_in_polygon(point, polygon[, tolerance])</code>	Determine if a point is inside, outside the polygon or at exactly at the border.
<code>point_segment_distance(p, a, b)</code>	Calculate the distance between a point p and a segment defined by two points a and b.

continues on next page

Table 35 – continued from previous page

<code>pointing_to_axis(x_pointing, y_pointing)</code>	Retrieve the axes from the HFSS X axis and Y pointing axis as per the definition of the AEDT interface coordinate system.
<code>points_distance(p1, p2)</code>	Evaluate the distance between two points expressed as their Cartesian coordinates.
<code>q_prod(p, q)</code>	Evaluate the product of two quaternions, p and q, defined as: $p = p_0 + p' = p_0 + ip_1 + jp_2 + kp_3$.
<code>q_rotation(v, q)</code>	Evaluate the rotation of a vector, defined by a quaternion.
<code>q_rotation_inv(v, q)</code>	Evaluate the inverse rotation of a vector that is defined by a quaternion.
<code>quaternion_to_axis(q)</code>	Convert a quaternion to a rotated frame defined by X, Y, and Z axes.
<code>quaternion_to_axis_angle(q)</code>	Convert a quaternion to the axis angle rotation formulation.
<code>quaternion_to_euler_zxz(q)</code>	Convert a quaternion to Euler angles following rotation sequence ZXZ.
<code>quaternion_to_euler_zyz(q)</code>	Convert a quaternion to Euler angles following rotation sequence ZYZ.
<code>rad2deg(angle)</code>	Convert the angle from radians to degrees.
<code>radians_default_rounded(angle, digits)</code>	Convert to radians with given round.
<code>radians_over_rounded(angle, digits)</code>	Radian angle ceiling.
<code>v_angle(a, b)</code>	Evaluate the angle between two geometry vectors.
<code>v_angle_sign(va, vb, vn[, right_handed])</code>	Evaluate the signed angle between two geometry vectors.
<code>v_angle_sign_2D(va, vb[, right_handed])</code>	Evaluate the signed angle between two 2D geometry vectors.
<code>v_cross(a, b)</code>	Evaluate the cross product of two geometry vectors.
<code>v_dot(a, b)</code>	Evaluate the dot product between two geometry vectors.
<code>v_norm(a)</code>	Evaluate the Euclidean norm of a geometry vector.
<code>v_points(p1, p2)</code>	Vector from one point to another point.
<code>v_prod(s, v)</code>	Evaluate the product between a scalar value and a vector.
<code>v_rotate_about_axis(vector, angle[, ...])</code>	Evaluate rotation of a vector around an axis.
<code>v_sub(a, b)</code>	Evaluate two geometry vectors by subtracting them (a-b).
<code>v_sum(a, b)</code>	Evaluate two geometry vectors by adding them (a+b).

```

from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call returns the CoordinateSystem object list
cs = app.modeler.coordinate_systems

# This call returns a CoordinateSystem object
new_cs = app.modeler.create_coordinate_system()

...

```

2.5.3 Advanced modeler operations

PyAEDT includes some advanced modeler tools like `MultiPartComponent` for 3D component management and `Stackup3D` for parametric creation of 3D modeler stackups.

Multi-part components

This section lists classes for creating and editing multi-part components in the 3D tools. This consists of a set of one or more 3D component objects, linked together and parametrized to allow movements.

<code>actors.Person</code>	Provides an instance of a person.
<code>actors.Vehicle</code>	Provides an instance of a vehicle.
<code>actors.Bird</code>	Provides an instance of a bird.
<code>actors.Radar</code>	Manages the radar definition and placement in the HFSS design.
<code>parts.Part</code>	Manages 3D component placement and definition.
<code>parts.Antenna</code>	Manages antennas.
<code>multiparts.MultiPartComponent</code>	Supports multi-part 3D components for HFSS SBR+.
<code>multiparts.Environment</code>	Supports multi-part 3D components without motion for HFSS SBR+.
<code>multiparts.Actor</code>	Provides an instance of an actor.

`pyaedt.modeler.advanced_cad.actors.Person`

```
class pyaedt.modeler.advanced_cad.actors.Person(actor_folder, speed='0', stride='0.8meters',  
                                              relative_cs_name=None)
```

Provides an instance of a person.

This class is derived from `pyaedt.modeler.multiparts.MultiPartComponent`.

Note: Motion is always forward in the X-axis direction of the person coordinate system.

Parameters

`actor_folder`

[`str`] Full path to the folder containing the definition of the person. This can be changed later in the `Person` class definition.

`speed`

[`float` or `str`, optional] Speed of the person in the X-axis direction. The default is "0".

`stride`

[`float` or `str`, optional] Stride length of the person. The default is 0.8meters.

`relative_cs_name`

[`str`, optional] Name of the relative coordinate system of the actor. The default is `None`, in which case the global coordinate system is used.

```
__init__(actor_folder, speed='0', stride='0.8meters', relative_cs_name=None)
```

Initialize person actor.

Methods

<code>insert(app[, motion])</code>	Insert the person in HFSS SBR+.
<code>position_in_app(app)</code>	Set up design variables and values to enable motion for the multi-part 3D component.
<code>start(app)</code>	Initialize app for SBR+ simulation.

Attributes

<code>cs_name</code>	Coordinate system name.
<code>index</code>	Number of multi-part components.
<code>modeler_units</code>	
<code>name</code>	Unique instance name.
<code>offset</code>	Offset values for the multi-part component.
<code>offset_names</code>	X-, Y-, and Z-axis offset names.
<code>offset_x_name</code>	X-axis offset name.
<code>offset_y_name</code>	Y-axis offset name.
<code>offset_z_name</code>	Z-axis offset name.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>speed_expression</code>	Speed variable expression.
<code>speed_name</code>	Speed variable name.
<code>stride</code>	Stride in meters.
<code>use_global_cs</code>	Global coordinate system.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

`pyaedt.modeler.advanced_cad.actors.Vehicle`

`class pyaedt.modeler.advanced_cad.actors.Vehicle(car_folder, speed=10.0, relative_cs_name=None)`

Provides an instance of a vehicle.

This class is derived from `pyaedt.modeler.multiparts.MultiPartComponent`.

Note: Motion is always forward in the X-axis direction.

Parameters

`car_folder`

[`str`, required] Full path to the folder containing the definition of the vehicle. This can be changed later.

`speed`

[`float` or `str`, optional] Speed of the vehicle. The default is `10.0`.

relative_cs_name

[str, optional] Name of the relative coordinate system of the actor. The default is `None`, in which case the global coordinate system is used.

__init__(car_folder, speed=10.0, relative_cs_name=None)

Vehicle class.

Methods

<code>insert(app[, motion])</code>	Insert the vehicle in HFSS SBR+.
<code>position_in_app(app)</code>	Set up design variables and values to enable motion for the multi-part 3D component.
<code>start(app)</code>	Initialize app for SBR+ simulation.

Attributes

<code>cs_name</code>	Coordinate system name.
<code>index</code>	Number of multi-part components.
<code>modeler_units</code>	
<code>name</code>	Unique instance name.
<code>offset</code>	Offset values for the multi-part component.
<code>offset_names</code>	X-, Y-, and Z-axis offset names.
<code>offset_x_name</code>	X-axis offset name.
<code>offset_y_name</code>	Y-axis offset name.
<code>offset_z_name</code>	Z-axis offset name.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>speed_expression</code>	Speed variable expression.
<code>speed_name</code>	Speed variable name.
<code>use_global_cs</code>	Global coordinate system.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

pyaedt.modeler.advanced_cad.actors.Bird

```
class pyaedt.modeler.advanced_cad.actors.Bird(bird_folder, speed='2.0', flapping_rate='50Hz',
                                             relative_cs_name=None)
```

Provides an instance of a bird.

This class is derived from `pyaedt.modeler.multiparts.MultiPartComponent`.

Note: Motion is always forward in the X-axis direction.

Parameters

bird_folder

[**str**] Full path to the directory containing the definition of the bird. This can be changed later.

speed

[**float** or **str**, optional] Speed of the bird. The default is "2.0".

flapping_rate

[**float** or **str**, optional] Flapping rate. The default is "50Hz".

relative_cs_name

[**str**, optional] Name of the relative coordinate system of the actor. The default is ``None``, in which case the global coordinate system is used.

__init__(bird_folder, speed='2.0', flapping_rate='50Hz', relative_cs_name=None)

Bike class.

Methods

insert(app[, motion])	Insert the bird in HFSS SBR+.
position_in_app(app)	Set up design variables and values to enable motion for the multi-part 3D component.
start(app)	Initialize app for SBR+ simulation.

Attributes

cs_name	Coordinate system name.
index	Number of multi-part components.
modeler_units	
name	Unique instance name.
offset	Offset values for the multi-part component.
offset_names	X-, Y-, and Z-axis offset names.
offset_x_name	X-axis offset name.
offset_y_name	Y-axis offset name.
offset_z_name	Z-axis offset name.
pitch	Pitch variable value.
pitch_name	Pitch variable name.
roll	Roll variable value.
roll_name	Roll variable name.
speed_expression	Speed variable expression.
speed_name	Speed variable name.
use_global_cs	Global coordinate system.
yaw	Yaw variable value.
yaw_name	Yaw variable name.

pyaedt.modeler.advanced_cad.actors.Radar

```
class pyaedt.modeler.advanced_cad.actors.Radar(radar_folder, name=None, motion=False,  
                                              use_relative_cs=False, offset=(0, 0, 0), speed=0,  
                                              relative_cs_name=None)
```

Manages the radar definition and placement in the HFSS design.

Parameters**radar_folder**

[**str**] Full path to the folder containing the radar file.

name

[**str**, optional] Name of the radar file. The default is None.

motion

[**bool**, optional] Whether the actor is in motion. The default is False.

use_relative_cs

[**bool**, optional] Whether to use the relative coordinate system. The default is False.

offset

[**list**, optional] List of offset values. The default is ("0", "0", "0").

speed

[**float** or **str**, optional] Speed of the vehicle. The default is 0.

relative_cs_name

[**str**, optional] Name of the relative coordinate system of the actor. The default is None, in which case the global coordinate system is used.

```
__init__(radar_folder, name=None, motion=False, use_relative_cs=False, offset=(0, 0, 0), speed=0,  
        relative_cs_name=None)
```

Methods

insert(app[, motion])	Insert radar in the HFSS application instance.
position_in_app(app)	Set up design variables and values to enable motion for the multi-part 3D component.
start(app)	Initialize app for SBR+ simulation.

Attributes

<code>cs_name</code>	Coordinate system name.
<code>index</code>	Number of multi-part components.
<code>modeler_units</code>	
<code>name</code>	Unique instance name.
<code>offset</code>	Offset values for the multi-part component.
<code>offset_names</code>	X-, Y-, and Z-axis offset names.
<code>offset_x_name</code>	X-axis offset name.
<code>offset_y_name</code>	Y-axis offset name.
<code>offset_z_name</code>	Z-axis offset name.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>speed_expression</code>	Speed variable expression.
<code>speed_name</code>	Speed variable name.
<code>units</code>	Multi-part units.
<code>use_global_cs</code>	Global coordinate system.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

`pyaedt.modeler.advanced_cad.parts.Part`

class `pyaedt.modeler.advanced_cad.parts.Part(part_folder, part_dict, parent=None, name=None)`

Manages 3D component placement and definition.

Parameters

`part_folder`

[`str`] Path to the folder with the A3DCOMP files.

`part_dict`

[`dict`] Defines relevant properties of the class with the following keywords: * `comp_name`: str, Name of the A3DCOMP file. * `offset`: list or str, Offset coordinate system definition relative to the parent. * `rotation_cs`: list or str, Rotation coordinate system relative to the parent. * `rotation`: str or numeric, Rotation angle. * `compensation_angle`: str or numeric, Initial angle. * `rotation_axis`: str, Rotation axis ("X", "Y", or "Z"). * `duplicate_number`: str or int, Number of instances for linear duplication. * `duplicate_vector`: list, Vector for duplication relative to the parent coordinate system.

`parent`

[`str`] The default is `None`.

`name`

[`str`, optional] Name of the A3DCOMP file without the extension. The default is `None`.

`__init__(part_folder, part_dict, parent=None, name=None)`

Methods

<code>do_rotate(app, aedt_object)</code>	Set the rotation coordinate system relative to the parent coordinate system.
<code>insert(app)</code>	Insert 3D component in AEDT.
<code>set_relative_cs(app)</code>	Create a parametric coordinate system.
<code>zero_offset(kw)</code>	Check if the coordinate system defined by kw is [0, 0, 0].

Attributes

<code>allowed_keys</code>	
<code>cs_name</code>	Coordinate system name.
<code>file_name</code>	Antenna file name.
<code>local_origin</code>	Local part offset values.
<code>name</code>	Part name.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>rot_cs_name</code>	Rotation coordinate system name.
<code>rotate_origin</code>	Origin rotation list.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

`pyaedt.modeler.advanced_cad.parts.Antenna`

`class pyaedt.modeler.advanced_cad.parts.Antenna(root_folder, ant_dict, parent=None, name=None)`

Manages antennas.

This class is derived from *Part*.

Parameters

`root_folder`
[str] Root directory

`ant_dict`
[dict] Antenna dictionary

`parent`
[str, optional] The default is None.

`name`
[str, optional] The default is None.

`__init__(root_folder, ant_dict, parent=None, name=None)`

Methods

<code>do_rotate(app, aedt_object)</code>	Set the rotation coordinate system relative to the parent coordinate system.
<code>insert(app[, units])</code>	Insert antenna in HFSS SBR+.
<code>set_relative_cs(app)</code>	Create a parametric coordinate system.
<code>zero_offset(kw)</code>	Check if the coordinate system defined by kw is [0, 0, 0].

Attributes

<code>allowed_keys</code>	
<code>cs_name</code>	Coordinate system name.
<code>file_name</code>	Antenna file name.
<code>local_origin</code>	Local part offset values.
<code>name</code>	Part name.
<code>params</code>	Multi-part component parameters.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>rot_cs_name</code>	Rotation coordinate system name.
<code>rotate_origin</code>	Origin rotation list.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

`pyaedt.modeler.advanced_cad.multipartes.MultiPartComponent`

```
class pyaedt.modeler.advanced_cad.multipartes.MultiPartComponent(comp_folder, name=None,
                                                               use_relative_cs=False,
                                                               relative_cs_name=None,
                                                               motion=False, offset=('0', '0',
                                                               '0'), yaw='0deg', pitch='0deg',
                                                               roll='0deg')
```

Supports multi-part 3D components for HFSS SBR+.

Note: Forward motion is in the X-axis direction if motion is set.

Parameters

`comp_folder`

[`str`] Full path to the folder with the JSON file containing the component definition. This JSON file must have the same name as the folder.

`name`

[`str`, optional] Name of the multipart component. If this value is set, the component is

selected from the corresponding JSON file in `comp_folder`. The default is `None`, in which case the name of the first JSON file in the folder is used.

use_relative_cs

[`bool`, optional] Whether to use the relative coordinate system. The default is `False`. Set to `False` if the multi-part component doesn't move. Set to `True` if the multi-part component moves relative to the global coordinate system.

relative_cs_name

[`str`, optional] Name of the coordinate system to connect the multipart relative system to when `use_relative_cs=True`.

motion

[`bool`, optional] Whether expressions should be used to define the position and orientation of the multi-part component. The default is `False`.

offset

[`list`, optional] List of [x, y, z] coordinate values defining the component offset. The default is ["0", "0", "0"].

yaw

[`str` or `float`, optional] Yaw angle, indicating the rotation about the components Z-axis. The default is "0deg".

pitch

[`str` or `float`, optional] Pitch angle, indicating the rotation about the component Y-axis. The default is "0deg".

roll

[`str` or `float`, optional] Roll angle, indicating the rotation about the component X-axis. The default is "0deg".

roll

[`str` or `float`, optional] Roll angle, indicating the rotation about the components X-axis. The default

`__init__(comp_folder, name=None, use_relative_cs=False, relative_cs_name=None, motion=False, offset=('0', '0', '0'), yaw='0deg', pitch='0deg', roll='0deg')`

Methods

<code>insert(app[, motion])</code>	Insert the object in HFSS SBR+.
<code>position_in_app(app)</code>	Set up design variables and values to enable motion for the multi-part 3D component.
<code>start(app)</code>	Initialize app for SBR+ simulation.

Attributes

<code>cs_name</code>	Coordinate system name.
<code>index</code>	Number of multi-part components.
<code>modeler_units</code>	
<code>name</code>	Unique instance name.
<code>offset</code>	Offset values for the multi-part component.
<code>offset_names</code>	X-, Y-, and Z-axis offset names.
<code>offset_x_name</code>	X-axis offset name.
<code>offset_y_name</code>	Y-axis offset name.
<code>offset_z_name</code>	Z-axis offset name.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>use_global_cs</code>	Global coordinate system.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

`pyaedt.modeler.advanced_cad.multiparts.Environment`

`class pyaedt.modeler.advanced_cad.multiparts.Environment(env_folder, relative_cs_name=None)`

Supports multi-part 3D components without motion for HFSS SBR+.

This class is derived from `MultiPartComponent`. Its call signature is identical to the parent class except `motion` is always set to `False`.

Parameters

<code>env_folder</code>	[<code>str</code>] Full path to the folder with the JSON file containing the component definition.
<code>relative_cs_name</code>	[<code>str</code> , optional] Name of the coordinate system to connect the components relative system to when <code>use_relative_cs=True</code> . The default is <code>None</code> , in which case the global coordinate system is used.

`__init__(env_folder, relative_cs_name=None)`

Methods

<code>insert(app[, motion])</code>	Insert the object in HFSS SBR+.
<code>position_in_app(app)</code>	Set up design variables and values to enable motion for the multi-part 3D component.
<code>start(app)</code>	Initialize app for SBR+ simulation.

Attributes

cs_name	Coordinate system name.
index	Number of multi-part components.
modeler_units	
name	Unique instance name.
offset	Offset for the multi-part component.
offset_names	X-, Y-, and Z-axis offset names.
offset_x_name	X-axis offset name.
offset_y_name	Y-axis offset name.
offset_z_name	Z-axis offset name.
pitch	Pitch variable value.
pitch_name	Pitch variable name.
roll	Roll variable value.
roll_name	Roll variable name.
use_global_cs	Global coordinate system.
yaw	Yaw variable value.
yaw_name	Yaw variable name.

pyaedt.modeler.advanced_cad.multiparts.Actor

```
class pyaedt.modeler.advanced_cad.multiparts.Actor(actor_folder, speed='0', relative_cs_name=None)
```

Provides an instance of an actor.

This class is derived from *MultiPartComponent*.

Note: Motion is always forward in the X-axis direction.

Parameters

actor_folder

[**str**] Full path to the folder containing the definition of the person. This can be changed later in the Person class definition.

speed

[**float** or **str**] Speed of the person in the X-direction. The default is 0`.

relative_cs_name

[**str**] Name of the relative coordinate system of the actor. The default is None, in which case the global coordinate system is used.

```
__init__(actor_folder, speed='0', relative_cs_name=None)
```

Methods

<code>insert(app[, motion])</code>	Insert the object in HFSS SBR+.
<code>position_in_app(app)</code>	Set up design variables and values to enable motion for the multi-part 3D component.
<code>start(app)</code>	Initialize app for SBR+ simulation.

Attributes

<code>cs_name</code>	Coordinate system name.
<code>index</code>	Number of multi-part components.
<code>modeler_units</code>	
<code>name</code>	Unique instance name.
<code>offset</code>	Offset values for the multi-part component.
<code>offset_names</code>	X-, Y-, and Z-axis offset names.
<code>offset_x_name</code>	X-axis offset name.
<code>offset_y_name</code>	Y-axis offset name.
<code>offset_z_name</code>	Z-axis offset name.
<code>pitch</code>	Pitch variable value.
<code>pitch_name</code>	Pitch variable name.
<code>roll</code>	Roll variable value.
<code>roll_name</code>	Roll variable name.
<code>speed_expression</code>	Speed variable expression.
<code>speed_name</code>	Speed variable name.
<code>use_global_cs</code>	Global coordinate system.
<code>yaw</code>	Yaw variable value.
<code>yaw_name</code>	Yaw variable name.

Stackup 3D components

This section lists `stackup_3d` classes for creating and editing a stackup and objects in the 3D tools. This consists of a set of one or more parametrized layer objects and placing lines, patches, polygons, and vias.

<code>stackup_3d.Stackup3D</code>	Main Stackup3D Class.
<code>stackup_3d.Layer3D</code>	Provides a class for a management of a parametric layer in 3D Modeler.
<code>stackup_3d.Padstack</code>	Provides the Padstack class member of Stackup3D.
<code>stackup_3d.PadstackLayer</code>	Provides a data class for the definition of a padstack layer and relative pad and antipad values.
<code>stackup_3d.Patch</code>	Patch Class in Stackup3D.
<code>stackup_3d.Trace</code>	Trace Class in Stackup3D.
<code>stackup_3d.Polygon</code>	Polygon Class in Stackup3D.
<code>stackup_3d.NamedVariable</code>	Cast PyAEDT variable object to simplify getters and setters in Stackup3D.

pyaedt.modeler.advanced_cad.stackup_3d.Stackup3D

```
class pyaedt.modeler.advanced_cad.stackup_3d.Stackup3D(application, frequency=None)
```

Main Stackup3D Class.

Parameters**application**

[`pyaedt.hfss`] HFSS design or project where the variable is to be created.

frequency

[`float`] The stackup frequency, it will be common to all layers in the stackup.

Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modeler.stackup_3d import Stackup3D
>>> hfss = Hfss(new_desktop_session=True)
>>> my_stackup = Stackup3D(hfss, 2.5e9)
```

```
__init__(application, frequency=None)
```

Methods

<code>add_dielectric_layer(name[, material, ...])</code>	Add a new dielectric layer to the stackup.
<code>add_ground_layer(name[, material, ...])</code>	Add a new ground layer to the stackup.
<code>add_layer(name[, layer_type, material_name, ...])</code>	Add a new layer to the stackup.
<code>add_padstack(name[, material])</code>	Add a new padstack definition.
<code>add_signal_layer(name[, material, ...])</code>	Add a new ground layer to the stackup.
<code>resize(percentage_offset)</code>	Resize the stackup around objects created by a percentage offset.
<code>resize_around_element(element[, ...])</code>	Resize the stackup around parametrized objects and make it parametrize.

Attributes

application	Application object.
dielectric_length	Stackup length.
dielectric_width	Stackup width.
dielectric_x_position	Stackup x origin.
dielectric_y_position	Stackup y origin.
dielectrics	List of dielectrics created.
duplicated_material_list	List of all duplicated material.
frequency	Frequency variable.
grounds	List of grounds created.
layer_names	List of all layer names.
layer_positions	List of all layer positions.
objects	List of objects created.
objects_by_layer	List of definitions created.
padstacks	List of definitions created.
signals	List of signals created.
stackup_layers	Dictionary of all stackup layers.
start_position	Variable containing the start position.
thickness	Total stackup thickness.
z_position_offset	Elevation.

pyaedt.modeler.advanced_cad.stackup_3d.Layer3D

```
class pyaedt.modeler.advanced_cad.stackup_3d.Layer3D(stackup, app, name, layer_type='S',
                                                     material_name='copper', thickness=0.035,
                                                     fill_material='FR4_epoxy', index=1,
                                                     frequency=None)
```

Provides a class for a management of a parametric layer in 3D Modeler. The Layer3D class is not intended to be used with its constructor, but by using the method add_layer available in the Stackup3D class.

Parameters

- stackup**
[pyaedt.modeler.stackup_3d.Stackup3D] The stackup where the layers will be added.
- app**
[pyaedt.hfss.Hfss] HFSS design or project where the variable is to be created.
- name**
[str] Name of the layer.
- layer_type**
[str] S for signal layers, D for dielectric layers, G for ground layers.
- material_name**
[str] The material name of the layer.
- thickness**
[float] The thickness of the layer.
- fill_material**
[str] In ground and signal layers, the dielectric material name which will fill the non-conductive areas of the layer.

index

[int] The number of the layer, starting from bottom to top.

frequency

[float] The layer frequency, it will be common to all geometric shapes on the layer.

Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modeler.advanced_cad.stackup_3d import Stackup3D
>>> hfss = Hfss()
>>> my_stackup = Stackup3D(hfss, 2.5e9)
>>> my_layer = my_stackup.add_layer("my_layer")
>>> gnd = my_stackup.add_ground_layer("gnd")
>>> diel = my_stackup.add_dielectric_layer("diel1", thickness=1.5, material="Duroid\u2122(tm)")
>>> top = my_stackup.add_signal_layer("top")
```

```
__init__(stackup, app, name, layer_type='S', material_name='copper', thickness=0.035,
        fill_material='FR4_epoxy', index=1, frequency=None)
```

Methods

add_patch(frequency, patch_width[, ...])	Create a parametric patch.
add_polygon(points[, material, is_void, ...])	Create a polygon.
add_trace(line_width, line_length[, ...])	Create a trace.
duplicate_parametrize_material(material_name)	Duplicate a material and parametrize all properties.
ml_patch(frequency, patch_width[, ...])	Create a new parametric patch using machine learning algorithm rather than analytic formulas.

Attributes

duplicated_material	Duplicated material.
elevation	Layer elevation.
elevation_value	Layer elevation value.
filling_material	Fill material.
filling_material_name	Fill material name.
frequency	Frequency variable.
material	Material.
material_name	Material name.
name	Layer name.
number	Layer ID.
stackup	Stackup.
thickness	Thickness variable.
thickness_value	Thickness value.
type	Layer type.

pyaedt.modeler.advanced_cad.stackup_3d.Padstack

```
class pyaedt.modeler.advanced_cad.stackup_3d.Padstack(app, stackup, name, material='copper')
```

Provides the Padstack class member of Stackup3D.

```
__init__(app, stackup, name, material='copper')
```

Methods

<code>add_via([position_x, position_y, ...])</code>	Insert a new via on this padstack.
<code>set_all_antipad_value(value)</code>	Set all antipads in all layers to a specified value.
<code>set_all_pad_value(value)</code>	Set all pads in all layers to a specified value.
<code>set_start_layer(layer)</code>	Set the start layer to a specified value.
<code>set_stop_layer(layer)</code>	Set the stop layer to a specified value.

Attributes

<code>num_sides</code>	Number of sides on the circle, which is 0 for a true circle.
<code>padstacks_by_layer</code>	Get the padstack definitions by layers.
<code>plating_ratio</code>	Plating ratio between 0 and 1.

pyaedt.modeler.advanced_cad.stackup_3d.PadstackLayer

```
class pyaedt.modeler.advanced_cad.stackup_3d.PadstackLayer(padstack, layer_name, elevation, thickness)
```

Provides a data class for the definition of a padstack layer and relative pad and antipad values.

```
__init__(padstack, layer_name, elevation, thickness)
```

Attributes

<code>antipad_radius</code>	Antipad radius on the specified layer.
<code>layer_name</code>	Padstack instance layer.
<code>pad_radius</code>	Pad radius on the specified layer.

pyaedt.modeler.advanced_cad.stackup_3d.Patch

```
class pyaedt.modeler.advanced_cad.stackup_3d.Patch(application, frequency, dx, signal_layer,
                                                    dielectric_layer, dy=None, patch_position_x=0,
                                                    patch_position_y=0, patch_name='patch',
                                                    reference_system=None, axis='X')
```

Patch Class in Stackup3D. Create a parametrized patch. It is preferable to use the add_patch method in the class Layer3D than directly the class constructor.

Parameters**application**

[pyaedt.hfss.Hfss] HFSS design or project where the variable is to be created.

frequency

[float, None] Target resonant frequency for the patch antenna. The default is None, in which case the patch frequency is that of the layer or of the stackup.

dx

[float] The patch width.

signal_layer

[pyaedt.modeler.stackup_3d.Layer3D] The signal layer where the patch will be drawn.

dielectric_layer

[pyaedt.modeler.stackup_3d.Layer3D] The dielectric layer between the patch and the ground layer. Its permittivity and thickness are used in prediction formulas.

dy

[float, None, optional] The patch length. By default, it is None and so the length is calculated by prediction formulas.

patch_position_x

[float, optional] Patch x position, by default it is 0.

patch_position_y

[float, optional] Patch y position, by default it is 0.

patch_name

[str, optional] Patch name, by default patch.

reference_system

[str, None, optional] Coordinate system of the patch. By default, None.

axis

[str, optional] Patch length axis, by default X.

Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modeler.advanced_cad.stackup_3d import Stackup3D
>>> hfss = Hfss()
>>> stackup = Stackup3D(hfss)
>>> gnd = stackup.add_ground_layer("ground", material="copper", thickness=0.035, +_
> fill_material="air")
>>> dielectric = stackup.add_dielectric_layer("dielectric", thickness="0.5" +_
> length_units, material="Duroid (tm)")
```

(continues on next page)

(continued from previous page)

```
>>> signal = stackup.add_signal_layer("signal", material="copper", thickness=0.035, ↵
    ↵fill_material="air")
>>> patch = signal.add_patch(patch_length=9.57, patch_width=9.25, patch_name="Patch" ↵
    ↵")
>>> stackup.resize_around_element(patch)
>>> pad_length = [3, 3, 3, 3, 3, 3] # Air bounding box buffer in mm.
>>> region = hfss.modeler.create_region(pad_length, is_percentage=False)
>>> hfss.assign_radiation_boundary_to_objects(region)
>>> patch.create_probe_port(gnd, rel_x_offset=0.485)
```

```
__init__(application, frequency, dx, signal_layer, dielectric_layer, dy=None, patch_position_x=0,
patch_position_y=0, patch_name='patch', reference_system=None, axis='X')
```

Methods

create_lumped_port(reference_layer[...])	Create a parametrized lumped port.
create_probe_port(reference_layer[...])	Create a coaxial probe port for the patch.
quarter_wave_feeding_line([impedance_to_adap	Create a Trace to feed the patch.
set_optimal_width()	Set the expression of the NamedVariable corresponding to the patch width, to an optimal expression.

Attributes

added_length	Added length calculation.
aedt_object	PyAEDT object 3D.
application	App object.
dielectric_layer	Dielectric layer that the object belongs to.
effective_permittivity	Effective permittivity.
frequency	Model frequency.
impedance	Impedance.
layer_name	Layer name.
layer_number	Layer ID.
length	Length.
material_name	Material name.
name	Object name.
permittivity	Permittivity.
points_on_layer	Object bounding box.
position_x	Starting position X.
position_y	Starting position Y.
reference_system	Coordinate system of the object.
signal_layer	Signal layer that the object belongs to.
substrate_thickness	Substrate thickness.
wave_length	Wave length.
width	Width.

pyaedt.modeler.advanced_cad.stackup_3d.Trace

```
class pyaedt.modeler.advanced_cad.stackup_3d.Trace(application, frequency, line_width,
                                                    line_impedance, signal_layer, dielectric_layer,
                                                    line_electrical_length=90, line_length=None,
                                                    line_position_x=0, line_position_y=0,
                                                    line_name='line', reference_system=None,
                                                    axis='X')
```

Trace Class in Stackup3D. Create a parametrized trace. It is preferable to use the add_trace method in the class Layer3D than directly the class constructor.

Parameters**application**

[pyaedt.hfss.Hfss] HFSS design or project where the variable is to be created.

frequency

[float, None] The line frequency, it is used in prediction formulas. If it is None, the line frequency will be that of the layer or of the stackup.

line_width

[float, None] The line width. If it is None, it will calculate it from characteristic impedance of the line.

line_impedance

[float] The characteristic impedance of the line. If a line width is entered by the user, the characteristic impedance will be calculated from it.

signal_layer

[pyaedt.modeler.stackup_3d.Layer3D] The signal layer where the line will be drawn.

dielectric_layer

[pyaedt.modeler.stackup_3d.Layer3D] The dielectric layer between the line and the ground layer. Its permittivity and thickness are used in prediction formulas.

line_electrical_length

[float, None, optional] The ratio between the line length and the wavelength in degree. By default 90 which is corresponding to the quarter of the wavelength. If it is None, it will be directly calculated from the line length entered by the user.

line_length

[float, None, optional] The line length. By default, it is None and so the length is calculated by prediction formulas according to the electrical length.

line_position_x

[float, optional] Line x position, by default it is 0.

line_position_y

[float, optional] Line y position, by default it is 0.

line_name

[str, optional] Line name, by default line.

reference_system

[str, None, optional] Coordinate system of the line. By default, None.

axis

[str, optional] Line length axis, by default X.

Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modeler.stackup_3d import Stackup3D
>>> hfss = Hfss(new_desktop_session=True)
>>> my_stackup = Stackup3D(hfss, 2.5e9)
>>> gnd = my_stackup.add_ground_layer("gnd")
>>> my_stackup.add_dielectric_layer("diell1", thickness=1.5, material="Duroid (tm)")
>>> top = my_stackup.add_signal_layer("top")
>>> my_trace = top.add_trace(line_width=2.5, line_length=22)
>>> my_stackup.resize_around_element(my_trace)
```

```
__init__(application, frequency, line_width, line_impedance, signal_layer, dielectric_layer,
line_electrical_length=90, line_length=None, line_position_x=0, line_position_y=0,
line_name='line', reference_system=None, axis='X')
```

Methods

create_lumped_port(reference_layer[, ...])	Create a parametrized lumped port.
--	------------------------------------

Attributes

added_length	Added Length.
aedt_object	PyAEDT object 3D.
application	App object.
charac_impedance	Characteristic Impedance.
dielectric_layer	Dielectric layer that the object belongs to.
effective_permittivity	Effective Permittivity.
effective_permittivity_h_w	Effective Permittivity when dielectric thickness is upper than width.
effective_permittivity_w_h	Effective Permittivity when width is upper than dielectric thickness.
electrical_length	Electrical Length.
frequency	Frequency.
layer_name	Layer name.
layer_number	Layer ID.
length	Length.
material_name	Material name.
name	Object name.
permittivity	Permittivity.
points_on_layer	Object bounding box.
position_x	Starting Position X.
position_y	Starting Position Y.
reference_system	Coordinate system of the object.
signal_layer	Signal layer that the object belongs to.
substrate_thickness	Substrate Thickness.
wave_length	Wave Length.
width	Width.
width_h_w	Width when the substrat thickness is two times upper than the width.
width_w_h	Width when the width is two times upper than substrat thickness.

pyaedt.modeler.advanced_cad.stackup_3d.Polygon

```
class pyaedt.modeler.advanced_cad.stackup_3d.Polygon(application, point_list, signal_layer,  
                                                    poly_name='poly', mat_name='copper',  
                                                    is_void=False, reference_system=None)
```

Polygon Class in Stackup3D. It is preferable to use the add_polygon method in the class Layer3D than directly the class constructor.

Parameters

application

[pyaedt.hfss.Hfss] HFSS design or project where the variable is to be created.

point_list

[list] Points list of [x,y] coordinates.

signal_layer

[pyaedt.modeler.stackup_3d.Layer3D] The signal layer where the line will be drawn.

poly_name

[str, optional] Polygon name. The default is poly.

mat_name

[`str`, optional] The polygon material name.

is_void

[`bool`, optional] Whether the polygon is a void. The default is `False`. On ground layers, it will act opposite of the Boolean value because the ground is negative.

reference_system

[`str`, `None`, optional] Coordinate system of the polygon. By default, `None`.

Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modeler.stackup_3d import Stackup3D
>>> hfss = Hfss(new_desktop_session=True)
>>> my_stackup = Stackup3D(hfss, 2.5e9)
>>> gnd = my_stackup.add_ground_layer("gnd", thickness=None)
>>> my_stackup.add_dielectric_layer("diel1", thickness=1.5, material="Duroid (tm)")
>>> top = my_stackup.add_signal_layer("top", thickness=None)
>>> my_polygon = top.add_polygon([[0, 0], [0, 1], [1, 1], [1, 0]])
>>> my_stackup.dielectric_x_position = "2mm"
>>> my_stackup.dielectric_y_position = "2mm"
>>> my_stackup.dielectric_length = "-3mm"
>>> my_stackup.dielectric_width = "-3mm"
```

```
__init__(application, point_list, signal_layer, poly_name='poly', mat_name='copper', is_void=False, reference_system=None)
```

Attributes

aedt_object	PyAEDT object 3D.
application	App object.
dielectric_layer	Dielectric layer that the object belongs to.
layer_name	Layer name.
layer_number	Layer ID.
material_name	Material name.
name	Object name.
points_on_layer	Object Bounding Box.
reference_system	Coordinate system of the object.
signal_layer	Signal layer that the object belongs to.

pyaedt.modeler.advanced_cad.stackup_3d.NamedVariable

```
class pyaedt.modeler.advanced_cad.stackup_3d.NamedVariable(application, name, expression)
```

Cast PyAEDT variable object to simplify getters and setters in Stackup3D.

Parameters

application

[`pyaedt.hfss.Hfss`] HFSS design or project where the variable is to be created.

name

[str] The name of the variable. If the name begins with an \$, the variable will be a project variable. Otherwise, it will be a design variable.

expression

[str] Expression of the value.

Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modeler.stackup_3d import NamedVariable
>>> hfss = Hfss()
>>> my_frequency = NamedVariable(hfss, "my_frequency", "900000Hz")
>>> wave_length_formula = "c0/" + my_frequency.name
>>> my_wave_length = NamedVariable(hfss, "my_wave_length", wave_length_formula)
>>> my_permittivity = NamedVariable(hfss, "my_permittivity", "2.2")
>>> my_wave_length.expression = my_wave_length.expression + "/" + my_permittivity.
˓→name
```

[__init__\(application, name, expression\)](#)

Methods

<u>hide_variable([value])</u>	Set the variable to a hidden variable.
<u>read_only_variable([value])</u>	Set the variable to a read-only variable.

Attributes

<u>evaluated_value</u>	String that combines the numeric value and the units.
<u>expression</u>	Expression of the variable as a string.
<u>name</u>	Name of the variable as a string.
<u>numeric_value</u>	Numeric part of the expression as a float value.
<u>unit_system</u>	Unit system of the expression as a string.
<u>units</u>	Units.
<u>value</u>	Value.

2.6 Modeler in HFSS 3D Layout

This section lists the core AEDT Modeler modules available in HFSS 3D Layout:

- Modeler
- Primitives
- Objects

They are accessible through the `modeler` module and `modeler.objects` property:

```
from pyaedt import Hfss3dLayout
hfss = Hfss3dLayout()
my_modeler = hfss.modeler

...
```

2.6.1 Modeler

The Modeler module contains all properties and methods needed to edit a modeler, including all primitives methods and properties:

- Modeler3DLayout for HFSS 3D Layout

<code>modelerpcb.Modeler3DLayout</code>	Manages Modeler 3D layouts.
---	-----------------------------

`pyaedt.modeler.modelerpcb.Modeler3DLayout`

`class pyaedt.modeler.modelerpcb.Modeler3DLayout(app)`

Manages Modeler 3D layouts. This class is inherited in the caller application and is accessible through the modeler variable object (for example, `hfss3dlayout.modeler`).

Parameters

`app`
`[pyaedt.application.Analysis3DLayout.FieldAnalysis3DLayout]` Inherited parent object.

Examples

```
>>> from pyaedt import Hfss3dLayout
>>> hfss = Hfss3dLayout()
>>> my_modeler = hfss.modeler
```

`__init__(app)`

Methods

<code>change_clip_plane_position(name, location)</code>	Change the clip plane position.
<code>change_net_visibility([assignment, visible])</code>	Change the visibility of one or more nets.
<code>change_property(assignment, name, value[, ...])</code>	Change an oeditor property.
<code>cleanup_objects()</code>	Clean up all 3D Layout geometries (circle, rectangles, polygons, lines and voids) that have been added or no longer exist in the modeler because they were removed by previous operations.
<code>clip_plane()</code>	Create a clip plane in the layout.
<code>colinear_heal(assignment[, tolerance])</code>	Remove small edges of one or more primitives.
<code>convert_to_selections(assignment[, turn_list])</code>	Convert one or more object to selections.

continues on next page

Table 36 – continued from previous page

<code>create_circle(layer, x, y, radius[, name, net])</code>	Create a circle on a layer.
<code>create_component_on_pins(pins[, ...])</code>	Create a component based on a pin list.
<code>create_line(layer, center_line_coordinates)</code>	Create a line based on a list of points.
<code>create_polygon(layer, point_list[, units, ...])</code>	Create a polygon on a specified layer.
<code>create_polygon_void(layer, points, assignment)</code>	Create a polygon void on a specified layer.
<code>create_rectangle(layer, origin, sizes[, ...])</code>	Create a rectangle on a layer.
<code>create_text(text, position[, layer, angle, ...])</code>	Create a text primitive object.
<code>create_via([padstack, x, y, rotation, ...])</code>	Create a via based on an existing padstack.
<code>duplicate(assignment, count, vector)</code>	Duplicate one or more elements along a vector.
<code>duplicate_across_layers(assignment, layers)</code>	Duplicate one or more elements along a vector.
<code>expand(assignment[, size, expand_type, ...])</code>	Expand the object by a specific size.
<code>fit_all()</code>	Fit all.
<code>geometry_check_and_fix_all([min_area])</code>	Run Geometry Check.
<code>import_cadence_brd(input_file[, output_dir, ...])</code>	Import a cadence board.
<code>import_ipc2581(input_file[, output_dir, name])</code>	Import an IPC file.
<code>intersect(assignment)</code>	Intersect objects from names.
<code>merge_design([merged_design, x, y, z, rotation])</code>	Merge a design into another.
<code>modeler_variable(value)</code>	Retrieve a modeler variable.
<code>new_padstack([name])</code>	Create a <i>Padstack</i> object that can be used to create a padstack.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>objects_by_layer(layer[, object_filter, ...])</code>	Retrieve the list of objects that belongs to a specific layer.
<code>objects_by_net(net[, object_filter, ...])</code>	Retrieve the list of objects that belongs to a specific net.
<code>obounding_box(assignment)</code>	Bounding box of a specified object.
<code>place_3d_component(component_path[, ...])</code>	Place an HFSS 3D component in HFSS 3D Layout.
<code>set_spice_model(assignment, input_file[, ...])</code>	Assign a Spice model to a component.
<code>set_temperature_dependence([...])</code>	Set the temperature dependence for the design.
<code>subtract(blank, tool)</code>	Subtract objects from one or more names.
<code>unite(assignment)</code>	Unite objects from names.

Attributes

<code>Padstack</code>	Padstack.
<code>circle_names</code>	Get the list of all rectangles in layout.
<code>circle_voids_names</code>	Get the list of all void circles in layout.
<code>circles</code>	Circles.
<code>circles_voids</code>	Void Circles.
<code>clip_planes</code>	All available clip planes.
<code>components</code>	Components.
<code>components_3d</code>	Components.
<code>defaultmaterial</code>	Default materials.
<code>edb</code>	EBD.
<code>geometries</code>	All Geometries including voids.
<code>line_names</code>	Get the list of all lines in layout.
<code>line_voids_names</code>	Get the list of all void lines in layout.
<code>lines</code>	Lines.
<code>lines_voids</code>	Void Lines.
<code>logger</code>	Logger.

continues on next page

Table 37 – continued from previous page

model_units	Model units as a string (for example, "mm").
modeler	Modeler.
nets	Nets.
no_nets	Nets without class type.
o_component_manager	Component manager object.
o_def_manager	AEDT Definition manager.
o_model_manager	Model manager object.
oeditor	Oeditor Module.
opadstackmanager	Aedt oPadstackManager.
padstacks	Read all definitions from HFSS 3D Layout.
pins	Pins.
polygon_names	Get the list of all polygons in layout.
polygon_voids_names	Get the list of all void polygons in layout.
polygons	Polygons.
polygons_voids	Void Polygons.
power_nets	Nets.
projdir	Project directory.
rectangle_names	Get the list of all rectangles in layout.
rectangle_void_names	Get the list of all void rectangles in layout.
rectangles	Rectangles.
rectangles_voids	Void Rectangles.
signal_nets	Nets.
stackup	Get the Stackup class and its methods.
version	AEDT version.
via_names	Get the list of all vias in layout.
vias	Vias.
voids	All voids.

2.6.2 Objects in HFSS 3D Layout

The following classes define the object properties for HFSS 3D Layout. They contain all getters and setters to simplify object manipulation.

<code>object3dlayout.Components3DLayout</code>	Contains components in HFSS 3D Layout.
<code>object3dlayout.Nets3DLayout</code>	Contains Nets in HFSS 3D Layout.
<code>object3dlayout.Pins3DLayout</code>	Contains the pins in HFSS 3D Layout.
<code>object3dlayout.Line3dLayout</code>	Manages Hfss 3D Layout lines.
<code>object3dlayout.Polygons3DLayout</code>	Manages Hfss 3D Layout polygons.
<code>object3dlayout.Circle3dLayout</code>	Manages Hfss 3D Layout circles.
<code>object3dlayout.Rect3dLayout</code>	Manages Hfss 3D Layout rectangles.
<code>object3dlayout.Points3dLayout</code>	Manages HFSS 3D Layout points.
<code>object3dlayout.Padstack</code>	Manages properties of a padstack.

pyaedt.modeler.pcb.object3dlayout.Components3DLayout

```
class pyaedt.modeler.pcb.object3dlayout.Components3DLayout(primitives, name='', edb_object=None)
    Contains components in HFSS 3D Layout.

    __init__(primitives, name='', edb_object=None)
```

Methods

change_property(value[, names])	Modify a property.
create_clearance_on_component([...])	Create a Clearance on Soldermask layer by drawing a rectangle.
set_die_type([die_type, orientation, ...])	Set the die type.
set_property_value(name, value)	Set a property value.
set_solderball([solderball_type, diameter, ...])	Set solderball on the active component.

Attributes

absolute_angle	Get the absolute angle location for 2 pins components.
angle	Get/Set the circle radius.
bounding_box	Get component bounding box.
die_enabled	Check if the die is enabled.
die_type	Die type.
enabled	Enable or Disable the RLC Component.
location	Retrieve/Set the absolute location in model units.
lock_position	Get/Set the lock position.
model	RLC model if available.
net_name	Get/Set the net name.
object_units	Object units.
part	Retrieve the component part.
part_type	Retrieve the component part type.
pins	Component pins.
placement_layer	Get/Set the placement layer of the object.
solderball_enabled	Check if solderball is enabled.

pyaedt.modeler.pcb.object3dlayout.Nets3DLayout

```
class pyaedt.modeler.pcb.object3dlayout.Nets3DLayout(primitives, name='')
    Contains Nets in HFSS 3D Layout.

    __init__(primitives, name='')
```

Attributes

<code>components</code>	Components that belongs to the Nets.
-------------------------	--------------------------------------

`pyaedt.modeler.pcb.object3dlayout.Pins3DLayout`

```
class pyaedt.modeler.pcb.object3dlayout.Pins3DLayout(primitives, name='', component_name=None,
is_pin=True)
```

Contains the pins in HFSS 3D Layout.

```
__init__(primitives, name='', component_name=None, is_pin=True)
```

Methods

<code>change_property(value[, names])</code>	Modify a property.
<code>create_clearance_on_component([...])</code>	Create a Clearance on Soldermask layer by drawing a rectangle.
<code>set_property_value(name, value)</code>	Set a property value.

Attributes

<code>absolute_angle</code>	Get the absolute angle location for 2 pins components.
<code>angle</code>	Get/Set the circle radius.
<code>bounding_box</code>	Get component bounding box.
<code>holediam</code>	Retrieve the hole diameter of the pin.
<code>location</code>	Retrieve/Set the absolute location in model units.
<code>lock_position</code>	Get/Set the lock position.
<code>net_name</code>	Get/Set the net name.
<code>object_units</code>	Object units.
<code>placement_layer</code>	Get/Set the placement layer of the object.
<code>start_layer</code>	Retrieve the starting layer of the pin.
<code>stop_layer</code>	Retrieve the starting layer of the pin.

`pyaedt.modeler.pcb.object3dlayout.Line3dLayout`

```
class pyaedt.modeler.pcb.object3dlayout.Line3dLayout(primitives, name, is_void=False)
```

Manages Hfss 3D Layout lines.

```
__init__(primitives, name, is_void=False)
```

Methods

add(point[, position])	Add a new point to the center line.
change_property(value[, names])	Modify a property.
create_clearance_on_component([...])	Create a Clearance on Soldermask layer by drawing a rectangle.
edge_by_point(point)	Return the closest edge to specified point.
get_property_value(name)	Retrieve a property value.
remove(point)	Remove one or more points from the center line.
set_property_value(name, value)	Set a property value.

Attributes

absolute_angle	Get the absolute angle location for 2 pins components.
angle	Get/Set the circle radius.
bend_type	Get/Set the line bend type.
bottom_edge_x	Compute the lower edge in the layout on x direction.
bottom_edge_y	Compute the lower edge in the layout on y direction.
bounding_box	Get component bounding box.
center_line	Get the center line points and arc height.
edges	Edges list.
end_cap_type	Get/Set the line end type.
is_closed	Either if the Geometry is closed or not.
length	Get the line length.
location	Retrieve/Set the absolute location in model units.
lock_position	Get/Set the lock position.
name	Name of Primitive.
negative	Get/Set the negative.
net_name	Get/Set the net name.
object_units	Object units.
obounding_box	Bounding box of a specified object.
placement_layer	Get/Set the placement layer of the object.
points	Provide the polygon points.
start_cap_type	Get/Set the line start type.
top_edge_x	Compute the upper edge in the layout on x direction.
top_edge_y	Compute the upper edge in the layout on y direction.
width	Get/Set the line width.

pyaedt.modelerpcb.object3dlayout.Polygons3DLayout

```
class pyaedt.modelerpcb.object3dlayout.Polygons3DLayout(primitives, name, prim_type='poly',  
is_void=False)
```

Manages Hfss 3D Layout polygons.

```
__init__(primitives, name, prim_type='poly', is_void=False)
```

Methods

<code>change_property(value[, names])</code>	Modify a property.
<code>create_clearance_on_component([...])</code>	Create a Clearance on Soldermask layer by drawing a rectangle.
<code>edge_by_point(point)</code>	Return the closest edge to specified point.
<code>get_property_value(name)</code>	Retrieve a property value.
<code>set_property_value(name, value)</code>	Set a property value.

Attributes

<code>absolute_angle</code>	Get the absolute angle location for 2 pins components.
<code>angle</code>	Get/Set the circle radius.
<code>bottom_edge_x</code>	Compute the lower edge in the layout on x direction.
<code>bottom_edge_y</code>	Compute the lower edge in the layout on y direction.
<code>bounding_box</code>	Get component bounding box.
<code>edges</code>	Edges list.
<code>is_closed</code>	Either if the Geometry is closed or not.
<code>location</code>	Retrieve/Set the absolute location in model units.
<code>lock_position</code>	Get/Set the lock position.
<code>name</code>	Name of Primitive.
<code>negative</code>	Get/Set the negative.
<code>net_name</code>	Get/Set the net name.
<code>object_units</code>	Object units.
<code>obounding_box</code>	Bounding box of a specified object.
<code>placement_layer</code>	Get/Set the placement layer of the object.
<code>points</code>	Provide the polygon points.
<code>polygon_voids</code>	All Polygon Voids.
<code>top_edge_x</code>	Compute the upper edge in the layout on x direction.
<code>top_edge_y</code>	Compute the upper edge in the layout on y direction.

`pyaedt.modeler.pcb.object3dlayout.Circle3dLayout`

```
class pyaedt.modeler.pcb.object3dlayout.Circle3dLayout(primitives, name, is_void=False)
```

Manages Hfss 3D Layout circles.

```
__init__(primitives, name, is_void=False)
```

Methods

change_property(value[, names])	Modify a property.
create_clearance_on_component([...])	Create a Clearance on Soldermask layer by drawing a rectangle.
edge_by_point(point)	Return the closest edge to specified point.
get_property_value(name)	Retrieve a property value.
set_property_value(name, value)	Set a property value.

Attributes

absolute_angle	Get the absolute angle location for 2 pins components.
angle	Get/Set the circle radius.
bottom_edge_x	Compute the lower edge in the layout on x direction.
bottom_edge_y	Compute the lower edge in the layout on y direction.
bounding_box	Get component bounding box.
center	Get/Set the circle center.
edges	Edges list.
is_closed	Either if the Geometry is closed or not.
location	Retrieve/Set the absolute location in model units.
lock_position	Get/Set the lock position.
name	Name of Primitive.
negative	Get/Set the negative.
net_name	Get/Set the net name.
object_units	Object units.
obounding_box	Bounding box of a specified object.
placement_layer	Get/Set the placement layer of the object.
points	Provide the polygon points.
radius	Get/Set the circle radius.
top_edge_x	Compute the upper edge in the layout on x direction.
top_edge_y	Compute the upper edge in the layout on y direction.

pyaedt.modeler.pcb.object3dlayout.Rect3dLayout

```
class pyaedt.modeler.pcb.object3dlayout.Rect3dLayout(primitives, name, is_void=False)
```

Manages Hfss 3D Layout rectangles.

```
__init__(primitives, name, is_void=False)
```

Methods

<code>change_property(value[, names])</code>	Modify a property.
<code>create_clearance_on_component([...])</code>	Create a Clearance on Soldermask layer by drawing a rectangle.
<code>edge_by_point(point)</code>	Return the closest edge to specified point.
<code>get_property_value(name)</code>	Retrieve a property value.
<code>set_property_value(name, value)</code>	Set a property value.

Attributes

<code>absolute_angle</code>	Get the absolute angle location for 2 pins components.
<code>angle</code>	Get/Set the circle radius.
<code>bottom_edge_x</code>	Compute the lower edge in the layout on x direction.
<code>bottom_edge_y</code>	Compute the lower edge in the layout on y direction.
<code>bounding_box</code>	Get component bounding box.
<code>center</code>	Get/Set the rectangle center.
<code>corner_radius</code>	Get/Set the circle radius.
<code>edges</code>	Edges list.
<code>height</code>	Get/Set the circle radius.
<code>is_closed</code>	Either if the Geometry is closed or not.
<code>location</code>	Retrieve/Set the absolute location in model units.
<code>lock_position</code>	Get/Set the lock position.
<code>name</code>	Name of Primitive.
<code>negative</code>	Get/Set the negative.
<code>net_name</code>	Get/Set the net name.
<code>object_units</code>	Object units.
<code>obounding_box</code>	Bounding box of a specified object.
<code>placement_layer</code>	Get/Set the placement layer of the object.
<code>point_a</code>	Get/Set the Point A value if 2Point Description is enabled.
<code>point_b</code>	Get/Set the Point B value if 2Point Description is enabled.
<code>points</code>	Provide the polygon points.
<code>top_edge_x</code>	Compute the upper edge in the layout on x direction.
<code>top_edge_y</code>	Compute the upper edge in the layout on y direction.
<code>two_point_description</code>	Get/Set the circle radius.
<code>width</code>	Get/Set the circle radius.

pyaedt.modeler.pcb.object3dlayout.Points3dLayout

```
class pyaedt.modeler.pcb.object3dlayout.Points3dLayout(primitives, point)
    Manages HFSS 3D Layout points.

    __init__(primitives, point)
```

Methods

move(location)	Move actual point to new location.
----------------	------------------------------------

Attributes

is_arc	Either if the Point is an arc or not.
position	Points x and y coordinate.

pyaedt.modeler.pcb.object3dlayout.Padstack

```
class pyaedt.modeler.pcb.object3dlayout.Padstack(name='Padstack', padstackmanager=None,
                                                units='mm')
```

Manages properties of a padstack.

Parameters

name
[str, optional] Name of the padstack. The default is "Padstack".

padstackmanager
[optional] The default is None.

units
[str, optional] The default is mm.

```
__init__(name='Padstack', padstackmanager=None, units='mm')
```

Methods

add_hole([hole_type, sizes, x, y, rotation])	Add a hole.
add_layer([layer, pad_hole, antipad_hole, ...])	Create a layer in the padstack.
create()	Create a padstack in AEDT.
remove()	Remove the padstack in AEDT.
update()	Update the padstack in AEDT.

Attributes

pads_args	Pad properties.
-----------	-----------------

```
from pyaedt import Hfss3dLayout
app = Hfss3dLayout(specified_version="2023.1",
                    non_graphical=False, new_desktop_session=True,
                    close_on_exit=True, student_version=False)

# This call returns the Modeler3DLayout class
modeler = app.modeler

# This call returns a Primitives3D object
primitives = modeler

# This call returns an Object3d object
my_rect = primitives.create_rectangle([0,0,0],[10,10])

# Getter and setter
my_rect.material_name

...
```

2.7 Modeler and components Circuit

This section lists the core AEDT Modeler modules:

- Modeler
- Primitives
- Objects

They are accessible through the `modeler` module and `modeler.objects` property:

```
from pyaedt import TwinBuilder
app = TwinBuilder(specified_version="2023.1",
                  non_graphical=False, new_desktop_session=True,
                  close_on_exit=True, student_version=False)

# This call returns the Modeler class
modeler = app.modeler

...
```

2.7.1 Modeler

The Modeler module contains all properties and methods needed to edit a modeler, including all primitives methods and properties:

- `ModelerNexxim` for Circuit
- `ModelerTwinBuilder` for Twin Builder
- `ModelerEmit` for EMIT

<code>schematic.ModelerNexxim</code>	ModelerNexxim class.
<code>schematic.ModelerTwinBuilder</code>	ModelerTwinBuilder class.
<code>schematic.ModelerEmit</code>	ModelerEmit class.
<code>schematic.ModelerMaxwellCircuit</code>	ModelerMaxwellCircuit class.

`pyaedt.modeler.schematic.ModelerNexxim`

`class pyaedt.modeler.schematic.ModelerNexxim(app)`

ModelerNexxim class.

Parameters

`app`
[pyaedt.application.AnalysisNexxim.FieldAnalysisCircuit]
`__init__(app)`

Methods

<code>change_text_property(assignment, name, value)</code>	Change an oeditor property.
<code>connect_schematic_components(...[, ...])</code>	Connect schematic components.
<code>create_text(text[, x_origin, y_origin, ...])</code>	Draw Text.
<code>move(assignment, offset[, units])</code>	Move the selections by the specified [x, y] coordinates.
<code>rotate(assignment[, degrees])</code>	Rotate the selections by degrees.
<code>zoom_to_fit()</code>	Zoom To Fit.

Attributes

<code>edb</code>	EDB.
<code>layout</code>	Primitives.
<code>layoutheditor</code>	Return the Circuit Layout Editor.
<code>logger</code>	Logger.
<code>model_units</code>	Layout model units.
<code>o_component_manager</code>	Component manager object.
<code>o_def_manager</code>	AEDT Definition manager.
<code>o_model_manager</code>	Model manager object.
<code>oeditor</code>	Oeditor Module.
<code>projdir</code>	Project directory.
<code>schematic</code>	Schematic Component.
<code>schematic_units</code>	Schematic units.

`pyaedt.modeler.schematic.ModelerTwinBuilder`

```
class pyaedt.modeler.schematic.ModelerTwinBuilder(app)
```

ModelerTwinBuilder class.

Parameters

<code>app</code>	[pyaedt.application.AnalysisTwinBuilder.AnalysisTwinBuilder]
<code>__init__(app)</code>	

Methods

<code>change_text_property(assignment, name, value)</code>	Change an oeditor property.
<code>connect_schematic_components(...[, ...])</code>	Connect schematic components.
<code>create_text(text[, x_origin, y_origin, ...])</code>	Draw Text.
<code>zoom_to_fit()</code>	Zoom To Fit.

Attributes

<code>logger</code>	Logger.
<code>o_component_manager</code>	Component manager object.
<code>o_def_manager</code>	AEDT Definition manager.
<code>o_model_manager</code>	Model manager object.
<code>oeditor</code>	Oeditor Module.
<code>projdir</code>	Project directory.
<code>schematic</code>	Schematic Object.
<code>schematic_units</code>	Schematic units.

pyaedt.modeler.schematic.ModelerEmit**class** pyaedt.modeler.schematic.**ModelerEmit**(app)

ModelerEmit class.

Parameters**app**

[pyaedt.application.AnalysisEmit]

__init__(app)**Methods**

change_text_property(assignment, name, value)	Change an oeditor property.
connect_schematic_components(...[, ...])	Connect schematic components.
create_text(text[, x_origin, y_origin, ...])	Draw Text.
zoom_to_fit()	Zoom To Fit.

Attributes

logger	Logger.
o_component_manager	Component manager object.
o_def_manager	AEDT Definition manager.
o_model_manager	Model manager object.
oeditor	Oeditor Module.
projdir	Project directory.
schematic_units	Schematic units.

pyaedt.modeler.schematic.ModelerMaxwellCircuit**class** pyaedt.modeler.schematic.**ModelerMaxwellCircuit**(app)

ModelerMaxwellCircuit class.

Parameters**app**

[pyaedt.application.AnalysisMaxwellCircuit]

__init__(app)

Methods

<code>change_text_property(assignment, name, value)</code>	Change an oeditor property.
<code>connect_schematic_components(...[, ...])</code>	Connect schematic components.
<code>create_text(text[, x_origin, y_origin, ...])</code>	Draw Text.
<code>zoom_to_fit()</code>	Zoom To Fit.

Attributes

<code>logger</code>	Logger.
<code>o_component_manager</code>	Component manager object.
<code>o_def_manager</code>	AEDT Definition manager.
<code>o_model_manager</code>	Model manager object.
<code>oeditor</code>	Oeditor Module.
<code>projdir</code>	Project directory.
<code>schematic</code>	Schematic Object.
<code>schematic_units</code>	Schematic units.

2.7.2 Schematic in Circuit

The following classes define the object properties for Circuit components. They contain all getters and setters to simplify object manipulation.

<code>NexximComponents</code>	Manages circuit components for Nexxim.
-------------------------------	--

`pyaedt.modeler.circuits.PrimitivesNexxim.NexximComponents`

`class pyaedt.modeler.circuits.PrimitivesNexxim.NexximComponents(modeler)`

Manages circuit components for Nexxim.

Parameters

`modeler`
`[pyaedt.modeler.schematic.ModelerNexxim]` Inherited parent object.

Examples

-

```
>>> from pyaedt import Circuit
>>> aedtapp = Circuit()
>>> prim = aedtapp.modeler.schematic
```

`__init__(modeler)`

Methods

<code>add_id_to_component(id)</code>	Add an ID to a component.
<code>add_pin_iports(name, id_num)</code>	Add ports on pins.
<code>add_siwave_dynamic_link(input_file[, ...])</code>	Add a siwave dinamyc link object.
<code>add_subcircuit_3dlayout(name)</code>	Add a subcircuit from a HFSS 3DLayout.
<code>add_subcircuit_dynamic_link([pyaedt_app, ...])</code>	Add a subcircuit from <i>HFSS</i> , <i>Q3d</i> or <i>2D Extractor</i> in circuit design.
<code>connect_components_in_parallel(assignment)</code>	Connect schematic components in parallel.
<code>connect_components_in_series(assignment[, ...])</code>	Connect schematic components in series.
<code>create_capacitor([name, value, location, ...])</code>	Create a capacitor.
<code>create_component([name, component_library, ...])</code>	Create a component from a library.
<code>create_component_from_spicemodel(input_file)</code>	Create and place a new component based on a spice .lib file.
<code>create_coupling_inductors(compname, l1, l2)</code>	Create a coupling inductor.
<code>create_current_dc([name, value, location, ...])</code>	Create a current DC source.
<code>create_current_pulse([name, value_lists, ...])</code>	Create a current pulse.
<code>create_diode([name, model_name, location, ...])</code>	Create a diode.
<code>create_field_model(design_name, ...[, ...])</code>	Create a field model.
<code>create_gnd([location, angle])</code>	Create a ground.
<code>create_inductor([name, value, location, ...])</code>	Create an inductor.
<code>create_interface_port(name[, location, angle])</code>	Create an interface port.
<code>create_line(points[, color, width])</code>	Draw a graphical line.
<code>create_model_from_touchstone(input_file[, ...])</code>	Create a model from a Touchstone file.
<code>create_new_component_from_symbol(name, pins)</code>	Create a component from a symbol.
<code>create_npn([name, value, location, angle, ...])</code>	Create an NPN transistor.
<code>create_page_port(name[, location, angle])</code>	Create a page port.
<code>create_pnp([name, value, location, angle, ...])</code>	Create a PNP transistor.
<code>create_resistor([name, value, location, ...])</code>	Create a resistor.
<code>create_subcircuit([location, angle, name, ...])</code>	Add a new Circuit subcircuit to the design.
<code>create_symbol(name, pins)</code>	Create a symbol.
<code>create_touchstone_component(model_name[, ...])</code>	Create a component from a Touchstone model.
<code>create_unique_id()</code>	Create an unique ID.
<code>create_voltage_dc([name, value, location, ...])</code>	Create a voltage DC source.
<code>create_voltage_probe([name, location, ...])</code>	Create a voltage probe.
<code>create_voltage_pulse([name, value_lists, ...])</code>	Create a voltage pulse.
<code>create_voltage_pwl([name, time_list, ...])</code>	Create a pwl voltage source.
<code>create_wire(points[, name])</code>	Create a wire.
<code>delete_component(name)</code>	Get and delete a component.
<code>disable_data_netlist(assignment)</code>	Disable the Nexxim global net list.
<code>duplicate(assignment[, location, angle, flip])</code>	Add a new subcircuit to the design.
<code>enable_global_netlist(assignment[, ...])</code>	Enable Nexxim global net list.
<code>enable_use_instance_name([...])</code>	Enable the use of the instance name.
<code>get_component(name)</code>	Get a component.
<code>get_obj_id(assignment)</code>	Retrieve the ID of an object.
<code>get_pin_location(assignment, pin)</code>	Retrieve the location of a pin.
<code>get_pins(assignment)</code>	Retrieve one or more pins.
<code>get_wire_by_name(name)</code>	Wire class by name.

continues on next page

Table 38 – continued from previous page

<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>refresh_all_ids()</code>	Refresh all IDs and return the number of components.
<code>refresh_dynamic_link(name)</code>	Refresh a dynamic link component.
<code>set_sim_option_on_hfss_subcircuit(componer)</code>	Set the simulation option on the HFSS subcircuit.
<code>set_sim_solution_on_hfss_subcircuit(componer)</code>	Set the simulation solution on the HFSS subcircuit.

Attributes

<code>components_catalog</code>	System library component catalog with all information.
<code>design_library</code>	Design library.
<code>design_type</code>	Design type.
<code>design_types</code>	Design types.
<code>model_units</code>	Model units.
<code>nets</code>	List of all schematic nets.
<code>o_component_manager</code>	Component manager object.
<code>o_definition_manager</code>	Aedt oDefinitionManager.
<code>o_symbol_manager</code>	Model manager object.
<code>schematic_units</code>	Schematic units.
<code>tab_name</code>	Tab name.
<code>version</code>	Version.
<code>wires</code>	All schematic wires in the design.

```
from pyaedt import Circuit
app = Circuit(specified_version="2023.1",
               non_graphical=False, new_desktop_session=True,
               close_on_exit=True, student_version=False)

# This call returns a Schematic object
schematic = modeler.schematic

# This call returns an Object3d object
my_res = schematic.create_resistor("R1", 50)
```

2.7.3 Objects in Circuit

The following classes define the object properties for Circuit. They contain all getters and setters to simplify object manipulation.

<code>object3dcircuit.CircuitComponent</code>	Manages circuit components.
<code>object3dcircuit.CircuitPins</code>	Manages circuit component pins.
<code>object3dcircuit.Wire</code>	Creates and manipulates a wire.

pyaedt.modeler.circuits.object3dcircuit.CircuitComponent

```
class pyaedt.modeler.circuits.object3dcircuit.CircuitComponent(circuit_components,  
                                                               tabname='PassedParameterTab',  
                                                               custom_editor=None)
```

Manages circuit components.

```
__init__(circuit_components, tabname='PassedParameterTab', custom_editor=None)
```

Methods

change_property(property_name[, names])	Modify a property.
delete()	Delete the component.
enforce_touchstone_model_passive()	Enforce touchstone model passive.
set_color([red, green, blue])	Set symbol color.
set_property(name, value)	Set a part property.
set_use_symbol_color([color])	Set symbol color usage.

Attributes

angle	Get the part angle.
bounding_box	Component bounding box.
component_info	Component parameters.
composed_name	Composed names.
location	Get the part location.
mirror	Get the part mirror.
model_data	Return the model data if the component has one.
model_name	Return Model Name if present.
parameters	Circuit Parameters.
pins	Pins of the component.
refdes	Reference designator.
units	Length units.

pyaedt.modeler.circuits.object3dcircuit.CircuitPins

```
class pyaedt.modeler.circuits.object3dcircuit.CircuitPins(circuit_comp, pinname)
```

Manages circuit component pins.

```
__init__(circuit_comp, pinname)
```

Methods

<code>connect_to_component(assignment[, ...])</code>	Connect schematic components.
--	-------------------------------

Attributes

<code>angle</code>	Pin angle.
<code>location</code>	Pin Position in [x,y] format.
<code>net</code>	Get pin net.
<code>units</code>	Length units.

`pyaedt.modeler.circuits.object3dcircuit.Wire`

```
class pyaedt.modeler.circuits.object3dcircuit.Wire(modeler, composed_name=None)
```

Creates and manipulates a wire.

```
__init__(modeler, composed_name=None)
```

Methods

<code>display_wire_properties([name, ...])</code>	Display wire properties.
---	--------------------------

Attributes

<code>logger</code>	Logger.
<code>points_in_segment</code>	Points in segment.
<code>wires</code>	List of all schematic wires in the design.

```
from pyaedt import Circuit
app = Circuit(specified_version="2023.1",
               non_graphical=False, new_desktop_session=True,
               close_on_exit=True, student_version=False)

# This call returns the Modeler class
modeler = app.modeler

# This call returns a Schematic object
schematic = modeler.schematic

# This call returns an Object3d object
my_res = schematic.create_resistor("R1", 50)

# Getter and setter
my_res.location
```

(continues on next page)

(continued from previous page)

```
my_res.parameters["R"] = 100
```

```
...
```

2.7.4 Schematic in EMIT

The following classes define the object properties for EMIT components. They contain all getters and setters to simplify object manipulation.

<i>EmitComponents</i>	EmitComponents class.
-----------------------	-----------------------

pyaedt.modeler.circuits.PrimitivesEmit.EmitComponents

```
class pyaedt.modeler.circuits.PrimitivesEmit.EmitComponents(parent, modeler)
```

EmitComponents class.

This is the class for managing all EMIT components.

```
__init__(parent, modeler)
```

Methods

<code>create_component(component_type[, name, library])</code>	Create a new component from a library.
<code>create_radio_antenna(radio_type[, ...])</code>	Create a new radio and antenna and connect them.
<code>get_antennas()</code>	Get all antennas in the design.
<code>get_obj_id(object_name)</code>	Get object ID.
<code>get_radios()</code>	Get all radios in the design.
<code>refresh_all_ids()</code>	Refresh all IDs and return the number of components.
<code>update_object_properties(o)</code>	Update the properties of an EMIT component.

Attributes

<code>design_type</code>	Design type.
<code>design_types</code>	Design types.
<code>messenger</code>	Messenger.
<code>model_units</code>	Model units.
<code>o_component_manager</code>	AEDT Component Manager.
<code>o_definition_manager</code>	Aedt Definition Manager.
<code>o_model_manager</code>	Aedt Model Manager.
<code>o_symbol_manager</code>	AEDT Symbol Manager.
<code>odesign</code>	Odesign module.
<code>oeditor</code>	Oeditor Module.
<code>version</code>	Version.

2.7.5 Schematic in Twin Builder

The following classes define the object properties for Twin Builder components. They contain all getters and setters to simplify object manipulation.

TwinBuilderComponents

TwinBuilderComponents class.

pyaedt.modeler.circuits.PrimitivesTwinBuilder.TwinBuilderComponents

class `pyaedt.modeler.circuits.PrimitivesTwinBuilder.TwinBuilderComponents(modeler)`

TwinBuilderComponents class.

This class is for managing all circuit components for Twin Builder.

Parameters

parent
modeler

Examples

Basic usage demonstrated with a Twin Builder design:

```
>>> from pyaedt import TwinBuilder  
>>> aedtapp = TwinBuilder()  
>>> prim = aedtapp.modeler.schematic
```

```
__init__(modeler)
```

Methods

add_id_to_component(id)	Add an ID to a component.
add_pin_iports(name, id_num)	Add ports on pins.
create_capacitor([name, value, location, ...])	Create a capacitor.
create_component([name, component_library, ...])	Create a component from a library.
create_diode([name, location, angle, ...])	Create a diode.
create_gnd([location, angle])	Create a ground.
create_inductor([name, value, location, ...])	Create an inductor.
create_interface_port(name[, location, angle])	Create an interface port.
create_line(points[, color, width])	Draw a graphical line.
create_model_from_touchstone(input_file[, ...])	Create a model from a Touchstone file.
create_npn([name, location, angle, ...])	Create an NPN transistor.
create_page_port(name[, location, angle])	Create a page port.
create_periodic_waveform_source([name, ...])	Create a periodic waveform source (non conservative real output).
create_pnp([name, location, angle, ...])	Create a PNP transistor.
create_resistor([name, value, location, ...])	Create a resistor.
create_symbol(name, pins)	Create a symbol.
create_touchstone_component(model_name[, ...])	Create a component from a Touchstone model.
create_unique_id()	Create an unique ID.
create_voltage_source([name, type, ...])	Create a voltage source (conservative electrical output).
create_wire(points[, name])	Create a wire.
disable_data_netlist(assignment)	Disable the Nexxim global net list.
enable_global_netlist(assignment[, ...])	Enable Nexxim global net list.
enable_use_instance_name([...])	Enable the use of the instance name.
get_obj_id(assignment)	Retrieve the ID of an object.
get_pin_location(assignment, pin)	Retrieve the location of a pin.
get_pins(assignment)	Retrieve one or more pins.
get_wire_by_name(name)	Wire class by name.
number_with_units(value[, units])	Convert a number to a string with units.
refresh_all_ids()	Refresh all IDs and return the number of components.

Attributes

components_catalog	Return the syslib component catalog with all info.
design_library	Design Library.
design_type	Design type.
design_types	Design types.
model_units	Model units.
nets	List of all schematic nets.
o_component_manager	Component manager object.
o_definition_manager	Aedt oDefinitionManager.
o_symbol_manager	Model manager object.
schematic_units	Schematic units.
tab_name	Tab name.
version	Version.
wires	All schematic wires in the design.

```

from pyaedt import TwinBuilder
app = TwinBuilder(specified_version="2023.1",
                   non_graphical=False, new_desktop_session=True,
                   close_on_exit=True, student_version=False)

# This call returns the Modeler class
modeler = app.modeler

# This call returns a Schematic object
schematic = modeler.schematic

# This call returns an Object3d object
my_res = schematic.create_resistor("R1", 50)

# Getter and setter
my_res.location
my_res.parameters["R"] = 100

...

```

2.7.6 Schematic in Maxwell Circuit

The following classes define the object properties for Maxwell Circuit components. They contain all getters and setters to simplify object manipulation.

MaxwellCircuitComponents

MaxwellCircuitComponents class.

pyaedt.modeler.circuits.PrimitivesMaxwellCircuit.MaxwellCircuitComponents

class pyaedt.modeler.circuits.PrimitivesMaxwellCircuit.**MaxwellCircuitComponents**(modeler)

MaxwellCircuitComponents class.

This class is for managing all circuit components for MaxwellCircuit.

Examples

Basic usage demonstrated with a MaxwellCircuit design:

```

>>> from pyaedt import MaxwellCircuit
>>> aedtapp = MaxwellCircuit()
>>> prim = aedtapp.modeler.schematic

```

__init__(modeler)

Methods

<code>add_id_to_component(id)</code>	Add an ID to a component.
<code>add_pin_iports(name, id_num)</code>	Add ports on pins.
<code>create_capacitor([name, value, location, ...])</code>	Create a capacitor.
<code>create_component([name, component_library, ...])</code>	Create a component from a library.
<code>create_diode([name, location, angle, ...])</code>	Create a diode.
<code>create_gnd([location, angle])</code>	Create a ground.
<code>create_inductor([name, value, location, ...])</code>	Create an inductor.
<code>create_interface_port(name[, location, angle])</code>	Create an interface port.
<code>create_line(points[, color, width])</code>	Draw a graphical line.
<code>create_model_from_touchstone(input_file[, ...])</code>	Create a model from a Touchstone file.
<code>create_page_port(name[, location, angle])</code>	Create a page port.
<code>create_resistor([name, value, location, ...])</code>	Create a resistor.
<code>create_symbol(name, pins)</code>	Create a symbol.
<code>create_touchstone_component(model_name[, ...])</code>	Create a component from a Touchstone model.
<code>create_unique_id()</code>	Create an unique ID.
<code>create_winding([name, location, angle, ...])</code>	Create an NPN transistor.
<code>create_wire(points[, name])</code>	Create a wire.
<code>disable_data_netlist(assignment)</code>	Disable the Nexxim global net list.
<code>enable_global_netlist(assignment[, ...])</code>	Enable Nexxim global net list.
<code>enable_use_instance_name([...])</code>	Enable the use of the instance name.
<code>get_obj_id(assignment)</code>	Retrieve the ID of an object.
<code>get_pin_location(assignment, pin)</code>	Retrieve the location of a pin.
<code>get_pins(assignment)</code>	Retrieve one or more pins.
<code>get_wire_by_name(name)</code>	Wire class by name.
<code>number_with_units(value[, units])</code>	Convert a number to a string with units.
<code>refresh_all_ids()</code>	Refresh all IDs and return the number of components.

Attributes

<code>design_library</code>	Design Library.
<code>design_type</code>	Design type.
<code>design_types</code>	Design types.
<code>model_units</code>	Model units.
<code>nets</code>	List of all schematic nets.
<code>o_component_manager</code>	Component manager object.
<code>o_definition_manager</code>	Aedt oDefinitionManager.
<code>o_symbol_manager</code>	Model manager object.
<code>schematic_units</code>	Schematic units.
<code>tab_name</code>	Tab name.
<code>version</code>	Version.
<code>wires</code>	All schematic wires in the design.

2.8 Boundary objects

This section lists classes for creating and editing boundaries in the 3D tools. These objects are returned by app methods and can be used to edit or delete a boundary condition.

<code>NativeComponentObject</code>	Manages Native Component data and execution.
<code>BoundaryObject</code>	Manages boundary data and execution.
<code>BoundaryObject3dLayout</code>	Manages boundary data and execution for Hfss3dLayout.
<code>FarFieldSetup</code>	Manages Far Field Component data and execution.
<code>Matrix</code>	Manages Matrix in Q3d and Q2d Projects.
<code>BoundaryObject3dLayout</code>	Manages boundary data and execution for Hfss3dLayout.
<code>Sources</code>	Manages sources in Circuit projects.
<code>Excitations</code>	Manages Excitations in Circuit Projects.

2.8.1 pyaedt.modules.Boundary.NativeComponentObject

```
class pyaedt.modules.Boundary.NativeComponentObject(app, component_type, component_name, props)
```

Manages Native Component data and execution.

Parameters

app	[object] An AEDT application from pyaedt.application.
component_type	[str] Type of the component.
component_name	[str] Name of the component.
props	[dict] Properties of the boundary.

Examples

in this example the par_beam returned object is a pyaedt.modules.Boundary.NativeComponentObject

```
>>> from pyaedt import Hfss >>> hfss = Hfss(solution_type=SBR+) >>> ffd_file = path/to/ffdfile.ffd >>>
par_beam = hfss.create_sbr_file_based_antenna(ffd_file) >>> par_beam.native_properties[Size] = 0.1mm >>>
par_beam.update() >>> par_beam.delete()
```

```
__init__(app, component_type, component_name, props)
```

Methods

<code>create()</code>	Create a Native Component in AEDT.
<code>delete()</code>	Delete the Native Component in AEDT.
<code>update()</code>	Update the Native Component in AEDT.

Attributes

available_properties	Available properties.
targetcs	Native Component Coordinate System.

2.8.2 pyaedt.modules.Boundary.BoundaryObject

```
class pyaedt.modules.Boundary.BoundaryObject(app, name, props=None, boundarytype=None,  
                                             auto_update=True)
```

Manages boundary data and execution.

Parameters

app	[object] An AEDT application from pyaedt.application.
name	[str] Name of the boundary.
props	[dict, optional] Properties of the boundary.
boundarytype	[str, optional] Type of the boundary.

Examples

Create a cylinder at the XY working plane and assign a copper coating of 0.2 mm to it. The Coating is a boundary operation and coat will return a pyaedt.modules.Boundary.BoundaryObject

```
>>> from pyaedt import Hfss  
>>> hfss = Hfss()  
>>> origin = hfss.modeler.Position(0, 0, 0)  
>>> inner = hfss.modeler.create_cylinder(hfss.PLANE.XY, origin, 3, 200, 0, "inner")  
>>> inner_id = hfss.modeler.get_obj_id("inner",)  
>>> coat = hfss.assign_coating([inner_id], "copper", use_thickness=True, thickness="0.  
→2mm")
```

```
__init__(app, name, props=None, boundarytype=None, auto_update=True)
```

Methods

create()	Create a boundary.
delete()	Delete the boundary.
update()	Update the boundary.
update_assignment()	Update the boundary assignment.

Attributes

<code>available_properties</code>	Available properties.
<code>name</code>	Boundary Name.
<code>object_properties</code>	Object-oriented properties.
<code>props</code>	Boundary data.
<code>type</code>	Boundary type.

2.8.3 pyaedt.modules.Boundary.BoundaryObject3dLayout

```
class pyaedt.modules.Boundary.BoundaryObject3dLayout(app, name, props, boundarytype)
```

Manages boundary data and execution for Hfss3dLayout.

Parameters

app	[object] An AEDT application from pyaedt.application.
name	[str] Name of the boundary.
props	[dict] Properties of the boundary.
boundarytype	[str] Type of the boundary.

```
__init__(app, name, props, boundarytype)
```

Methods

<code>delete()</code>	Delete the boundary.
<code>update()</code>	Update the boundary.

Attributes

<code>available_properties</code>	Available properties.
<code>name</code>	Boundary Name.
<code>object_properties</code>	Object-oriented properties.
<code>props</code>	Excitation data.

2.8.4 pyaedt.modules.Boundary.FarFieldSetup

```
class pyaedt.modules.Boundary.FarFieldSetup(app, component_name, props, component_type,  
                                         units='deg')
```

Manages Far Field Component data and execution.

Examples

in this example the sphere1 returned object is a pyaedt.modules.Boundary.FarFieldSetup >>> from pyaedt import Hfss >>> hfss = Hfss() >>> sphere1 = hfss.insert_infinite_sphere() >>> sphere1.props[ThetaStart] = -90deg >>> sphere1.props[ThetaStop] = 90deg >>> sphere1.props[ThetaStep] = 2deg >>> sphere1.delete()

```
__init__(app, component_name, props, component_type, units='deg')
```

Methods

create()	Create a Field Setup Component in HFSS.
delete()	Delete the Field Setup in AEDT.
update()	Update the Field Setup in AEDT.

Attributes

<code>available_properties</code>	Available properties.
<code>azimuth_start</code>	Set/Get the Far Field Azimuth Start Angle if Definition is Set to <i>Az Over El</i> or <i>El Over Az</i> .
<code>azimuth_step</code>	Set/Get the Far Field Azimuth Step Angle if Definition is Set to <i>Az Over El</i> or <i>El Over Az</i> .
<code>azimuth_stop</code>	Set/Get the Far Field Azimuth Stop Angle if Definition is Set to <i>Az Over El</i> or <i>El Over Az</i> .
<code>custom_radiation_surface</code>	Set/Get the Far Field Radiation Surface FaceList.
<code>definition</code>	Set/Get the Far Field Angle Definition.
<code>elevation_start</code>	Set/Get the Far Field Elevation Start Angle if Definition is Set to <i>Az Over El</i> or <i>El Over Az</i> .
<code>elevation_step</code>	Set/Get the Far Field Elevation Step Angle if Definition is Set to <i>Az Over El</i> or <i>El Over Az</i> .
<code>elevation_stop</code>	Set/Get the Far Field Elevation Stop Angle if Definition is Set to <i>Az Over El</i> or <i>El Over Az</i> .
<code>local_coordinate_system</code>	Set/Get the custom Coordinate System name.
<code>name</code>	Variable name.
<code>phi_start</code>	Set/Get the Far Field Phi Start Angle if Definition is Set to <i>Theta-Phi</i> .
<code>phi_step</code>	Set/Get the Far Field Phi Step Angle if Definition is Set to <i>Theta-Phi</i> .
<code>phi_stop</code>	Set/Get the Far Field Phi Stop Angle if Definition is Set to <i>Theta-Phi</i> .
<code>polarization</code>	Set/Get the Far Field Polarization.
<code>slant_angle</code>	Set/Get the Far Field Slant Angle if Polarization is Set to <i>Slant</i> .
<code>theta_start</code>	Set/Get the Far Field Theta Start Angle if Definition is Set to <i>Theta-Phi</i> .
<code>theta_step</code>	Set/Get the Far Field Theta Step Angle if Definition is Set to <i>Theta-Phi</i> .
<code>theta_stop</code>	Set/Get the Far Field Theta Stop Angle if Definition is Set to <i>Theta-Phi</i> .
<code>use_custom_radiation_surface</code>	Set/Get the Far Field Radiation Surface Enable.
<code>use_local_coordinate_system</code>	Set/Get the usage of a custom Coordinate System.

2.8.5 pyaedt.modules.Boundary.Matrix

```
class pyaedt.modules.Boundary.Matrix(app, name, operations=None)
```

Manages Matrix in Q3d and Q2d Projects.

```
__init__(app, name, operations=None)
```

Methods

add_operation(operation_type[, ...])	Add a new operation to existing matrix.
create([source_names, new_net_name, ...])	Create a new matrix.
delete()	Delete current matrix.
get_sources_for_plot([get_self_terms, ...])	Return a list of source of specified matrix ready to be used in plot reports.
sources([is_gc_sources])	List of matrix sources.

Attributes

operations	List of matrix operations.
------------	----------------------------

2.8.6 pyaedt.modules.Boundary.Sources

```
class pyaedt.modules.Boundary.Sources(app, name, source_type=None)  
    Manages sources in Circuit projects.  
  
    __init__(app, name, source_type=None)
```

Methods

create()	Create a new source in AEDT.
delete()	Delete the source in AEDT.
update([original_name, new_source])	Update the source in AEDT.

Attributes

name	Source name.
------	--------------

2.8.7 pyaedt.modules.Boundary.Excitations

```
class pyaedt.modules.Boundary.Excitations(app, name)  
    Manages Excitations in Circuit Projects.  
  
    __init__(app, name)
```

Methods

<code>delete()</code>	Delete the port in AEDT.
<code>update()</code>	Update the excitation in AEDT.

Attributes

<code>angle</code>	Symbol angle.
<code>enable_noise</code>	Enable noise.
<code>enabled_analyses</code>	Enabled analyses.
<code>enabled_sources</code>	Enabled sources.
<code>impedance</code>	Port termination.
<code>location</code>	Port location.
<code>microwave_symbol</code>	Enable microwave symbol.
<code>mirror</code>	Enable port mirror.
<code>name</code>	Excitation name.
<code>noise_temperature</code>	Enable noise.
<code>reference_node</code>	Reference node.
<code>use_symbol_color</code>	Use symbol color.

Example without Native Component Object:

```
from pyaedt import Icepak
ipk = Icepak()
component_name = "RadioBoard1"
native_comp = self.aedtapp.create_ipk_3dcomponent_pcb(
    component_name, link_data, solution_freq, resolution, custom_x_resolution=400,
    ↴custom_y_resolution=500
)
# native_comp is a NativeComponentObject
...
ipk.release_desktop()
```

2.9 Mesh operations

The Mesh module includes these classes:

- Mesh for HFSS, Maxwell 2D, Maxwell 3D, Q2D Extractor, and Q3D Extractor
- IcepakMesh for Icepak
- Mesh3d for HFSS 3D Layout

They are accessible through the mesh property:

<code>Mesh.Mesh</code>	Manages AEDT mesh functions for 2D and 3D solvers (HFSS, Maxwell, and Q3D).
<code>MeshIcepak.IcepakMesh</code>	Manages Icepak meshes.
<code>Mesh3DLayout.Mesh3d</code>	Manages mesh operations for HFSS 3D Layout.

2.9.1 pyaedt.modules.Mesh.Mesh

```
class pyaedt.modules.Mesh(app)
```

Manages AEDT mesh functions for 2D and 3D solvers (HFSS, Maxwell, and Q3D).

Parameters

```
app  
[pyaedt.application.Analysis3D.FieldAnalysis3D]
```

Examples

Basic usage demonstrated with an HFSS design:

```
>>> from pyaedt import Hfss  
>>> hfss = Hfss()  
>>> cylinder = hfss.modeler.create_cylinder(0, [0, 0, 0], 3, 20, 0)  
>>> model_resolution = hfss.mesh.assign_model_resolution(cylinder, 1e-4, "ModelRes1")
```

```
__init__(app)
```

Methods

assign_curvature_extraction(assignment[, ...])	Assign curvature extraction.
assign_curvilinear_elements(assignment[, ...])	Assign curvilinear elements.
assign_cylindrical_gap(entity[, name, ...])	Assign a cylindrical gap for a 2D or 3D design to enable a clone mesh and associated band mapping angle.
assign_density_control(assignment[, ...])	Assign density control.
assign_edge_cut(assignment[, ...])	Assign an edge cut layer mesh.
assign_initial_mesh_from_slider([level, ...])	Assign a surface mesh level to an object.
assign_length_mesh(assignment[, ...])	Assign a length for the model resolution.
assign_model_resolution(assignment[, ...])	Assign the model resolution.
assign_rotational_layer(assignment[, ...])	Assign a rotational layer mesh.
assign_skin_depth(assignment[, skin_depth, ...])	Assign a skin depth for the mesh refinement.
assign_surf_priority_for_tau(assignment[, ...])	Assign a surface representation priority for the TAU mesh.
assign_surface_mesh(assignment, level[, name])	Assign a surface mesh level to one or more objects.
assign_surface_mesh_manual(assignment[, ...])	Assign a surface mesh to a list of faces.
delete_mesh_operations([mesh_type])	Remove mesh operations from a design.
generate_mesh(name)	Generate the mesh for a design.

Attributes

<code>initial_mesh_settings</code>	Return the global mesh object.
<code>meshoperation_names</code>	Return the available mesh operation names.
<code>meshoperations</code>	Return the available mesh operations.
<code>omeshmodule</code>	Aedt Mesh Module.

2.9.2 pyaedt.modules.MeshIcepak.IcepakMesh

```
class pyaedt.modules.MeshIcepak.IcepakMesh(app)
```

Manages Icepak meshes.

Parameters

<code>app</code>	[pyaedt.application.Analysis3D.FieldAnalysis3D]
<code>__init__(app)</code>	

Methods

<code>assign_mesh_from_file(assignment, file_name)</code>	Assign a mesh from a file to objects.
<code>assign_mesh_level(mesh_order[, name])</code>	Assign a mesh level to objects.
<code>assign_mesh_level_to_group(mesh_level, ...)</code>	Assign a mesh level to a group.
<code>assign_mesh_region([assignment, level, name])</code>	Assign a predefined surface mesh level to an object.
<code>assign_priorities(assignment)</code>	Set objects priorities.
<code>automatic_mesh_3D(accuracy[, able_stair_step])</code>	Create a generic custom mesh for a custom 3D object.
<code>generate_mesh([name])</code>	Generate the mesh for a given setup name.

Attributes

<code>boundingdimension</code>	Bounding dimension.
<code>meshregions_dict</code>	Get mesh regions in the design.
<code>omeshmodule</code>	Icepak Mesh Module.

2.9.3 pyaedt.modules.Mesh3DLayout.Mesh3d

```
class pyaedt.modules.Mesh3DLayout.Mesh3d(app)
```

Manages mesh operations for HFSS 3D Layout.

Provides the main AEDT mesh functionality. The inherited class `AEDTConfig` contains all `_desktop` hierarchical calls needed by this class.

Parameters

<code>app</code>	[pyaedt.application.Analysis3DLayout.FieldAnalysis3DLayout]
------------------	---

`__init__(app)`

Methods

<code>assign_length_mesh(setup, layer, net[, ...])</code>	Assign mesh length.
<code>assign_skin_depth(setup, layer, net[, ...])</code>	Assign skin depth to the mesh.
<code>delete_mesh_operations(setup, name)</code>	Remove mesh operations from a setup.
<code>generate_mesh(name)</code>	Generate the mesh for a design.

Attributes

<code>omeshmodule</code>	AEDT Mesh Module.
--------------------------	-------------------

```
from pyaedt import Maxwell3d
app = Maxwell3d(specified_version="2023.1",
                 non_graphical=False, new_desktop_session=True,
                 close_on_exit=True, student_version=False)
# This call returns the Mesh class
my_mesh = app.mesh
# This call executes a Mesh method and creates an object to control the mesh operation
mesh_operation_object = my_mesh.assign_surface_mesh("MyBox", 2)
...
```

2.10 Setup

This section lists setup modules:

- Setup for HFSS, Maxwell 2D, Maxwell 3D, Q2D Extractor, and Q3D Extractor
- Setup3DLayout for HFSS 3D Layout
- SetupCircuit for Circuit and Twin Builder

The Setup object is accessible through the `create_setup` method and `setups` object list.

<code>SetupHFSS</code>	Initializes, creates, and updates an HFSS setup.
<code>SetupHFSSAuto</code>	Initializes, creates, and updates an HFSS SBR+ or HFSS Auto setup.
<code>SetupSBR</code>	Initializes, creates, and updates an HFSS SBR+ or HFSS Auto setup.
<code>SetupQ3D</code>	Initializes, creates, and updates an Q3D setup.
<code>SetupMaxwell</code>	Initializes, creates, and updates an HFSS setup.
<code>Setup</code>	Initializes, creates, and updates a 3D setup.
<code>Setup3DLayout</code>	Initializes, creates, and updates a 3D Layout setup.
<code>SetupCircuit</code>	Manages a circuit setup.

2.10.1 pyaedt.modules.SolveSetup.SetupHFSS

```
class pyaedt.modules.SolveSetup.SetupHFSS(app, solution_type, name='MySetupAuto',
                                         is_new_setup=True)
```

Initializes, creates, and updates an HFSS setup.

Parameters

app	[pyaedt.application.Analysis.Analysis] Inherited app object.
solution_type	[int, str] Type of the setup.
name	[str, optional] Name of the setup. The default is "MySetupAuto".
is_new_setup	[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.

```
__init__(app, solution_type, name='MySetupAuto', is_new_setup=True)
```

Methods

add_derivatives(derivative_list)	Add derivatives to the setup.
add_mesh_link(design[, solution, ...])	Add a mesh link to another design.
add_sweep([name, sweep_type])	Add a sweep to the project.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Add a new setup based on class settings in AEDT.
create_frequency_sweep([unit, ...])	Create a sweep with the specified number of points.
create_linear_step_sweep([unit, ...])	Create a Sweep with a specified frequency step.
create_report([expressions, domain, ...])	Create a report in AEDT.
create_single_point_sweep([unit, freq, ...])	Create a Sweep with a single frequency point.
delete()	Delete actual Setup.
delete_sweep(name)	Delete a sweep.
disable([name])	Disable a setup.
enable([name])	Enable a setup.
enable_adaptive_setup_broadband(...[, ...])	Enable HFSS broadband setup.
enable_adaptive_setup_multifrequency(freque	Enable HFSS multi-frequency setup.
enable_adaptive_setup_single([freq, ...])	Enable HFSS single frequency setup.
enable_expression_cache(expressions[, ...])	Enable an expression cache.
get_derivative_variables()	Return Derivative Enabled variables.
get_solution_data([expressions, domain, ...])	Get a simulation result from a solved setup and cast it in a SolutionData object.
get_sweep([name])	Return frequency sweep object of a given sweep.
get_sweep_names()	Get the names of all sweeps in a given analysis setup.
set_tuning_offset(offsets)	Set derivative variable to a specific offset value.
start_continue_from_previous_setup(design, ...)	Start or continue from a previously solved setup.
update([properties])	Update the setup based on either the class argument or a dictionary.

Attributes

available_properties	Available properties.
default_intrinsics	Retrieve default intrinsic for actual setup.
is_solved	Verify if solutions are available for given setup.
name	Name.
omodule	Analysis module.
p_app	Parent.

2.10.2 pyaedt.modules.SolveSetup.SetupHFSSAuto

```
class pyaedt.modules.SolveSetup.SetupHFSSAuto(app, solution_type, name='MySetupAuto',  
                                              is_new_setup=True)
```

Initializes, creates, and updates an HFSS SBR+ or HFSS Auto setup.

Parameters

- app**
[pyaedt.application.Analysis.Analysis] Inherited app object.
 - solution_type**
[int, str] Type of the setup.
 - name**
[str, optional] Name of the setup. The default is "MySetupAuto".
 - is_new_setup**
[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.
- __init__(app, solution_type, name='MySetupAuto', is_new_setup=True)**

Methods

<code>add_derivatives(derivative_list)</code>	Add derivatives to the setup.
<code>add_mesh_link(design[, solution, ...])</code>	Add a mesh link to another design.
<code>add_subrange(range_type, start[, end, ...])</code>	Add a subrange to the sweep.
<code>analyze([cores, tasks, gpus, acf_file, ...])</code>	Solve the active design.
<code>create()</code>	Add a new setup based on class settings in AEDT.
<code>create_report([expressions, domain, ...])</code>	Create a report in AEDT.
<code>delete()</code>	Delete actual Setup.
<code>disable([name])</code>	Disable a setup.
<code>enable([name])</code>	Enable a setup.
<code>enable_adaptive_setup_broadband(...[, ...])</code>	Enable HFSS broadband setup.
<code>enable_adaptive_setup_multifrequency(freque</code>	Enable HFSS multi-frequency setup.
<code>enable_adaptive_setup_single([frequency,</code>	Enable HFSS single frequency setup.
<code>...])</code>	
<code>enable_expression_cache(expressions[, ...])</code>	Enable an expression cache.
<code>get_derivative_variables()</code>	Return Derivative Enabled variables.
<code>get_solution_data([expressions, domain, ...])</code>	Get a simulation result from a solved setup and cast it in a <code>SolutionData</code> object.
<code>set_tuning_offset(offsets)</code>	Set derivative variable to a specific offset value.
<code>start_continue_from_previous_setup(design,</code>	Start or continue from a previously solved setup.
<code>...)</code>	
<code>update([properties])</code>	Update the setup based on either the class argument or a dictionary.

Attributes

<code>available_properties</code>	Available properties.
<code>default_intrinsics</code>	Retrieve default intrinsic for actual setup.
<code>is_solved</code>	Verify if solutions are available for given setup.
<code>name</code>	Name.
<code>omodule</code>	Analysis module.
<code>p_app</code>	Parent.

2.10.3 pyaedt.modules.SolveSetup.SetupSBR

```
class pyaedt.modules.SolveSetup.SetupSBR(app, solution_type, name='MySetupAuto', is_new_setup=True)
```

Initializes, creates, and updates an HFSS SBR+ or HFSS Auto setup.

Parameters

- app**
[pyaedt.application.Analysis.Analysis] Inherited app object.
- solution_type**
[int, str] Type of the setup.
- name**
[str, optional] Name of the setup. The default is "MySetupAuto".

is_new_setup

[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.

__init__(app, solution_type, name='MySetupAuto', is_new_setup=True)

Methods

add_mesh_link(design[, solution, ...])	Add a mesh link to another design.
add_subrange(range_type, start[, end, ...])	Add a subrange to the sweep.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Add a new setup based on class settings in AEDT.
create_report([expressions, domain, ...])	Create a report in AEDT.
delete()	Delete actual Setup.
disable([name])	Disable a setup.
enable([name])	Enable a setup.
enable_expression_cache(expressions[, ...])	Enable an expression cache.
get_solution_data([expressions, domain, ...])	Get a simulation result from a solved setup and cast it in a <code>SolutionData</code> object.
start_continue_from_previous_setup(design, ...)	Start or continue from a previously solved setup.
update([properties])	Update the setup based on either the class argument or a dictionary.

Attributes

available_properties	Available properties.
default_intrinsics	Retrieve default intrinsic for actual setup.
is_solved	Verify if solutions are available for given setup.
name	Name.
omodule	Analysis module.
p_app	Parent.

2.10.4 pyaedt.modules.SolveSetup.SetupQ3D

class pyaedt.modules.SolveSetup.SetupQ3D(app, solution_type, name='MySetupAuto', is_new_setup=True)
Initializes, creates, and updates an Q3D setup.

Parameters**app**

[pyaedt.application.Analysis3D.FieldAnalysis3D] Inherited app object.

solution_type

[int, str] Type of the setup.

name

[str, optional] Name of the setup. The default is "MySetupAuto".

is_new_setup

[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.

__init__(app, solution_type, name='MySetupAuto', is_new_setup=True)

Methods

add_mesh_link(design[, solution, ...])	Add a mesh link to another design.
add_sweep([name, sweep_type])	Add a sweep to the project.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Add a new setup based on class settings in AEDT.
create_frequency_sweep([unit, ...])	Create a sweep with the specified number of points.
create_linear_step_sweep([unit, ...])	Create a sweep with a specified frequency step.
create_report([expressions, domain, ...])	Create a report in AEDT.
create_single_point_sweep([unit, freq, ...])	Create a sweep with a single frequency point.
delete()	Delete actual Setup.
disable([name])	Disable a setup.
enable([name])	Enable a setup.
enable_expression_cache(expressions[, ...])	Enable an expression cache.
get_solution_data([expressions, domain, ...])	Get a simulation result from a solved setup and cast it in a SolutionData object.
get_sweep([name])	Get the frequency sweep object of a given sweep.
start_continue_from_previous_setup(design, ...)	Start or continue from a previously solved setup.
update([properties])	Update the setup based on either the class argument or a dictionary.

Attributes

ac_rl_enabled	Get/Set the AC RL solution in active Q3D setup.
available_properties	Available properties.
capacitance_enabled	Get/Set the Capacitance solution in active Q3D setup.
dc_enabled	Get/Set the DC solution in active Q3D setup.
dc_resistance_only	Get/Set the DC Resistance Only or Resistance/Inductance calculation in active Q3D setup.
default_intrinsics	Retrieve default intrinsic for actual setup.
is_solved	Verify if solutions are available for given setup.
name	Name.
omodule	Analysis module.
p_app	Parent.

2.10.5 pyaedt.modules.SolveSetup.SetupMaxwell

```
class pyaedt.modules.SolveSetup.SetupMaxwell(app, solution_type, name='MySetupAuto',  
                                             is_new_setup=True)
```

Initializes, creates, and updates an HFSS setup.

Parameters

app
[pyaedt.application.Analysis.Analysis] Inherited app object.

solution_type
[int, str] Type of the setup.

name
[str, optional] Name of the setup. The default is "MySetupAuto".

is_new_setup
[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.

```
__init__(app, solution_type, name='MySetupAuto', is_new_setup=True)
```

Methods

add_eddy_current_sweep([range_type, start, ...])	Create a Maxwell Eddy Current Sweep.
add_mesh_link(design[, solution, ...])	Add a mesh link to another design.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Add a new setup based on class settings in AEDT.
create_report([expressions, domain, ...])	Create a report in AEDT.
delete()	Delete actual Setup.
disable([name])	Disable a setup.
enable([name])	Enable a setup.
enable_control_program(control_program_path)	Enable control program option is solution setup. Provide externally created executable files, or Python (*.py) scripts that are called after each time step, and allow you to control the source input, circuit elements, mechanical quantities, time step, and stopping criteria, based on the updated solutions.
enable_expression_cache(expressions[, ...])	Enable an expression cache.
get_solution_data([expressions, domain, ...])	Get a simulation result from a solved setup and cast it in a SolutionData object.
start_continue_from_previous_setup(design, ...)	Start or continue from a previously solved setup.
update([properties])	Update the setup based on either the class argument or a dictionary.

Attributes

<code>available_properties</code>	Available properties.
<code>default_intrinsics</code>	Retrieve default intrinsic for actual setup.
<code>is_solved</code>	Verify if solutions are available for given setup.
<code>name</code>	Name.
<code>omodule</code>	Analysis module.
<code>p_app</code>	Parent.

2.10.6 pyaedt.modules.SolveSetup.Setup

`class pyaedt.modules.SolveSetup.Setup(app, solution_type, name='MySetupAuto', is_new_setup=True)`
Initializes, creates, and updates a 3D setup.

Parameters

<code>app</code>	[pyaedt.application.Analysis.Analysis] Inherited app object.
<code>solution_type</code>	[int, str] Type of the setup.
<code>name</code>	[str, optional] Name of the setup. The default is "MySetupAuto".
<code>is_new_setup</code>	[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.
<code>__init__(app, solution_type, name='MySetupAuto', is_new_setup=True)</code>	

Methods

<code>add_mesh_link(design[, solution, ...])</code>	Add a mesh link to another design.
<code>analyze([cores, tasks, gpus, acf_file, ...])</code>	Solve the active design.
<code>create()</code>	Add a new setup based on class settings in AEDT.
<code>create_report([expressions, domain, ...])</code>	Create a report in AEDT.
<code>delete()</code>	Delete actual Setup.
<code>disable([name])</code>	Disable a setup.
<code>enable([name])</code>	Enable a setup.
<code>enable_expression_cache(expressions[, ...])</code>	Enable an expression cache.
<code>get_solution_data([expressions, domain, ...])</code>	Get a simulation result from a solved setup and cast it in a <code>SolutionData</code> object.
<code>start_continue_from_previous_setup(design, ...)</code>	Start or continue from a previously solved setup.
<code>update([properties])</code>	Update the setup based on either the class argument or a dictionary.

Attributes

available_properties	Available properties.
default_intrinsics	Retrieve default intrinsic for actual setup.
is_solved	Verify if solutions are available for given setup.
name	Name.
omodule	Analysis module.
p_app	Parent.

2.10.7 pyaedt.modules.SolveSetup.Setup3DLayout

```
class pyaedt.modules.SolveSetup.Setup3DLayout(app, solution_type, name='MySetupAuto',  
                                              is_new_setup=True)
```

Initializes, creates, and updates a 3D Layout setup.

Parameters

app	[pyaedt.application.Analysis3DLayout.FieldAnalysis3DLayout] Inherited app object.
solution_type	[int or str] Type of the setup.
name	[str, optional] Name of the setup. The default is "MySetupAuto".
is_new_setup	[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.
__init__ (app, solution_type, name='MySetupAuto', is_new_setup=True)	

Methods

add_sweep([name, sweep_type])	Add a frequency sweep.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Add a new setup based on class settings in AEDT.
create_report([expressions, domain, ...])	Create a report in AEDT.
disable()	Disable a setup.
enable()	Enable a setup.
export_to_hfss(output_file[, keep_net_name])	Export the HFSS 3D Layout design to an HFSS 3D design.
export_to_json(file_path[, overwrite])	Export all setup properties into a json file.
export_to_q3d(output_file[, keep_net_name])	Export the HFSS 3D Layout design to a Q3D design.
get_solution_data([expressions, domain, ...])	Get a simulation result from a solved setup and cast it in a SolutionData object.
get_sweep([name])	Return frequency sweep object of a given sweep.
import_from_json(file_path)	Import setup properties from a json file.
update([properties])	Update the setup based on the class arguments or a dictionary.

Attributes

<code>available_properties</code>	Available properties.
<code>default_intrinsics</code>	Retrieve default intrinsic for actual setup.
<code>is_solved</code>	Verify if solutions are available for a given setup.
<code>name</code>	Name.
<code>omodule</code>	Analysis module.
<code>p_app</code>	Parent.
<code>solver_type</code>	Setup type.

2.10.8 pyaedt.modules.SolveSetup.SetupCircuit

```
class pyaedt.modules.SolveSetup.SetupCircuit(app, solution_type, name='MySetupAuto',
                                             is_new_setup=True)
```

Manages a circuit setup.

Parameters

<code>app</code>	[pyaedt.application.AnalysisNexxim.FieldAnalysisCircuit] Inherited app object.
<code>solution_type</code>	[str, int] Type of the setup.
<code>name</code>	[str, optional] Name of the setup. The default is "MySetupAuto".
<code>is_new_setup</code>	[bool, optional] Whether to create the setup from a template. The default is True. If False, access is to the existing setup.

```
__init__(app, solution_type, name='MySetupAuto', is_new_setup=True)
```

Methods

<code>add_sweep_count([sweep_variable, start, ...])</code>	Add a step sweep to existing Circuit Setup.
<code>add_sweep_points([sweep_variable, ...])</code>	Add a linear count sweep to existing Circuit Setup.
<code>add_sweep_step([sweep_variable, start, ...])</code>	Add a linear count sweep to existing Circuit Setup.
<code>analyze([cores, tasks, gpus, acf_file, ...])</code>	Solve the active design.
<code>create()</code>	Add a new setup based on class settings in AEDT.
<code>create_report([expressions, domain, ...])</code>	Create a report in AEDT.
<code>disable([name])</code>	Disable a setup.
<code>enable([name])</code>	Enable a setup.
<code>enable_expression_cache(expressions[, ...])</code>	Enable a setup expression cache.
<code>get_solution_data([expressions, domain, ...])</code>	Get a simulation result from a solved setup and cast it in a SolutionData object.
<code>update([properties])</code>	Update the setup based on the class arguments or a dictionary.

Attributes

available_properties	Available properties.
default_intrinsics	Retrieve default intrinsic for actual setup.
is_solved	Verify if solutions are available for given setup.
name	Name.
omodule	Analysis module.
p_app	Parent.

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call returns the Setup class
my_setup = app.setups[0]

# This call returns a Setup object
setup = app.create_setup("MySetup")

... 
```

2.11 Sweep classes

This section lists sweep classes and their default values:

- SweepHFSS for HFSS
- SweepHFSS3DLayout for HFSS 3D Layout
- SweepMatrix for Q3D and 2D Extractor

The Setup object is accessible through the methods available for sweep creation.

SweepHFSS	Initializes, creates, and updates sweeps in HFSS.
SweepHFSS3DLayout	Initializes, creates, and updates sweeps in HFSS 3D Layout.
SweepMatrix	Initializes, creates, and updates sweeps in Q3D.

2.11.1 pyaedt.modules.SolveSweeps.SweepHFSS

```
class pyaedt.modules.SolveSweeps.SweepHFSS(setup, name, sweep_type='Interpolating', props=None)
```

Initializes, creates, and updates sweeps in HFSS.

Parameters

setup

[:class pyaedt.modules.SolveSetup.Setup] Setup to use for the analysis.

name

[`str`] Name of the sweep.

sweep_type

[`str`, optional] Type of the sweep. Options are "Fast", "Interpolating", and "Discrete". The default is "Interpolating".

props

[`dict`, optional] Dictionary of the properties. The default is `None`, in which case the default properties are retrieved.

Examples

```
>>> hfss = Hfss(specified_version=version, projectname=proj, designname=gtemDesign, u
   ↪solution_type=solutiontype,
   ↪setup_name=setup_name, new_desktop_session=False, close_on_
   ↪exit=False)
>>> hfss_setup = hfss.setups[0]
>>> hfss_sweep = SweepHFSS(hfss_setup, 'Sweep', sweep_type ='Interpolating', u
   ↪props=None)
```

`__init__(setup, name, sweep_type='Interpolating', props=None)`

Methods

<code>add_subrange(range_type, start[, end, ...])</code>	Add a range to the sweep.
<code>create()</code>	Create a sweep.
<code>update()</code>	Update a sweep.

Attributes

<code>basis_frequencies</code>	List of all frequencies that have fields available.
<code>frequencies</code>	List of all frequencies of the active sweep.
<code>is_solved</code>	Verify if solutions are available for the sweep.

2.11.2 pyaedt.modules.SolveSweeps.SweepHFSS3DLayout

```
class pyaedt.modules.SolveSweeps.SweepHFSS3DLayout(setup, name, sweep_type='Interpolating',
   save_fields=True, props=None, **kwargs)
```

Initializes, creates, and updates sweeps in HFSS 3D Layout.

Parameters**setup**

[:class pyaedt.modules.SolveSetup.Setup] Setup to use for the analysis.

name

[`str`] Name of the sweep.

sweep_type
[`str`, optional] Type of the sweep. Options are "Interpolating" and "Discrete". The default is "Interpolating".

save_fields
[`bool`, optional] Whether to save the fields. The default is True.

props
[`dict`, optional] Dictionary of the properties. The default is None, in which case the default properties are retrieved.

__init__(setup, name, sweep_type='Interpolating', save_fields=True, props=None, **kwargs)

Methods

<code>add_subrange(range_type, start[, end, ...])</code>	Add a subrange to the sweep.
<code>change_range(range_type, start[, end, ...])</code>	Change the range of the sweep.
<code>change_type(sweep_type)</code>	Change the type of the sweep.
<code>create()</code>	Create a sweep.
<code>set_save_fields(save_fields[, save_rad_fields])</code>	Choose whether to save fields.
<code>update()</code>	Update the sweep.

Attributes

<code>combined_name</code>	Compute the setup_name : sweep_name string.
<code>is_solved</code>	Verify if solutions are available for the sweep.

2.11.3 pyaedt.modules.SolveSweeps.SweepMatrix

class pyaedt.modules.SolveSweeps.SweepMatrix(setup, name, sweep_type='Interpolating', props=None)
Initializes, creates, and updates sweeps in Q3D.

Parameters

setup
[:class pyaedt.modules.SolveSetup.Setup] Setup used for the analysis.

name
[`str`] Name of the sweep.

sweep_type
[`str`, optional] Type of the sweep. Options are "Fast", "Interpolating", and "Discrete". The default is "Interpolating".

props
[`dict`] Dictionary of the properties. The default is None, in which case the default properties are retrieved.

__init__(setup, name, sweep_type='Interpolating', props=None)

Methods

<code>add_subrange(range_type, start[, end, ...])</code>	Add a subrange to the sweep.
<code>create()</code>	Create a sweep.
<code>update()</code>	Update the sweep.

Attributes

<code>basis_frequencies</code>	Get the list of all frequencies that have fields available.
<code>frequencies</code>	List of all frequencies of the active sweep.
<code>is_solved</code>	Verify if solutions are available for given sweep.

2.12 Postprocessing

This section lists modules for creating and editing plots in AEDT. They are accessible through the `post` property.

Note: Some capabilities of the `AdvancedPostProcessing` module require Python 3 and installations of the `numpy`, `matplotlib`, and `pyvista` packages.

Note: Some functionalities are available only when AEDT is running in graphical mode.

<code>AdvancedPostProcessing.PostProcessor</code>	Contains advanced postprocessing functionalities that require Python 3.x packages like NumPy and Matplotlib.
<code>solutions.SolutionData</code>	Contains information from the <code>GetSolutionDataPerVariation()</code> method.
<code>solutions.FieldPlot</code>	Provides for creating and editing field plots.
<code>solutions.FfdSolutionData</code>	Provides antenna array far-field data.

2.12.1 pyaedt.modules.AdvancedPostProcessing.PostProcessor

```
class pyaedt.modules.AdvancedPostProcessing.PostProcessor(app)
```

Contains advanced postprocessing functionalities that require Python 3.x packages like NumPy and Matplotlib.

Parameters

`app`

Inherited parent object.

Examples

Basic usage demonstrated with an HFSS, Maxwell, or any other design:

```
>>> from pyaedt import Hfss
>>> aedtapp = Hfss()
>>> post = aedtapp.post

__init__(app)
```

Methods

animate_fields_from_aedtplt(plot_name[, ...])	Generate a field plot to an image file (JPG or PNG) using PyVista.
available_display_types([report_category])	Retrieve display types for a report categories.
available_quantities_categories([...])	Compute the list of all available report categories.
available_report_quantities([...])	Compute the list of all available report quantities of a given report quantity category.
available_report_solutions([report_category])	Get the list of available solutions that can be used for the reports.
change_field_plot_scale(plot_name, ...[, ...])	Change Field Plot Scale.
change_field_property(plot_name, ...)	Modify a field plot property.
copy_report_data(plot_name)	Copy report data as static data.
create_3d_plot(solution_data[, ...])	Create a 3D plot using Matplotlib.
create_creeping_plane_visual_ray_tracing([...])	Create a Creeping Wave Plane Wave Visual Ray Tracing and return the class object.
create_creeping_point_visual_ray_tracing([...])	Create a Creeping Wave Point Source Visual Ray Tracing and return the class object.
create_fieldplot_cutplane(assignment, quantity)	Create a field plot of cut planes.
create_fieldplot_layers(layers, quantity[, ...])	Create a field plot of stacked layer plot.
create_fieldplot_layers_nets(layers_nets, ...)	Create a field plot of stacked layer plot.
create_fieldplot_line(assignment, quantity)	Create a field plot of the line.
create_fieldplot_line_traces(seeding_faces)	Create a field plot of the line.
create_fieldplot_surface(assignment, quantity)	Create a field plot of surfaces.
create_fieldplot_volume(assignment, quantity)	Create a field plot of volumes.
create_report([expressions, ...])	Create a report in AEDT.
create_report_from_configuration([...])	Create a report based on a JSON file, TOML file, or dictionary of properties.
create_sbr_plane_visual_ray_tracing([...])	Create an SBR Plane Wave Visual Ray Tracing and return the class object.
create_sbr_point_visual_ray_tracing([...])	Create an SBR Point Source Visual Ray Tracing and return the class object.
delete_field_plot(name)	Delete a field plot.
delete_report([plot_name])	Delete all reports or specific report.
export_field_file(quantity[, solution, ...])	Use the field calculator to create a field file based on a solution and variation.
export_field_file_on_grid(quantity[, ...])	Use the field calculator to create a field file on a grid based on a solution and variation.

continues on next page

Table 39 – continued from previous page

<code>export_field_jpg(file_name, plot_name, ...)</code>	Export a field plot and coordinate system to a JPG file.
<code>export_field_plot(plot_name, output_dir[, ...])</code>	Export a field plot.
<code>export_mesh_obj([setup, intrinsics, ...])</code>	Export the mesh in AEDTPLT format.
<code>export_model_obj([assignment, export_path, ...])</code>	Export the model.
<code>export_model_picture([full_name, show_axis, ...])</code>	Export a snapshot of the model to a JPG file.
<code>export_report_to_csv(project_dir, plot_name)</code>	Export the 2D Plot data to a CSV file.
<code>export_report_to_file(output_dir, plot_name, ...)</code>	Export a 2D Plot data to a file.
<code>export_report_to_jpg(project_path, plot_name)</code>	Export the SParameter plot to a JPG file.
<code>get_efields_data([setup_sweep_name, ...])</code>	Compute Etheta and EPhi.
<code>get_far_field_data([expressions, ...])</code>	Generate far field data using the GetSolutionDataPerVariation() method.
<code>get_model_plotter_geometries([objects, ...])</code>	Initialize the Model Plotter object with actual modeler objects and return it.
<code>get_scalar_field_value(quantity[, ...])</code>	Use the field calculator to Compute Scalar of a Field.
<code>get_solution_data([expressions, ...])</code>	Get a simulation result from a solved setup and cast it in a SolutionData object.
<code>get_solution_data_per_variation(...)</code>	Retrieve solution data for each variation.
<code>nb_display([show_axis, show_grid, show_ruler])</code>	Show the Jupyter Notebook display.
<code>plot_animated_field(quantity, assignment[, ...])</code>	Create an animated field plot using Python PyVista and export to a gif file.
<code>plot_field(quantity, assignment[, ...])</code>	Create a field plot using Python PyVista and export to an image file (JPG or PNG).
<code>plot_field_from_fieldplot(plot_name[, ...])</code>	Export a field plot to an image file (JPG or PNG) using Python PyVista.
<code>plot_model_obj([objects, show, export_path, ...])</code>	Plot the model or a subset of objects.
<code>plot_scene(frames, gif_path[, norm_index, ...])</code>	Plot the current model 3D scene with overlapping animation coming from a file list and save the gif.
<code>power_budget([units, temperature, output_type])</code>	Power budget calculation.
<code>rename_report(plot_name, new_name)</code>	Rename a plot.
<code>set_tuning_offset(setup, offsets)</code>	Set derivative variable to a specific offset value.
<code>steal_focus_oneditor()</code>	Remove the selection of an object that would prevent the image from exporting correctly.
<code>volumetric_loss(assignment)</code>	Use the field calculator to create a variable for volumetric losses.

Attributes

all_report_names	List of all report names.
available_report_types	Report types.
logger	Logger.
model_units	Model units.
modeler	Modeler.
ofieldsreporter	Fields reporter.
oreportsetup	Report setup.
post_osolution	Solution.
post_solution_type	Design solution type.
update_report_dynamically	Get/Set the boolean to automatically update reports on edits.

2.12.2 pyaedt.modules.solutions.SolutionData

```
class pyaedt.modules.solutions.SolutionData(aedtdata)
```

Contains information from the GetSolutionDataPerVariation() method.

```
__init__(aedtdata)
```

Methods

data_db10([expression, convert_to_SI])	Retrieve the data in the database for an expression and convert in db10.
data_db20([expression, convert_to_SI])	Retrieve the data in the database for an expression and convert in db20.
data_imag([expression, convert_to_SI])	Retrieve the imaginary part of the data for an expression.
data_magnitude([expression, convert_to_SI])	Retrieve the data magnitude of an expression.
data_phase([expression, radians])	Retrieve the phase part of the data for an expression.
data_real([expression, convert_to_SI])	Retrieve the real part of the data for an expression.
export_data_to_csv(output[, delimiter])	Save to output csv file the Solution Data.
ifft([curve_header, u_axis, v_axis, window])	Create IFFT of given complex data.
ifft_to_file([u_axis, v_axis, ...])	Save IFFT matrix to a list of CSV files (one per time step).
init_solutions_data()	Initialize the database and store info in variables.
is_real_only([expression])	Check if the expression has only real values or not.
plot([curves, formula, size, show_legend, ...])	Create a matplotlib figure based on a list of data.
plot_3d([curve, x_axis, y_axis, x_label, ...])	Create a matplotlib 3D figure based on a list of data.
set_active_variation([var_id])	Set the active variations to one of available variations in self.variations.
to_degrees(input_list)	Convert an input list from radians to degrees.
to_radians(input_list)	Convert an input list from degrees to radians.
update_sweeps()	Update sweeps.
variation_values(variation)	Get the list of the specific variation available values.

Attributes

<code>enable_pandas_output</code>	Set/Get a flag to use Pandas to export dict and lists.
<code>expressions</code>	Expressions.
<code>full_matrix_mag_phase</code>	Get the full available solution data magnitude and phase in radians.
<code>full_matrix_real_imag</code>	Get the full available solution data in Real and Imaginary parts.
<code>intrinsics</code>	Get intrinsics dictionary on active variation.
<code>nominal_variation</code>	Nominal variation.
<code>primary_sweep</code>	Primary sweep.
<code>primary_sweep_values</code>	Retrieve the primary sweep for a given data and primary variable.
<code>primary_sweep_variations</code>	Retrieve the variations lists for a given primary variable.

2.12.3 pyaedt.modules.solutions.FieldPlot

```
class pyaedt.modules.solutions.FieldPlot(postprocessor, objects=[], surfaces=[], lines=[],
                                         cutplanes=[], solution='', quantity='', intrinsics={},
                                         seeding_faces=[], layer_nets=[], layer_plot_type='LayerNetsExtFace')
```

Provides for creating and editing field plots.

Parameters

- postprocessor**
[pyaedt.modules.PostProcessor.PostProcessor]
- objects**
[list] List of objects.
- solution**
[str] Name of the solution.
- quantity**
[str] Name of the plot or the name of the object.
- intrinsics**
[dict, optional] Name of the intrinsic dictionary. The default is {}.

```
__init__(postprocessor, objects=[], surfaces=[], lines=[], cutplanes=[], solution='', quantity='',
        intrinsics={}, seeding_faces=[], layer_nets=[], layer_plot_type='LayerNetsExtFace')
```

Methods

change_plot_scale(minimum_value, maximum_value)	Change Field Plot Scale.
create()	Create a field plot.
delete()	Delete the field plot.
export_image([full_path, width, height, ...])	Export the active plot to an image file.
export_image_from_aedtplt([export_path, ...])	Save an image of the active plot using PyVista.
update()	Update the field plot.
update_field_plot_settings()	Modify the field plot settings.

Attributes

field_line_trace_plot_settings	Settings for the field line traces in the plot.
field_plot_settings	Field Plot Settings.
filter_boxes	Volumes on which filter the plot.
intrinsicVar	Intrinsic variable.
plotGeomInfo	Plot geometry information.
plotsettings	Plot settings.
surfacePlotInstruction	Surface plot settings.
surfacePlotInstructionLineTraces	Surface plot settings for field line traces.

2.12.4 pyaedt.modules.solutions.FfdSolutionData

```
class pyaedt.modules.solutions.FfdSolutionData(eep_files, frequencies)
```

Provides antenna array far-field data.

Read embedded element patterns generated in HFSS and return the Python interface to plot and analyze the array far-field data.

Parameters

eep_files

[list or str] List of embedded element pattern files for each frequency. If data is only provided for a single frequency, then a string can be passed instead of a one-element list.

frequencies

[list, str, int, or float] List of frequencies. If data is only available for a single frequency, then a float or integer may be passed instead of a one-element list.

Examples

```
>>> import pyaedt
>>> from pyaedt.modules.solutions import FfdSolutionData
>>> app = pyaedt.Hfss(specified_version="2023.2", designname="Antenna")
>>> setup_name = "Setup1 : LastAdaptive"
>>> frequencies = [77e9]
>>> sphere = "3D"
>>> data = app.get_antenna_ffd_solution_data(frequencies, setup_name, sphere)
```

(continues on next page)

(continued from previous page)

```
>>> eep_files = data.eep_files
>>> frequencies = data.frequencies
>>> app.release_desktop()
>>> farfield_data = FfdSolutionData(frequencies=frequencies, eep_files=eep_files)
>>> farfield_data.polar_plot_3d_pyvista(quantity_format="dB10",qty_str="rETotal")
```

`__init__(eep_files, frequencies)`

Methods

<code>combine_farfield([phi_scan, theta_scan])</code>	Compute the far field pattern calculated for a specific phi and theta scan angle requested.
<code>get_far_field_mesh([quantity, quantity_format])</code>	Generate a PyVista UnstructuredGrid object that represents the far field mesh.
<code>plot_2d_cut([quantity, primary_sweep, ...])</code>	Create a 2D plot of a specified quantity in Matplotlib.
<code>plot_farfield_contour([quantity, phi, ...])</code>	Create a contour plot of a specified quantity.
<code>polar_plot_3d([quantity, phi, theta, title, ...])</code>	Create a 3D plot of a specified quantity.
<code>polar_plot_3d_pyvista([quantity, ...])</code>	Create a 3D polar plot of the geometry with a radiation pattern in PyVista.

Attributes

<code>frequency</code>	Active frequency.
<code>frequency_value</code>	Frequency value in Hz.
<code>mag_offset</code>	List of additional magnitudes on each port.
<code>origin</code>	Far field origin in meters.
<code>phase_offset</code>	List of additional phase offsets in degrees on each port.
<code>taper</code>	Taper type.

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call returns the PostProcessor class
post = app.post

# This call returns a FieldPlot object
plotf = post.create_fieldplot_volume(objects, quantity_name, setup_name, intrinsics)

# This call returns a SolutionData object
my_data = post.get_solution_data(expressions=trace_names)

# This call returns a new standard report object and creates one or multiple reports.
# from it.
standard_report = post.reports_by_category.standard("db(S(1,1))")
```

(continues on next page)

(continued from previous page)

```
standard_report.create()  
sols = standard_report.get_solution_data()  
...
```

2.12.5 AEDT report management

AEDT provides great flexibility in reports. PyAEDT has classes for manipulating any report property.

Note: Some functionalities are available only when AEDT is running in graphical mode.

<code>report_templates.Trace</code>	Provides trace management.
<code>report_templates.LimitLine</code>	Line Limit Management Class.
<code>report_templates.Standard</code>	Provides a reporting class that fits most of the app's standard reports.
<code>report_templates.Fields</code>	Provides for managing fields.
<code>report_templates.NearField</code>	Provides for managing near field reports.
<code>report_templates.FarField</code>	Provides for managing far field reports.
<code>report_templates.EyeDiagram</code>	Provides for managing eye diagram reports.
<code>report_templates.Emission</code>	Provides for managing emission reports.
<code>report_templates.Spectral</code>	Provides for managing spectral reports from transient data.

`pyaedt.modules.report_templates.Trace`

```
class pyaedt.modules.report_templates.Trace(report_setup, aedt_name)  
    Provides trace management.  
    __init__(report_setup, aedt_name)
```

Methods

<code>set_symbol_properties([show, style, ...])</code>	Set symbol properties.
<code>set_trace_properties([trace_style, width, ...])</code>	Set trace properties.

Attributes

<code>name</code>	Trace name.
-------------------	-------------

pyaedt.modules.report_templates.LimitLine

```
class pyaedt.modules.report_templates.LimitLine(report_setup, trace_name)
    Line Limit Management Class.

    __init__(report_setup, trace_name)
```

Methods

<code>set_line_properties([style, width, ...])</code>	Set trace properties.
---	-----------------------

pyaedt.modules.report_templates.Standard

```
class pyaedt.modules.report_templates.Standard(app, report_category, setup_name, expressions=None)
    Provides a reporting class that fits most of the apps standard reports.

    __init__(app, report_category, setup_name, expressions=None)
```

Methods

<code>add_cartesian_x_marker(<i>val</i>[, <i>name</i>])</code>	Add a cartesian X marker.
<code>add_cartesian_y_marker(<i>val</i>[, <i>name</i>, <i>y_axis</i>])</code>	Add a cartesian Y marker.
<code>add_limit_line_from_equation(<i>start_x</i>, ...[, <i>...]</i>)</code>	Add a Cartesian limit line from point lists.
<code>add_limit_line_from_points(<i>x_list</i>, <i>y_list</i>[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_note(<i>text</i>[, <i>x_position</i>, <i>y_position</i>])</code>	Add a note at a position.
<code>add_trace_to_report(<i>traces</i>[, <i>setup_name</i>, ...])</code>	Add a trace to a specific report.
<code>create([<i>plot_name</i>])</code>	Create a report.
<code>delete()</code>	Delete current report.
<code>delete_traces(<i>plot_name</i>, <i>traces_list</i>)</code>	Delete an existing trace or traces.
<code>edit_general_settings([<i>background_color</i>, ...])</code>	Edit general settings for the plot.
<code>edit_grid([<i>minor_x</i>, <i>minor_y</i>, <i>major_x</i>, ...])</code>	Edit the grid settings for the plot.
<code>edit_header([<i>company_name</i>, ...])</code>	Edit the plot header.
<code>edit_legend([<i>show_solution_name</i>, ...])</code>	Edit the plot legend.
<code>edit_x_axis([<i>font</i>, <i>font_size</i>, <i>italic</i>, <i>bold</i>, ...])</code>	Edit the X-axis settings.
<code>edit_x_axis_scaling([<i>linear_scaling</i>, ...])</code>	Edit the X-axis scaling settings.
<code>edit_y_axis([<i>axis_name</i>, <i>font</i>, <i>font_size</i>, ...])</code>	Edit the Y-axis settings.
<code>edit_y_axis_scaling([<i>axis_name</i>, ...])</code>	Edit the Y-axis scaling settings.
<code>get_solution_data()</code>	Get the report solution data.
<code>hide_legend([<i>solution_name</i>, <i>trace_name</i>, ...])</code>	Hide the Legend.
<code>import_traces(<i>file_path</i>, <i>plot_name</i>)</code>	Import report data from a file into a specified report.
<code>update_expressions_with_defaults([...])</code>	Update the list of expressions by taking all quantities from a given category.
<code>update_trace_in_report(<i>traces</i>[, <i>setup_name</i>, ...])</code>	Update a trace in a specific report.

Attributes

differential_pairs	Differential pairs flag.
domain	Plot domain.
expressions	Expressions.
limit_lines	List of available limit lines in the report.
matrix	2D or Q3D matrix name.
notes	List of available notes in the report.
plot_name	Plot name.
polyline	Polyline name for the field report.
primary_sweep	Primary sweep report.
primary_sweep_range	Primary sweep range report.
pulse_rise_time	Value of Pulse rise time for TDR plot.
report_category	Report category.
report_type	Report type.
secondary_sweep	Secondary sweep report.
secondary_sweep_range	Secondary sweep range report.
sub_design_id	Subdesign ID for a Circuit or HFSS 3D Layout sub-design.
time_start	Time start value.
time_stop	Time stop value.
time_windowing	Returns the TDR time windowing. Options are:
traces	List of available traces in the report.
use_pulse_in_tdr	Defines if the TDR should use a pulse or step.
variations	Variations.

`pyaedt.modules.report_templates.Fields`

```
class pyaedt.modules.report_templates.Fields(app, report_category, setup_name, expressions=None)
```

Provides for managing fields.

```
__init__(app, report_category, setup_name, expressions=None)
```

Methods

<code>add_cartesian_x_marker(val[, name])</code>	Add a cartesian X marker.
<code>add_cartesian_y_marker(val[, name, y_axis])</code>	Add a cartesian Y marker.
<code>add_limit_line_from_equation(start_x, ...[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_limit_line_from_points(x_list, y_list[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_note(text[, x_position, y_position])</code>	Add a note at a position.
<code>add_trace_to_report(traces[, setup_name, ...])</code>	Add a trace to a specific report.
<code>create([plot_name])</code>	Create a report.
<code>delete()</code>	Delete current report.
<code>delete_traces(plot_name, traces_list)</code>	Delete an existing trace or traces.
<code>edit_general_settings([background_color, ...])</code>	Edit general settings for the plot.
<code>edit_grid([minor_x, minor_y, major_x, ...])</code>	Edit the grid settings for the plot.
<code>edit_header([company_name, ...])</code>	Edit the plot header.
<code>edit_legend([show_solution_name, ...])</code>	Edit the plot legend.
<code>edit_x_axis([font, font_size, italic, bold, ...])</code>	Edit the X-axis settings.
<code>edit_x_axis_scaling([linear_scaling, ...])</code>	Edit the X-axis scaling settings.
<code>edit_y_axis([axis_name, font, font_size, ...])</code>	Edit the Y-axis settings.
<code>edit_y_axis_scaling([axis_name, ...])</code>	Edit the Y-axis scaling settings.
<code>get_solution_data()</code>	Get the report solution data.
<code>hide_legend([solution_name, trace_name, ...])</code>	Hide the Legend.
<code>import_traces(file_path, plot_name)</code>	Import report data from a file into a specified report.
<code>update_expressions_with_defaults([...])</code>	Update the list of expressions by taking all quantities from a given category.
<code>update_trace_in_report(traces[, setup_name, ...])</code>	Update a trace in a specific report.

Attributes

<code>differential_pairs</code>	Differential pairs flag.
<code>domain</code>	Plot domain.
<code>expressions</code>	Expressions.
<code>limit_lines</code>	List of available limit lines in the report.
<code>matrix</code>	2D or Q3D matrix name.
<code>notes</code>	List of available notes in the report.
<code>plot_name</code>	Plot name.
<code>point_number</code>	Polygon point number.
<code>polyline</code>	Polyline name for the field report.
<code>primary_sweep</code>	Primary sweep report.
<code>primary_sweep_range</code>	Primary sweep range report.
<code>report_category</code>	Report category.
<code>report_type</code>	Report type.
<code>secondary_sweep</code>	Secondary sweep report.
<code>secondary_sweep_range</code>	Secondary sweep range report.
<code>traces</code>	List of available traces in the report.
<code>use_pulse_in_tdr</code>	Defines if the TDR should use a pulse or step.
<code>variations</code>	Variations.

pyaedt.modules.report_templates.NearField

```
class pyaedt.modules.report_templates.NearField(app, report_category, setup_name,  
                                              expressions=None)
```

Provides for managing near field reports.

```
__init__(app, report_category, setup_name, expressions=None)
```

Methods

add_cartesian_x_marker(val[, name])	Add a cartesian X marker.
add_cartesian_y_marker(val[, name, y_axis])	Add a cartesian Y marker.
add_limit_line_from_equation(start_x, ...[, ...])	Add a Cartesian limit line from point lists.
add_limit_line_from_points(x_list, y_list[, ...])	Add a Cartesian limit line from point lists.
add_note(text[, x_position, y_position])	Add a note at a position.
add_trace_to_report(traces[, setup_name, ...])	Add a trace to a specific report.
create([plot_name])	Create a report.
delete()	Delete current report.
delete_traces(plot_name, traces_list)	Delete an existing trace or traces.
edit_general_settings([background_color, ...])	Edit general settings for the plot.
edit_grid([minor_x, minor_y, major_x, ...])	Edit the grid settings for the plot.
edit_header([company_name, ...])	Edit the plot header.
edit_legend([show_solution_name, ...])	Edit the plot legend.
edit_x_axis([font, font_size, italic, bold, ...])	Edit the X-axis settings.
edit_x_axis_scaling([linear_scaling, ...])	Edit the X-axis scaling settings.
edit_y_axis([axis_name, font, font_size, ...])	Edit the Y-axis settings.
edit_y_axis_scaling([axis_name, ...])	Edit the Y-axis scaling settings.
get_solution_data()	Get the report solution data.
hide_legend([solution_name, trace_name, ...])	Hide the Legend.
import_traces(file_path, plot_name)	Import report data from a file into a specified report.
update_expressions_with_defaults([...])	Update the list of expressions by taking all quantities from a given category.
update_trace_in_report(traces[, setup_name, ...])	Update a trace in a specific report.

Attributes

differential_pairs	Differential pairs flag.
domain	Plot domain.
expressions	Expressions.
limit_lines	List of available limit lines in the report.
matrix	2D or Q3D matrix name.
near_field	Near field name.
notes	List of available notes in the report.
plot_name	Plot name.
polyline	Polyline name for the field report.
primary_sweep	Primary sweep report.
primary_sweep_range	Primary sweep range report.
report_category	Report category.
report_type	Report type.
secondary_sweep	Secondary sweep report.
secondary_sweep_range	Secondary sweep range report.
traces	List of available traces in the report.
use_pulse_in_tdr	Defines if the TDR should use a pulse or step.
variations	Variations.

pyaedt.modules.report_templates.FarField

```
class pyaedt.modules.report_templates.FarField(app, report_category, setup_name, expressions=None)
```

Provides for managing far field reports.

```
__init__(app, report_category, setup_name, expressions=None)
```

Methods

add_cartesian_x_marker(val[, name])	Add a cartesian X marker.
add_cartesian_y_marker(val[, name, y_axis])	Add a cartesian Y marker.
add_limit_line_from_equation(start_x, ...[, ...])	Add a Cartesian limit line from point lists.
add_limit_line_from_points(x_list, y_list[, ...])	Add a Cartesian limit line from point lists.
add_note(text[, x_position, y_position])	Add a note at a position.
add_trace_to_report(traces[, setup_name, ...])	Add a trace to a specific report.
create([plot_name])	Create a report.
delete()	Delete current report.
delete_traces(plot_name, traces_list)	Delete an existing trace or traces.
edit_general_settings([background_color, ...])	Edit general settings for the plot.
edit_grid([minor_x, minor_y, major_x, ...])	Edit the grid settings for the plot.
edit_header([company_name, ...])	Edit the plot header.
edit_legend([show_solution_name, ...])	Edit the plot legend.
edit_x_axis([font, font_size, italic, bold, ...])	Edit the X-axis settings.
edit_x_axis_scaling([linear_scaling, ...])	Edit the X-axis scaling settings.
edit_y_axis([axis_name, font, font_size, ...])	Edit the Y-axis settings.
edit_y_axis_scaling([axis_name, ...])	Edit the Y-axis scaling settings.
get_solution_data()	Get the report solution data.
hide_legend([solution_name, trace_name, ...])	Hide the Legend.
import_traces(file_path, plot_name)	Import report data from a file into a specified report.
update_expressions_with_defaults([...])	Update the list of expressions by taking all quantities from a given category.
update_trace_in_report(traces[, setup_name, ...])	Update a trace in a specific report.

Attributes

differential_pairs	Differential pairs flag.
domain	Plot domain.
expressions	Expressions.
far_field_sphere	Far field sphere name.
limit_lines	List of available limit lines in the report.
matrix	2D or Q3D matrix name.
notes	List of available notes in the report.
plot_name	Plot name.
polyline	Polyline name for the field report.
primary_sweep	Primary sweep report.
primary_sweep_range	Primary sweep range report.
report_category	Report category.
report_type	Report type.
secondary_sweep	Secondary sweep report.
secondary_sweep_range	Secondary sweep range report.
traces	List of available traces in the report.
use_pulse_in_tdr	Defines if the TDR should use a pulse or step.
variations	Variations.

`pyaedt.modules.report_templates.EyeDiagram`

```
class pyaedt.modules.report_templates.EyeDiagram(app, report_category, setup_name,  
    expressions=None)
```

Provides for managing eye diagram reports.

```
__init__(app, report_category, setup_name, expressions=None)
```

Methods

<code>add_all_eye_measurements()</code>	Add all eye measurements to the plot.
<code>add_cartesian_x_marker(<i>val</i>[, <i>name</i>])</code>	Add a cartesian X marker.
<code>add_cartesian_y_marker(<i>val</i>[, <i>name</i>, <i>y_axis</i>])</code>	Add a cartesian Y marker.
<code>add_limit_line_from_equation(<i>start_x</i>, ...[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_limit_line_from_points(<i>x_list</i>, <i>y_list</i>[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_note(<i>text</i>[, <i>x_position</i>, <i>y_position</i>])</code>	Add a note at a position.
<code>add_trace_characteristics(<i>trace_name</i>[, ...])</code>	Add a trace characteristic to the plot.
<code>add_trace_to_report(<i>traces</i>[, <i>setup_name</i>, ...])</code>	Add a trace to a specific report.
<code>clear_all_eye_measurements()</code>	Clear all eye measurements from the plot.
<code>create([<i>plot_name</i>])</code>	Create an eye diagram report.
<code>delete()</code>	Delete current report.
<code>delete_traces(<i>plot_name</i>, <i>traces_list</i>)</code>	Delete an existing trace or traces.
<code>edit_general_settings([<i>background_color</i>, ...])</code>	Edit general settings for the plot.
<code>edit_grid([<i>minor_x</i>, <i>minor_y</i>, <i>major_x</i>, ...])</code>	Edit the grid settings for the plot.
<code>edit_header([<i>company_name</i>, ...])</code>	Edit the plot header.
<code>edit_legend([<i>show_solution_name</i>, ...])</code>	Edit the plot legend.
<code>edit_x_axis([<i>font</i>, <i>font_size</i>, <i>italic</i>, <i>bold</i>, ...])</code>	Edit the X-axis settings.
<code>edit_x_axis_scaling([<i>linear_scaling</i>, ...])</code>	Edit the X-axis scaling settings.
<code>edit_y_axis([<i>axis_name</i>, <i>font</i>, <i>font_size</i>, ...])</code>	Edit the Y-axis settings.
<code>edit_y_axis_scaling([<i>axis_name</i>, ...])</code>	Edit the Y-axis scaling settings.
<code>export_maskViolation([<i>out_file</i>])</code>	Export the eye diagram mask violations to a TAB file.
<code>eye_mask(<i>points</i>[, <i>xunits</i>, <i>yunits</i>, ...])</code>	Create an eye diagram in the plot.
<code>get_solution_data()</code>	Get the report solution data.
<code>hide_legend([<i>solution_name</i>, <i>trace_name</i>, ...])</code>	Hide the Legend.
<code>import_traces(<i>file_path</i>, <i>plot_name</i>)</code>	Import report data from a file into a specified report.
<code>rectangular_plot([<i>value</i>])</code>	Enable or disable the rectangular plot on the chart.
<code>update_expressions_with_defaults([...])</code>	Update the list of expressions by taking all quantities from a given category.
<code>update_trace_in_report(<i>traces</i>[, <i>setup_name</i>, ...])</code>	Update a trace in a specific report.

Attributes

auto_compute_eye_meas	Flag for automatically computing eye measurements.
auto_cross_amplitude	Auto-cross amplitude flag.
auto_delay	Auto-delay flag.
cross_amplitude	Cross-amplitude value when auto_cross_amplitude=False.
differential_pairs	Differential pairs flag.
domain	Plot domain.
dy_dx_tolerance	DY DX tolerance.
expressions	Expressions.
eye_measurement_point	Eye measurement point.
limit_lines	List of available limit lines in the report.
manual_delay	Manual delay value when auto_delay=False.
matrix	2D or Q3D matrix name.
notes	List of available notes in the report.
offset	Offset value.
plot_name	Plot name.
polyline	Polyline name for the field report.
primary_sweep	Primary sweep report.
primary_sweep_range	Primary sweep range report.
quantity_type	Quantity type used in the AMI analysis plot.
report_category	Report category.
report_type	Report type.
secondary_sweep	Secondary sweep report.
secondary_sweep_range	Secondary sweep range report.
thinning	Thinning flag.
thinning_points	Number of thinning points.
time_start	Time start value.
time_stop	Time stop value.
traces	List of available traces in the report.
unit_interval	Unit interval value.
use_pulse_in_tdr	Defines if the TDR should use a pulse or step.
variations	Variations.

pyaedt.modules.report_templates.Emission

```
class pyaedt.modules.report_templates.Emission(app, report_category, setup_name, expressions=None)
```

Provides for managing emission reports.

```
__init__(app, report_category, setup_name, expressions=None)
```

Methods

<code>add_cartesian_x_marker(val[, name])</code>	Add a cartesian X marker.
<code>add_cartesian_y_marker(val[, name, y_axis])</code>	Add a cartesian Y marker.
<code>add_limit_line_from_equation(start_x, ...[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_limit_line_from_points(x_list, y_list[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_note(text[, x_position, y_position])</code>	Add a note at a position.
<code>add_trace_to_report(traces[, setup_name, ...])</code>	Add a trace to a specific report.
<code>create([plot_name])</code>	Create a report.
<code>delete()</code>	Delete current report.
<code>delete_traces(plot_name, traces_list)</code>	Delete an existing trace or traces.
<code>edit_general_settings([background_color, ...])</code>	Edit general settings for the plot.
<code>edit_grid([minor_x, minor_y, major_x, ...])</code>	Edit the grid settings for the plot.
<code>edit_header([company_name, ...])</code>	Edit the plot header.
<code>edit_legend([show_solution_name, ...])</code>	Edit the plot legend.
<code>edit_x_axis([font, font_size, italic, bold, ...])</code>	Edit the X-axis settings.
<code>edit_x_axis_scaling([linear_scaling, ...])</code>	Edit the X-axis scaling settings.
<code>edit_y_axis([axis_name, font, font_size, ...])</code>	Edit the Y-axis settings.
<code>edit_y_axis_scaling([axis_name, ...])</code>	Edit the Y-axis scaling settings.
<code>get_solution_data()</code>	Get the report solution data.
<code>hide_legend([solution_name, trace_name, ...])</code>	Hide the Legend.
<code>import_traces(file_path, plot_name)</code>	Import report data from a file into a specified report.
<code>update_expressions_with_defaults([...])</code>	Update the list of expressions by taking all quantities from a given category.
<code>update_trace_in_report(traces[, setup_name, ...])</code>	Update a trace in a specific report.

Attributes

<code>differential_pairs</code>	Differential pairs flag.
<code>domain</code>	Plot domain.
<code>expressions</code>	Expressions.
<code>limit_lines</code>	List of available limit lines in the report.
<code>matrix</code>	2D or Q3D matrix name.
<code>notes</code>	List of available notes in the report.
<code>plot_name</code>	Plot name.
<code>polyline</code>	Polyline name for the field report.
<code>primary_sweep</code>	Primary sweep report.
<code>primary_sweep_range</code>	Primary sweep range report.
<code>report_category</code>	Report category.
<code>report_type</code>	Report type.
<code>secondary_sweep</code>	Secondary sweep report.
<code>secondary_sweep_range</code>	Secondary sweep range report.
<code>traces</code>	List of available traces in the report.
<code>use_pulse_in_tdr</code>	Defines if the TDR should use a pulse or step.
<code>variations</code>	Variations.

pyaedt.modules.report_templates.Spectral**class** `pyaedt.modules.report_templates.Spectral`(*app*, *report_category*, *setup_name*, *expressions=None*)

Provides for managing spectral reports from transient data.

`__init__`(*app*, *report_category*, *setup_name*, *expressions=None*)**Methods**

<code>add_cartesian_x_marker(val[, name])</code>	Add a cartesian X marker.
<code>add_cartesian_y_marker(val[, name, y_axis])</code>	Add a cartesian Y marker.
<code>add_limit_line_from_equation(start_x, ...[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_limit_line_from_points(x_list, y_list[, ...])</code>	Add a Cartesian limit line from point lists.
<code>add_note(text[, x_position, y_position])</code>	Add a note at a position.
<code>add_trace_to_report(traces[, setup_name, ...])</code>	Add a trace to a specific report.
<code>create([plot_name])</code>	Create an eye diagram report.
<code>delete()</code>	Delete current report.
<code>delete_traces(plot_name, traces_list)</code>	Delete an existing trace or traces.
<code>edit_general_settings([background_color, ...])</code>	Edit general settings for the plot.
<code>edit_grid([minor_x, minor_y, major_x, ...])</code>	Edit the grid settings for the plot.
<code>edit_header([company_name, ...])</code>	Edit the plot header.
<code>edit_legend([show_solution_name, ...])</code>	Edit the plot legend.
<code>edit_x_axis([font, font_size, italic, bold, ...])</code>	Edit the X-axis settings.
<code>edit_x_axis_scaling([linear_scaling, ...])</code>	Edit the X-axis scaling settings.
<code>edit_y_axis([axis_name, font, font_size, ...])</code>	Edit the Y-axis settings.
<code>edit_y_axis_scaling([axis_name, ...])</code>	Edit the Y-axis scaling settings.
<code>get_solution_data()</code>	Get the report solution data.
<code>hide_legend([solution_name, trace_name, ...])</code>	Hide the Legend.
<code>import_traces(file_path, plot_name)</code>	Import report data from a file into a specified report.
<code>update_expressions_with_defaults([...])</code>	Update the list of expressions by taking all quantities from a given category.
<code>update_trace_in_report(traces[, setup_name, ...])</code>	Update a trace in a specific report.

Attributes

adjust_coherent_gain	Coherent gain flag.
differential_pairs	Differential pairs flag.
domain	Plot domain.
expressions	Expressions.
kaiser_coeff	Kaiser value.
limit_lines	List of available limit lines in the report.
matrix	2D or Q3D matrix name.
max_frequency	Maximum spectrum frequency.
notes	List of available notes in the report.
plot_continious_spectrum	Continuous spectrum flag.
plot_name	Plot name.
polyline	Polyline name for the field report.
primary_sweep	Primary sweep report.
primary_sweep_range	Primary sweep range report.
report_category	Report category.
report_type	Report type.
secondary_sweep	Secondary sweep report.
secondary_sweep_range	Secondary sweep range report.
time_start	Time start value.
time_stop	Time stop value.
traces	List of available traces in the report.
use_pulse_in_tdr	Defines if the TDR should use a pulse or step.
variations	Variations.
window	Window value.

2.12.6 Plot fields and data outside AEDT

PyAEDT supports external report capabilities available with installed third-party packages like `numpy`, `pandas`, `matplotlib`, and `pyvista`.

<code>solutions.SolutionData</code>	Contains information from the <code>GetSolutionDataPerVariation()</code> method.
<code>solutions.FieldPlot</code>	Provides for creating and editing field plots.
<code>solutions.FfdSolutionData</code>	Provides antenna array far-field data.
<code>AdvancedPostProcessing.ModelPlotter</code>	Manages the data to be plotted with <code>pyvista</code> .

`pyaedt.modules.AdvancedPostProcessing.ModelPlotter`

`class pyaedt.modules.AdvancedPostProcessing.ModelPlotter`

Manages the data to be plotted with `pyvista`.

Examples

This Class can be instantiated within Pyaedt (with plot_model_object or different field plots and standalone). Here an example of standalone project

```
>>> model = ModelPlotter()
>>> model.add_object(r'D:\Simulation\antenna.obj', (200,20,255), 0.6, "in")
>>> model.add_object(r'D:\Simulation\helix.obj', (0,255,0), 0.5, "in")
>>> model.add_field_from_file(r'D:\Simulation\helic_antenna.csv', True, "meter", 1)
>>> model.background_color = (0,0,0)
>>> model.plot()
```

And here an example of animation:

```
>>> model = ModelPlotter()
>>> model.add_object(r'D:\Simulation\antenna.obj', (200,20,255), 0.6, "in")
>>> model.add_object(r'D:\Simulation\helix.obj', (0,255,0), 0.5, "in")
>>> frames = [r'D:\Simulation\helic_antenna.csv', r'D:\Simulation\helic_antenna_10.
->fld',
...             r'D:\Simulation\helic_antenna_20.fld', r'D:\Simulation\helic_antenna_
->30.fld',
...             r'D:\Simulation\helic_antenna_40.fld']
>>> model.gif_file = r"D:\Simulation\animation.gif"
>>> model.animate()
```

`__init__()`

Methods

<code>add_field_from_data(coordinates, fields_data)</code>	Add field data to the scenario.
<code>add_field_from_file(field_path[, log_scale, ...])</code>	Add a field file to the scenario.
<code>add_frames_from_file(field_files[, ...])</code>	Add a field file to the scenario.
<code>add_object(cad_path[, cad_color, opacity, units])</code>	Add a mesh file to the scenario.
<code>animate()</code>	Animate the current field plot.
<code>clean_cache_and_files([remove_objs, ...])</code>	Clean downloaded files, and, on demand, also the cached meshes.
<code>generate_geometry_mesh()</code>	Generate mesh for objects only.
<code>plot([export_image_path, show])</code>	Plot the current available Data.
<code>set_orientation([camera_position, ...])</code>	Change the plot default orientation.

Attributes

<code>azimuth_angle</code>	Get/Set the azimuth angle value.
<code>background_color</code>	Background color.
<code>background_image</code>	Background image.
<code>camera_position</code>	Get or set the camera position value.
<code>convert_fields_in_db</code>	Either if convert the fields before plotting in dB.
<code>elevation_angle</code>	Get/Set the elevation angle value.
<code>fields</code>	List of fields object.
<code>focal_point</code>	Get/Set the camera focal point value.
<code>frames</code>	Frames list for animation.
<code>isometric_view</code>	Enable or disable the default iso view.
<code>log_multiplier</code>	Multiply the log value.
<code>objects</code>	List of class objects.
<code>roll_angle</code>	Get/Set the roll angle value.
<code>view_up</code>	Get/Set the camera view axis.
<code>x_scale</code>	Scale plot on X.
<code>y_scale</code>	Scale plot on Y.
<code>z_scale</code>	Scale plot on Z.
<code>zoom</code>	Get/Set the zoom value.

2.12.7 Icepak monitors

<code>FaceMonitor</code>	Provides Icepak face monitor properties and methods.
<code>PointMonitor</code>	Provides Icepak point monitor methods and properties.
<code>Monitor</code>	Provides Icepak monitor methods.

`pyaedt.modules.monitor_icepak.FaceMonitor`

```
class pyaedt.modules.monitor_icepak.FaceMonitor(monitor_name, monitor_type, face_id, quantity, app)
    Provides Icepak face monitor properties and methods.

    __init__(monitor_name, monitor_type, face_id, quantity, app)
```

Methods

<code>delete()</code>	Delete a monitor object.
<code>value([quantity, setup, ...])</code>	Get a list of values obtained from the monitor object.

Attributes

geometry_assignment	Get the geometry assignment for the monitor object.
id	Get the name, or id of geometry assignment.
location	Get the monitor location in terms of face or surface center.
name	Get the name of the monitor object.
properties	Get a dictionary of properties.
quantities	Get the quantities being monitored.
type	Get the monitor type.

pyaedt.modules.monitor_icepak.PointMonitor

```
class pyaedt.modules.monitor_icepak.PointMonitor(monitor_name, monitor_type, point_id, quantity, app)
```

Provides Icepak point monitor methods and properties.

```
__init__(monitor_name, monitor_type, point_id, quantity, app)
```

Methods

delete()	Delete a monitor object.
value([quantity, setup, ...])	Get a list of values obtained from the monitor object.

Attributes

geometry_assignment	Get the geometry assignment for the monitor object.
id	Get the name, or id of geometry assignment.
location	Get the monitor point location.
name	Get the name of the monitor object.
properties	Get a dictionary of properties.
quantities	Get the quantities being monitored.
type	Get the monitor type.

pyaedt.modules.monitor_icepak.Monitor

```
class pyaedt.modules.monitor_icepak.Monitor(p_app)
```

Provides Icepak monitor methods.

```
__init__(p_app)
```

Methods

<code>assign_face_monitor(face_id[, ...])</code>	Assign a face monitor.
<code>assign_point_monitor(point_position[, ...])</code>	Create and assign a point monitor.
<code>assign_point_monitor_in_object(name[, ...])</code>	Assign a point monitor in the centroid of a specific object.
<code>assign_point_monitor_to_vertex(vertex_id[, ...])</code>	Create and assign a point monitor to a vertex.
<code>assign_surface_monitor(surface_name[, ...])</code>	Assign a surface monitor.
<code>delete_monitor(monitor_name)</code>	Delete monitor object.
<code>get_icepak_monitor_object(monitor_name)</code>	Get Icepak monitor object.
<code>get_monitor_object_assignment(monitor)</code>	Get the object that the monitor is applied to.
<code>insert_monitor_object_from_dict(monitor_dict)</code>	Insert a monitor.

Attributes

<code>all_monitors</code>	Get all monitor objects.
<code>face_monitors</code>	Get point monitor objects.
<code>point_monitors</code>	Get face monitor objects.

2.13 Logger

This section lists modules for creating and editing PyAEDT log files.

<code>AedtLogger</code>	Specifies the logger to use for each AEDT logger.
<code>AppFilter</code>	Specifies the destination of the logger.

2.13.1 pyaedt.aedt_logger.AedtLogger

`class pyaedt.aedt_logger.AedtLogger(level=10, filename=None, to_stdout=False, desktop=None)`

Specifies the logger to use for each AEDT logger.

This class allows you to add a handler to write messages to a file and to indicate whether to write messages to the standard output (stdout).

Parameters

level

[`int`, optional] Logging level to filter the message severity allowed in the logger. The default is `logging.DEBUG`.

filename

[`str`, optional] Name of the file to write messages to. The default is `None`.

to_stdout

[`bool`, optional] Whether to write log messages to stdout. The default is `False`.

`__init__(level=10, filename=None, to_stdout=False, desktop=None)`

Methods

<code>add_debug_message(message_text[, level])</code>	Parameterized message to the message manager to specify the type and project or design level.
<code>add_error_message(message_text[, level])</code>	Add a type 2 "Error" message to the message manager tree.
<code>add_file_logger(filename, project_name[, level])</code>	Add a new file to the logger handlers list.
<code>add_info_message(message_text[, level])</code>	Add a type 0 "Info" message to the active design level of the message manager tree.
<code>add_logger(destination[, level])</code>	Add a logger for either the active project or active design.
<code>add_message(message_type, message_text[, ...])</code>	Add a message to the message manager to specify the type and project or design level.
<code>add_warning_message(message_text[, level])</code>	Add a type 1 "Warning" message to the message manager tree.
<code>clear_messages([proj_name, des_name, level])</code>	Clear all messages.
<code>debug(msg, *args, **kwargs)</code>	Write a debug message to the global logger.
<code>disable_desktop_log()</code>	Disable the log in AEDT.
<code>disable_log_on_file()</code>	Disable writing log messages to an output file.
<code>disable_stdout_log()</code>	Disable printing log messages to stdout.
<code>enable_desktop_log()</code>	Enable the log in AEDT.
<code>enable_log_on_file()</code>	Enable writing log messages to an output file.
<code>enable_stdout_log()</code>	Enable printing log messages to stdout.
<code>error(msg, *args, **kwargs)</code>	Write an error message to the global logger.
<code>get_messages([project_name, design_name, ...])</code>	Get the message manager content for a specified project and design.
<code>info(msg, *args, **kwargs)</code>	Write an info message to the global logger.
<code>info_timer(msg[, start_time])</code>	Write an info message to the global logger with elapsed time.
<code>remove_all_project_file_logger()</code>	Remove all the local files from the logger handlers list.
<code>remove_file_logger(project_name)</code>	Remove a file from the logger handlers list.
<code>reset_timer([time_val])</code>	Reset actual timer to actual time or specified time.
<code>warning(msg, *args, **kwargs)</code>	Write a warning message to the global logger.

Attributes

aedt_error_messages	Message manager content for the active project and design.
aedt_info_messages	Message manager content for the active project and design.
aedt_messages	Message manager content for the active project and design.
aedt_warning_messages	Message manager content for the active project and design.
design_logger	Design logger.
design_name	Name of current logger design.
error_messages	Message manager content for the active pyaedt session.
glb	Global logger.
info_messages	Message manager content for the active pyaedt session.
logger	AEDT logger object.
messages	Message manager content for the active session.
non_graphical	Check if desktop is graphical or not.
odesign	Design object.
oproject	Project object.
project_logger	Project logger.
project_name	Name of current logger project.
warning_messages	Message manager content for the active pyaedt session.

2.13.2 pyaedt.aedt_logger.AppFilter

```
class pyaedt.aedt_logger.AppFilter(destination='Global', extra="")
```

Specifies the destination of the logger.

AEDT exposes three different loggers, which are the global, project, and design loggers.

Parameters

destination

[**str**, optional] Logger to write to. Options are "Global", ``"Project", and "Design". The default is "Global".

extra

[**str**, optional] Name of the design or project. The default is "".

```
__init__(destination='Global', extra="")
```

Initialize a filter.

Initialize with the name of the logger which, together with its children, will have its events allowed through the filter. If no name is specified, allow every event.

Methods

filter(record)	Modify the record sent to the logger.
----------------	---------------------------------------

2.14 Optimetrics

This module contains all properties and methods needed to create optimetrics setups.

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# returns the ParametericsSetups Class
app.parametrics

# returns the OptimizationSetups Class
app.optimizations

# adds an optimization and returns Setup class with all settings and methods
sweep3 = hfss.opti_optimization.add_optimization(calculation="dB(S(1,1))", calculation_
→value="2.5GHz")

...
```

ParametricSetups	Sets up Parametrics analyses.
OptimizationSetups	Sets up optimizations.
SetupParam	Sets up a parametric analysis in Optimetrics.
SetupOpti	Sets up an optimization in Optimetrics.

2.14.1 pyaedt.modules.DesignXPloration.ParametricSetups

```
class pyaedt.modules.DesignXPloration.ParametricSetups(p_app)
```

Sets up Parametrics analyses. It includes Parametrics, Sensitivity and Statistical Analysis.

Examples

```
>>> from pyaedt import Hfss
>>> app = Hfss()
>>> sensitivity_setups = app.parametrics
```

```
__init__(p_app)
```

Methods

<code>add(sweep_var, start_point[, end_point, ...])</code>	Add a basic sensitivity analysis.
<code>add_from_file(filename[, parametricname])</code>	Add a Parametric Setup from a csv file.
<code>delete(setup_name)</code>	Delete a defined Parametric Setup.

Attributes

<code>optimodule</code>	Optimetrics module.
<code>p_app</code>	Parent.

2.14.2 pyaedt.modules.DesignXPloration.OptimizationSetups

```
class pyaedt.modules.DesignXPloration.OptimizationSetups(p_app)
```

Sets up optimizations. It includes Optimization, DOE and DesignXplorer Analysis.

Examples

```
>>> from pyaedt import Hfss
>>> app = Hfss()
>>> optimization_setup = app.optimizations
```

```
__init__(p_app)
```

Methods

<code>add([calculation, ranges, variables, ...])</code>	Add a basic optimization analysis.
<code>delete(setup_name)</code>	Delete a defined Optimetrics Setup.

Attributes

<code>optimodule</code>	Optimetrics module.
<code>p_app</code>	Parent.

2.14.3 pyaedt.modules.DesignXPloration.SetupParam

```
class pyaedt.modules.DesignXPloration.SetupParam(p_app, name, dictinputs=None,  
                                                optim_type='OptiParametric')
```

Sets up a parametric analysis in Optimetrics.

```
__init__(p_app, name, dictinputs=None, optim_type='OptiParametric')
```

Methods

add_calculation(calculation, ranges[, ...])	Add a calculation to the parametric setup.
add_variation(sweep_var, start_point[, ...])	Add a variation to an existing parametric setup.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Create a setup.
delete()	Delete a defined Optimetrics Setup.
export_to_csv(filename)	Export the current Parametric Setup to csv.
sync_variables(variables[, sync_n])	Sync variable variations in an existing parametric setup.
update([update_dictionary])	Update the setup based on stored properties.

Attributes

available_properties	Available properties.
----------------------	-----------------------

2.14.4 pyaedt.modules.DesignXPloration.SetupOpti

```
class pyaedt.modules.DesignXPloration.SetupOpti(app, name, dictinputs=None,  
                                               optim_type='OptiDesignExplorer')
```

Sets up an optimization in Opimetrics.

```
__init__(app, name, dictinputs=None, optim_type='OptiDesignExplorer')
```

Methods

add_calculation(calculation[, ranges, ...])	Add a calculation to the setup.
add_goal(calculation, ranges[, variables, ...])	Add a goal to the setup.
add_variation(variable_name, min_value, ...)	Add a new variable as input for the optimization and defines its ranges.
analyze([cores, tasks, gpus, acf_file, ...])	Solve the active design.
create()	Create a setup.
delete()	Delete a defined Optimetrics Setup.
update([update_dictionary])	Update the setup based on stored properties.

Attributes

<code>available_properties</code>	Available properties.
-----------------------------------	-----------------------

2.15 Variable

This module provides all functionalities for creating and editing design and project variables in the 3D tools.

```
from pyaedt import Hfss
app = Hfss(specified_version="2023.1",
            non_graphical=False, new_desktop_session=True,
            close_on_exit=True, student_version=False)

# This call returns the VariableManager class
variable_manager = self.aedtapp._variable_manager

# Set and get a variable
app["w"] = "10mm"
a = app["w"]
...
```

<code>VariableManager</code>	Manages design properties and project variables.
<code>Variable</code>	Stores design properties and project variables and provides operations to perform on them.
<code>DataSet</code>	Manages datasets.
<code>CSVDataset</code>	Reads in a CSV file and extracts data, which can be augmented with constant values.

2.15.1 `pyaedt.application.Variables.VariableManager`

```
class pyaedt.application.Variables.VariableManager(app)
```

Manages design properties and project variables.

Design properties are the local variables in a design. Project variables are defined at the project level and start with \$.

This class provides access to all variables or a subset of the variables. Manipulation of the numerical or string definitions of variable values is provided in the `pyaedt.application.Variables.Variable` class.

Parameters

`variables`

[`dict`] Dictionary of all design properties and project variables in the active design.

`design_variables`

[`dict`] Dictionary of all design properties in the active design.

`project_variables`

[`dict`] Dictionary of all project variables available to the active design (key by variable name).

dependent_variables

[`dict`] Dictionary of all dependent variables available to the active design (key by variable name).

independent_variables

[`dict`] Dictionary of all independent variables (constant numeric values) available to the active design (key by variable name).

independent_design_variables

[`dict`]

independent_project_variables

[`dict`]

variable_names

[`str` or `list`] One or more variable names.

project_variable_names

[`str` or `list`] One or more project variable names.

design_variable_names

[`str` or `list`] One or more design variable names.

dependent_variable_names

[`str` or `list`] All dependent variable names within the project.

independent_variable_names

[`list` of `str`] All independent variable names within the project. These can be sweep variables for optimetrics.

independent_project_variable_names

[`str` or `list`] All independent project variable names within the project. These can be sweep variables for optimetrics.

independent_design_variable_names

[`str` or `list`] All independent design properties (local variables) within the project. These can be sweep variables for optimetrics.

See also:

[`pyaedt.application.Variables.Variable`](#)

Examples

```
>>> from pyaedt.maxwell import Maxwell3d
>>> from pyaedt.desktop import Desktop
>>> d = Desktop()
>>> aedtapp = Maxwell3d()
```

Define some test variables.

```
>>> aedtapp["Var1"] = 3
>>> aedtapp["Var2"] = "12deg"
>>> aedtapp["Var3"] = "Var1 * Var2"
>>> aedtapp["$PrjVar1"] = "pi"
```

Get the variable manager for the active design.

```
>>> v = aedtapp.variable_manager
```

Get a dictionary of all project and design variables.

```
>>> v.variables
{'Var1': <pyaedt.application.Variables.Variable at 0x2661f34c448>,
 'Var2': <pyaedt.application.Variables.Variable at 0x2661f34c308>,
 'Var3': <pyaedt.application.Variables.Expression at 0x2661f34cb48>,
 '$PrjVar1': <pyaedt.application.Variables.Expression at 0x2661f34cc48>}
```

Get a dictionary of only the design variables.

```
>>> v.design_variables
{'Var1': <pyaedt.application.Variables.Variable at 0x2661f339508>,
 'Var2': <pyaedt.application.Variables.Variable at 0x2661f3415c8>,
 'Var3': <pyaedt.application.Variables.Expression at 0x2661f341808>}
```

Get a dictionary of only the independent design variables.

```
>>> v.independent_design_variables
{'Var1': <pyaedt.application.Variables.Variable at 0x2661f335d08>,
 'Var2': <pyaedt.application.Variables.Variable at 0x2661f3557c8>}
```

`__init__(app)`

Methods

<code>aedt_object(variable)</code>	Retrieve an AEDT object.
<code>decompose(variable_value)</code>	Decompose a variable string to a floating with its unit.
<code>delete_separator(separator_name)</code>	Delete a separator from either the active project or design.
<code>delete_unused_variables()</code>	Delete unused design and project variables.
<code>delete_variable(var_name)</code>	Delete a variable.
<code>get_expression(variable_name)</code>	Retrieve the variable value of a project or design variable as a string.
<code>is_used(var_name)</code>	Find if a variable is used.
<code>set_variable(variable_name[, expression, ...])</code>	Set the value of a design property or project variable.

Attributes

dependent_design_variable_names	List of dependent design variables.
dependent_design_variables	Dependent design variables.
dependent_project_variable_names	List of dependent project variables.
dependent_project_variables	Dependent project variables.
dependent_variable_names	List of dependent variables.
dependent_variables	Dependent variables.
design_variable_names	List of design variables.
design_variables	Design variables.
independent_design_variable_names	List of independent design variables.
independent_design_variables	Independent design variables.
independent_project_variable_names	List of independent project variables.
independent_project_variables	Independent project variables.
independent_variable_names	List of independent variables.
independent_variables	Independent variables.
post_processing_variables	Post Processing variables.
project_variable_names	List of project variables.
project_variables	Project variables.
variable_names	List of variables.
variables	Variables.

2.15.2 pyaedt.application.Variables.Variable

```
class pyaedt.application.Variables.Variable(expression, units=None, si_value=None,  
                                             full_variables=None, name=None, app=None,  
                                             readonly=False, hidden=False, description=None,  
                                             postprocessing=False, circuit_parameter=True)
```

Stores design properties and project variables and provides operations to perform on them.

Parameters

value

[`float`, `str`] Numerical value of the variable in SI units.

units

[`str`] Units for the value.

Examples

```
>>> from pyaedt.application.Variables import Variable
```

Define a variable using a string value consistent with the AEDT properties.

```
>>> v = Variable("45mm")
```

Define an unitless variable with a value of 3.0.

```
>>> v = Variable(3.0)
```

Define a variable defined by a numeric result and a unit string.

```
>>> v = Variable(3.0 * 4.5, units="mm")
>>> assert v.numeric_value == 13.5
>>> assert v.units == "mm"
```

`__init__(expression, units=None, si_value=None, full_variables=None, name=None, app=None, readonly=False, hidden=False, description=None, postprocessing=False, circuit_parameter=True)`

Methods

<code>decompose()</code>	Decompose a variable value to a floating with its unit.
<code>format(format)</code>	Retrieve the string value with the specified numerical formatting.
<code>rescale_to(units)</code>	Rescale the expression to a new unit within the current unit system.

Attributes

<code>circuit_parameter</code>	Circuit parameter flag value.
<code>description</code>	Description value.
<code>evaluated_value</code>	String value.
<code>expression</code>	Expression.
<code>hidden</code>	Hidden flag value.
<code>is_optimization_enabled</code>	"Check if optimization is enabled.
<code>is_sensitivity_enabled</code>	Check if Sensitivity is enabled.
<code>is_statistical_enabled</code>	Check if statistical is enabled.
<code>is_tuning_enabled</code>	Check if tuning is enabled.
<code>name</code>	Variable name.
<code>numeric_value</code>	Numeric part of the expression as a float value.
<code>optimization_max_value</code>	"Optimization max value.
<code>optimization_min_value</code>	"Optimization min value.
<code>post_processing</code>	Postprocessing flag value.
<code>read_only</code>	Read-only flag value.
<code>sensitivity_initial_disp</code>	"Sensitivity initial value.
<code>sensitivity_max_value</code>	"Sensitivity max value.
<code>sensitivity_min_value</code>	"Sensitivity min value.
<code>tuning_max_value</code>	"Tuning max value.
<code>tuning_min_value</code>	"Tuning min value.
<code>tuning_step_value</code>	"Tuning Step value.
<code>unit_system</code>	Unit system of the expression as a string.
<code>units</code>	Units.
<code>value</code>	Value.

2.15.3 pyaedt.application.Variables.DataSet

```
class pyaedt.application.Variables.DataSet(app, name, x, y, z=None, v=None, xunit="", yunit="",
                                           zunit="", vunit="")
```

Manages datasets.

Parameters

app
name
 [str] Name of the app.
x
 [list] List of X-axis values for the dataset.
y
 [list] List of Y-axis values for the dataset.
z
 [list, optional] List of Z-axis values for a 3D dataset only. The default is None.
v
 [list, optional] List of V-axis values for a 3D dataset only. The default is None.
xunit
 [str, optional] Units for the X axis. The default is "".
yunit
 [str, optional] Units for the Y axis. The default is "".
zunit
 [str, optional] Units for the Z axis for a 3D dataset only. The default is "".
vunit
 [str, optional] Units for the V axis for a 3D dataset only. The default is "".
__init__(app, name, x, y, z=None, v=None, xunit="", yunit="", zunit="", vunit="")

Methods

<code>add_point(x, y[, z, v])</code>	Add a point to the dataset.
<code>create()</code>	Create a dataset.
<code>delete()</code>	Delete the dataset.
<code>export([output_dir])</code>	Export the dataset.
<code>remove_point_from_index(id_to_remove)</code>	Remove a point from an index.
<code>remove_point_from_x(x)</code>	Remove a point from an X-axis value.
<code>update()</code>	Update the dataset.

2.15.4 pyaedt.application.Variables.CSVDataset

```
class pyaedt.application.Variables.CSVDataset(csv_file=None, separator=None, units_dict=None,
                                              append_dict=None, valid_solutions=True,
                                              invalid_solutions=False)
```

Reads in a CSV file and extracts data, which can be augmented with constant values.

Parameters

csv_file

[`str`, optional] Input file consisting of delimited data with the first line as the header. The CSV value includes the header and data, which supports AEDT units information such as "1.23Wb". You can also augment the data with constant values.

separator

[`str`, optional] Value to use for the delimiter. The default is ``None`` in which case a comma is assumed.

units_dict

[`dict`, optional] Dictionary consisting of {Variable Name: unit} to rescale the data if it is not in the desired unit system.

append_dict

[`dict`, optional] Dictionary consisting of {New Variable Name: value} to add variables with constant values to all data points. This dictionary is used to add multiple sweeps to one result file.

valid_solutions

[`bool`, optional] The default is True.

invalid_solutions

[`bool`, optional] The default is False.

```
__init__(csv_file=None, separator=None, units_dict=None, append_dict=None, valid_solutions=True,
        invalid_solutions=False)
```

Methods

<code>next()</code>	Yield the next row.
---------------------	---------------------

Attributes

<code>data</code>	Data.
<code>header</code>	Header.
<code>number_of_columns</code>	Number of columns.
<code>number_of_rows</code>	Number of rows.
<code>path</code>	Path.

2.16 Constants

This section lists constants that are commonly used in PyAEDT.

Example of constants usage:

```
from pyaedt import constants
ipk = Icepak()
# Use of AXIS Constant
cylinder = ipk.modeler.create_cylinder(constants.AXIS.X, [0,0,0],10,3)
# Use of PLANE Constant
ipk.modeler.split(cylinder, constants.PLANE.YZ, sides="Both")
...
ipk.release_desktop()
```

class pyaedt.generic.constants.AXIS

CoordinateSystemAxis Enumerator class.

class pyaedt.generic.constants.BasisOrder

Enumeration-class for HFSS basis order settings.

Warning: the value single has been renamed to Single for consistency. Please update references to single.

class pyaedt.generic.constants.CATEGORIESQ3D

Plot Categories for Q2d and Q3d.

class pyaedt.generic.constants.CROSSESECTION

CROSSESECTION Enumerator class.

class pyaedt.generic.constants.CSMODE

COORDINATE SYSTEM MODE Enumerator class.

class pyaedt.generic.constants.FILLET

FilletType Enumerator class.

class pyaedt.generic.constants.FlipChipOrientation

Chip orientation enumerator class.

class pyaedt.generic.constants.GLOBALCS

GlobalCS Enumerator class.

class pyaedt.generic.constants.GRAVITY

GravityDirection Enumerator class.

class pyaedt.generic.constants.INFINITE_SPHERE_TYPE

INFINITE_SPHERE_TYPE Enumerator class.

class pyaedt.generic.constants.LineStyle

Provides trace line style constants.

class pyaedt.generic.constants.MATRIXOPERATIONSQ2D

Matrix Reduction types.

class pyaedt.generic.constants.MATRIXOPERATIONSQ3D

Matrix Reduction types.

```
class pyaedt.generic.constants.NodeType
    Type of node for source creation.

class pyaedt.generic.constants.PLANE
    CoordinateSystemPlane Enumerator class.

class pyaedt.generic.constants.SEGMENTTYPE
    CROSSSECTION Enumerator class.

class pyaedt.generic.constants.SETUPS
    Provides constants for the default setup types.

class pyaedt.generic.constants.SOLUTIONS
    Provides the names of default solution types.

class Circuit
    Provides Circuit solution types.

class Hfss
    Provides HFSS solution types.

class Icepak
    Provides Icepak solution types.

class Maxwell2d
    Provides Maxwell 2D solution types.

class Maxwell3d
    Provides Maxwell 3D solution types.

class Mechanical
    Provides Mechanical solution types.

class pyaedt.generic.constants.SWEEPDRIFT
    SweepDraftType Enumerator class.

class pyaedt.generic.constants.SolverType
    Provides solver type classes.

class pyaedt.generic.constants.SourceType
    Type of excitation enumerator.

class pyaedt.generic.constants.SymbolStyle
    Provides symbol style constants.

class pyaedt.generic.constants.TraceType
    Provides trace type constants.

class pyaedt.generic.constants.VIEW
    View Enumerator class.

pyaedt.generic.constants.cel2kel(val, inverse=True)
    Convert a temperature from Celsius to Kelvin.
```

Parameters

val
[float] Temperature value in Celsius.

inverse

[bool, optional] The default is True.

Returns**float**

Temperature value converted to Kelvin.

`pyaedt.generic.constants.db10(x, inverse=True)`

Convert db10 to decimal and vice versa.

`pyaedt.generic.constants.db20(x, inverse=True)`

Convert db20 to decimal and vice versa.

`pyaedt.generic.constants.dbm(x, inverse=True)`

Convert W to decimal and vice versa.

`pyaedt.generic.constants.dbw(x, inverse=True)`

Convert W to decimal and vice versa.

`pyaedt.generic.constants.fah2kel(val, inverse=True)`

Convert a temperature from Fahrenheit to Kelvin.

Parameters**val**

[**float**] Temperature value in Fahrenheit.

inverse

[bool, optional] The default is True.

Returns**float**

Temperature value converted to Kelvin.

`pyaedt.generic.constants.scale_units(scale_to_unit)`

Find the scale_to_unit into main system unit.

Parameters**scale_to_unit**

[**str**] Unit to Scale.

Returns**float**

Return the scaling factor if any.

`pyaedt.generic.constants.unit_converter(values, unit_system='Length', input_units='meter', output_units='mm')`

Convert unit in specified unit system.

Parameters**values**

[**float, list**] Values to convert.

unit_system

[**str**] Unit system. Default is *Length*.

input_units

[**str**] Input units. Default is *meter*.

output_units

[str] Output units. Default is *mm*.

Returns**float, list**

Converted value.

`pyaedt.generic.constants.unit_system(units)`

Retrieve the name of the unit system associated with a unit string.

Parameters**units**

[str] Units for retrieving the associated unit system name.

Returns**str**

Key from the AEDT_units when successful. For example, "AngularSpeed".

False when the units specified are not defined in AEDT units.

`pyaedt.generic.constants.validate_enum_class_value(cls, value)`

Check whether the value for the class enumeration-class is valid.

Parameters**cls**

[class] Enumeration-style class with integer members in range(0, N) where `cls.Invalid` equals N-1.

value

[int] Value to check.

Returns**bool**

True when the value is valid for the enumeration-class, False otherwise.

2.17 Configuration files

This module contains all methods to export project settings to a JSON file and import and apply settings to a new design. Currently the configuration cover the following apps: * HFSS * Q2D and Q3D Extractor * Maxwell * Icepak * Mechanical

The sections covered are:

- Variables
- Mesh operations
- Setup and optimetrics
- Material properties
- Object properties
- Boundaries and excitations

When a boundary is attached to a face, the tool tries to match it with a FaceByPosition on the same object name on the target design. If, for any reason, this face position has changed or the object name in the target design has changed, the boundary fails to apply.

<i>Configurations</i>	Enables export and import of a JSON configuration file that can be applied to a new or existing design.
<i>ConfigurationsOptions</i>	Options class for the configurations.
<i>ImportResults</i>	Contains the results of the import operations.

2.17.1 pyaedt.generic.configurations.Configurations

```
class pyaedt.generic.configurations.Configurations(app)
```

Enables export and import of a JSON configuration file that can be applied to a new or existing design.

```
__init__(app)
```

Methods

<code>export_config([config_file, overwrite])</code>	Export current design properties to a JSON or TOML file.
<code>import_config(config_file, *args)</code>	Import configuration settings from a JSON or TOML file and apply it to the current design.
<code>validate(config)</code>	Validate a configuration file against the schema. The default schema

2.17.2 pyaedt.generic.configurations.ConfigurationsOptions

```
class pyaedt.generic.configurations.ConfigurationsOptions(is_layout=False)
```

Options class for the configurations. User can enable or disable import export components.

```
__init__(is_layout=False)
```

Methods

<code>set_all_export()</code>	Set all export properties to <i>True</i> .
<code>set_all_import()</code>	Set all import properties to <i>True</i> .
<code>unset_all_export()</code>	Set all export properties to <i>False</i> .
<code>unset_all_import()</code>	Set all import properties to <i>False</i> .

Attributes

<code>export_boundaries</code>	Define if the boundaries have to be exported to json file.
<code>export_coordinate_systems</code>	Define if the Coordinate Systems have to be exported to json file.
<code>export_datasets</code>	Define if datasets have to be exported to json file.
<code>export_materials</code>	Define if the materials have to be exported to json file.
<code>export_mesh_operations</code>	Define if the Mesh Operations have to be exported to json file.
<code>export_object_properties</code>	Define if object properties have to be exported to json file.
<code>export_optimizations</code>	Define if the optimizations have to be exported to json file.
<code>export_parametrics</code>	Define if the parametrics have to be exported to json file.
<code>export_setups</code>	Define if the setups have to be exported to json file.
<code>export_variables</code>	Define if the variables have to be exported into json file.
<code>import_boundaries</code>	Define if the boundaries have to be imported/created from json file.
<code>import_coordinate_systems</code>	Define if the Coordinate Systems have to be imported/created from json file.
<code>import_datasets</code>	Define if datasets have to be imported from json file.
<code>import_materials</code>	Define if the materials have to be imported/created from json file.
<code>import_mesh_operations</code>	Define if the Mesh Operations have to be imported/created from json file.
<code>import_object_properties</code>	Define if object properties have to be imported/created from json file.
<code>import_optimizations</code>	Define if the optimizations have to be imported/created from json file.
<code>import_output_variables</code>	Define if the output variables have to be imported/created from json file.
<code>import_parametrics</code>	Define if the parametrics have to be imported/created from json file.
<code>import_setups</code>	Define if the setups have to be imported/created from json file.
<code>import_variables</code>	Define if the variablbes have to be imported/created from json file.
<code>object_mapping_tolerance</code>	Get/Set the tolerance value to be used in the object mapping (used e.g. for boundaries).
<code>skip_import_if_exists</code>	Define if the existing boundaries or properties will be updated or not.

2.17.3 pyaedt.generic.configurations.ImportResults

```
class pyaedt.generic.configurations.ImportResults
```

Contains the results of the import operations.

Each result can be True or False.

```
__init__()
```

Attributes

global_import_success	Returns True if all imports are successful.
-----------------------	---

```
from pyaedt import Hfss
app = Hfss(project_name="original_project", specified_version="2023.1",
           non_graphical=False, new_desktop_session=True,
           close_on_exit=True, student_version=False)

conf_file = self.aedtapp.configurations.export_config()

app2 = Hfss(projec_name='newproject')
app2.modeler.import_3d_cad(file_path)
out = app2.configurations.import_config(conf_file)
app2.configurations.results.global_import_success

...
```

2.18 Setup templates

This section lists all setup templates with their default values and keys.

You can edit a setup after it is created. Here is an example:

```
Launch AEDT 2023 R1 in non-graphical mode

from pyaedt import Hfss

hfss = Hfss()
# Any property of this setup can be found on this page.
setup = hfss.create_setup()
setup.props["AdaptMultipleFreqs"] = True
setup.update()
```

2.18.1 HFSS templates and arguments

This section lists all setup templates with their default values and keys available in HFSS.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Hfss

hfss = Hfss()
# Any property of this setup can be found on this page.
setup = hfss.create_setup()
setup.props["AdaptMultipleFreqs"] = True
setup.update()
```

HFSSDrivenAuto

```
OrderedDict([ ('.IsEnabled', True),
    ('MeshLink', OrderedDict([(('ImportMesh', False)]))),
    ('AutoSolverSetting', 'Balanced'),
    ('Sweeps',
        OrderedDict([ ('Sweep',
            OrderedDict([ ('RangeType',
                'LinearStep'),
                ('RangeStart',
                '1GHz'),
                ('RangeEnd', '10GHz'),
                ('RangeStep',
                '1GHz'))]))),
    ('SaveRadFieldsOnly', False),
    ('SaveAnyFields', True),
    ('Type', 'Discrete')])]
```

HFSSDrivenDefault

```
OrderedDict([ ('SolveType', 'Single'),
    ('MultipleAdaptiveFreqsSetup',
        OrderedDict([ ('1GHz', [0.02]),
                    ('2GHz', [0.02]),
                    ('5GHz', [0.02])])),
    ('Frequency', '5GHz'),
    ('MaxDeltaS', 0.02),
    ('PortsOnly', False),
    ('UseMatrixConv', False),
    ('MaximumPasses', 6),
    ('MinimumPasses', 1),
    ('MinimumConvergedPasses', 1),
    ('PercentRefinement', 30),
    ('IsEnabled', True),
    ('MeshLink', OrderedDict([(('ImportMesh', False)]))),
    ('BasisOrder', 1),
    ('DoLambdaRefine', True),
    ('DoMaterialLambda', True),
    ('SetLambdaTarget', False),
```

(continues on next page)

(continued from previous page)

```
('Target', 0.3333),
('UseMaxTetIncrease', False),
('PortAccuracy', 2),
('UseABCOnPort', False),
('SetPortMinMaxTri', False),
('UseDomains', False),
('UseIterativeSolver', False),
('SaveRadFieldsOnly', False),
('SaveAnyFields', True),
('IESolverType', 'Auto'),
('LambdaTargetForIESolver', 0.15),
('UseDefaultLambdaTgtForIESolver', True),
('IE Solver Accuracy', 'Balanced'))]
```

HFSSDrivenDefault

```
OrderedDict([
    ('SolveType', 'Single'),
    ('MultipleAdaptiveFreqsSetup',
     OrderedDict([
         ('1GHz', [0.02]),
         ('2GHz', [0.02]),
         ('5GHz', [0.02])])),
    ('Frequency', '5GHz'),
    ('MaxDeltaS', 0.02),
    ('PortsOnly', False),
    ('UseMatrixConv', False),
    ('MaximumPasses', 6),
    ('MinimumPasses', 1),
    ('MinimumConvergedPasses', 1),
    ('PercentRefinement', 30),
    ('IsEnabled', True),
    ('MeshLink', OrderedDict([('ImportMesh', False)])),
    ('BasisOrder', 1),
    ('DoLambdaRefine', True),
    ('DoMaterialLambda', True),
    ('SetLambdaTarget', False),
    ('Target', 0.3333),
    ('UseMaxTetIncrease', False),
    ('PortAccuracy', 2),
    ('UseABCOnPort', False),
    ('SetPortMinMaxTri', False),
    ('UseDomains', False),
    ('UseIterativeSolver', False),
    ('SaveRadFieldsOnly', False),
    ('SaveAnyFields', True),
    ('IESolverType', 'Auto'),
    ('LambdaTargetForIESolver', 0.15),
    ('UseDefaultLambdaTgtForIESolver', True),
    ('IE Solver Accuracy', 'Balanced'))])
```

HFSSTransient

```
OrderedDict([
    ('Frequency', '5GHz'),
    ('MaxDeltaS', 0.02),
    ('MaximumPasses', 20),
    ('UseImplicitSolver', True),
    ('IsEnabled', True),
    ('MeshLink', OrderedDict([('ImportMesh', False)])),
    ('BasisOrder', -1),
    ('Transient',
        OrderedDict([
            ('TimeProfile', 'Broadband Pulse'),
            ('HfssFrequency', '5GHz'),
            ('MinFreq', '100MHz'),
            ('MaxFreq', '1GHz'),
            ('NumFreqsExtracted', 401),
            ('SweepMinFreq', '100MHz'),
            ('SweepMaxFreq', '1GHz'),
            ('UseAutoTermination', 1),
            ('SteadyStateCriteria', 0.01),
            ('UseMinimumDuration', 0),
            ('TerminateOnMaximum', 0)])))]))
```

HFSSSBR

```
OrderedDict([
    ('.IsEnabled', True),
    ('MeshLink', OrderedDict([('ImportMesh', False)])),
    ('IsSbrRangeDoppler', False),
    ('RayDensityPerWavelength', 4),
    ('MaxNumberOfBounces', 5),
    ('RadiationSetup', ''),
    ('PTDUTDSimulationSettings', 'None'),
    ('Sweeps',
        OrderedDict([
            ('Sweep',
                OrderedDict([
                    ('RangeType', 'LinearStep'),
                    ('RangeStart', '1GHz'),
                    ('RangeEnd', '10GHz'),
                    ('RangeStep', '1GHz')])))]),
    ('ComputeFarFields', True)]))
```

2.18.2 HFSS 3D Layout and arguments

This section lists all setup templates with their default values and keys available in HFSS 3D Layout.

You can edit a setup after it is created. Here is an example:

```
Launch AEDT 2023 R1 in non-graphical mode

from pyaedt import Hfss

hfss = Hfss()
# Any property of this setup can be found on this page.
```

(continues on next page)

(continued from previous page)

```
setup = hfss.create_setup()
setup.props["AdaptMultipleFreqs"] = True
setup.update()
```

HFSS3DLayout

```
OrderedDict([ ('Properties', OrderedDict([('Enable', 'true')])),  
            ('CustomSetup', False),  
            ('SolveSetupType', 'HFSS'),  
            ('PercentRefinementPerPass', 30),  
            ('MinNumberOfPasses', 1),  
            ('MinNumberOfConvergedPasses', 1),  
            ('UseDefaultLambda', True),  
            ('UseMaxRefinement', False),  
            ('MaxRefinement', 1000000),  
            ('SaveAdaptiveCurrents', False),  
            ('SaveLastAdaptiveRadFields', False),  
            ('ProdMajVerID', -1),  
            ('ProjDesignSetup', ''),  
            ('ProdMinVerID', -1),  
            ('Refine', False),  
            ('Frequency', '10GHz'),  
            ('LambdaRefine', True),  
            ('MeshSizeFactor', 1.5),  
            ('QualityRefine', True),  
            ('MinAngle', '15deg'),  
            ('UniformityRefine', False),  
            ('MaxRatio', 2),  
            ('Smooth', False),  
            ('SmoothingPasses', 5),  
            ('UseEdgeMesh', False),  
            ('UseEdgeMeshAbsLength', False),  
            ('EdgeMeshRatio', 0.1),  
            ('EdgeMeshAbsLength', '1000mm'),  
            ('LayerProjectThickness', '0meter'),  
            ('UseDefeature', True),  
            ('UseDefeatureAbsLength', False),  
            ('DefeatureRatio', 1e-06),  
            ('DefeatureAbsLength', '0mm'),  
            ('InfArrayDimX', 0),  
            ('InfArrayDimY', 0),  
            ('InfArrayOrigX', 0),  
            ('InfArrayOrigY', 0),  
            ('InfArraySkew', 0),  
            ('ViaNumSides', 6),  
            ('ViaMaterial', 'copper'),  
            ('Style25DVia', 'Mesh'),  
            ('Replace3DTriangles', True),  
            ('LayerSnapTol', '1e-05'),  
            ('ViaDensity', 0),  
            ('HfssMesh', True),  
            ('Q3dPostProc', False),
```

(continues on next page)

(continued from previous page)

```
('UnitFactor', 1000),
('Verbose', False),
('NumberOfProcessors', 0),
('SmallVoidArea', -2e-09),
('HealingOption', 1),
('InclBBoxOption', 1),
('AuxBlock', OrderedDict()),
('DoAdaptive', True),
('Color', ['R', 0, 'G', 0, 'B', 0]),
( 'AdvancedSettings',
    OrderedDict([
        ('AccuracyLevel', 2),
        ('GapPortCalibration', True),
        ('ReferenceLengthRatio', 0.25),
        ('RefineAreaRatio', 4),
        ('DRCon', False),
        ('FastSolverOn', False),
        ('StartFastSolverAt', 3000),
        ('LoopTreeOn', True),
        ('SingularElementsOn', False),
        ('UseStaticPortSolver', False),
        ('UseThinMetalPortSolver', False),
        ('ComputeBothEvenAndOddCPWModes', False),
        ('ZeroMetalLayerThickness', 0),
        ('ThinDielectric', 0),
        ('UseShellElements', False),
        ('SVDHighCompression', False),
        ('NumProcessors', 1),
        ('SolverType', 'Direct Solver'),
        ('UseHfssIterativeSolver', False),
        ('UseHfssMUMPSSolver', True),
        ('RelativeResidual', 1e-06),
        ('EnhancedLowFreqAccuracy', False),
        ('OrderBasis', -1),
        ('MaxDeltaZo', 2),
        ('UseRadBoundaryOnPorts', False),
        ('SetTrianglesForWavePort', False),
        ('MinTrianglesForWavePort', 100),
        ('MaxTrianglesForWavePort', 500),
        ('numprocessorsdistrib', 1),
        ('CausalMaterials', True),
        ('enabledsoforopti', True),
        ('usehfsssolvelicense', False),
        ('ExportAfterSolve', False),
        ('ExportDir', ''),
        ('CircuitSparamDefinition', False),
        ('CircuitIntegrationType', 'FFT'),
        ('DesignType', 'generic'),
        ('MeshingMethod', 'Phi'),
        ('EnableDesignIntersectionCheck', True),
        ('UseAlternativeMeshMethodsAsFallBack',
```

(continues on next page)

(continued from previous page)

```

        True),
        ('ModeOption', 'General mode'),
        ('BroadbandFreqOption',
         'AutoMaxFreq'),
        ('BroadbandMaxNumFreq', 5),
        ('SaveADP', True),
        ('UseAdvancedDCExtrap', False),
        ('PhiMesherDeltaZRatio', 100000)])),
    ('CurveApproximation',
     OrderedDict([ ('ArcAngle', '30deg'),
                  ('StartAzimuth', '0deg'),
                  ('UseError', False),
                  ('Error', '0meter'),
                  ('MaxPoints', 8),
                  ('UnionPolys', True),
                  ('Replace3DTriangles', True])),
    ('Q3D_DCSettings',
     OrderedDict([ ('SolveResOnly', True),
                  ('Cond',
                   OrderedDict([ ('MaxPass', 10),
                               ('MinPass', 1),
                               ('MinConvPass', 1),
                               ('PerError', 1),
                               ('PerRefine', 30)])),
                  ('Mult',
                   OrderedDict([ ('MaxPass', 1),
                               ('MinPass', 1),
                               ('MinConvPass', 1),
                               ('PerError', 1),
                               ('PerRefine', 30)])),
                  ('Solution Order', 'Normal')])),
    ('AdaptiveSettings',
     OrderedDict([ ('DoAdaptive', True),
                  ('SaveFields', False),
                  ('SaveRadFieldsOnly', False),
                  ('MaxRefinePerPass', 30),
                  ('MinPasses', 1),
                  ('MinConvergedPasses', 1),
                  ('AdaptType', 'kSingle'),
                  ('Basic', True),
                  ('SingleFrequencyDataList',
                   OrderedDict([ ('AdaptiveFrequencyData',
                                 OrderedDict([ ('AdaptiveFrequency',
                                              '5GHz'),
                                              ('MaxDelta',
                                               '0.02'),
                                              ('MaxPasses',
                                               10),
                                              ('Expressions',
                                               [ ])])]))),
                  ('BroadbandFrequencyDataList',
                   OrderedDict([ ('AdaptiveFrequencyData',
                                 ...
```
```

(continues on next page)

(continued from previous page)

### 2.18.3 Maxwell templates and arguments

This section lists all setup templates with their default values and keys available in Maxwell 2D and 3D.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Maxwell3d

Maxwell3d = Maxwell3d ()
Any property of this setup can be found on this page.
setup = Maxwell3d.create_setup ()
setup.props["MaximumPasses"] = 5
setup.update ()
```

#### MaxwellTransient

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False)])),
 ('NonlinearSolverResidual', '0.005'),
 ('ScalarPotential', 'Second Order'),
 ('SmoothBHCurve', False),
 ('StopTime', '10000000ns'),
 ('TimeStep', '2000000ns'),
 ('OutputError', False),
 ('UseControlProgram', False),
 ('ControlProgramName', ''),
 ('ControlProgramArg', ''),
 ('CallCtrlProgAfterLastStep', False),
 ('FastReachSteadyState', False),
 ('AutoDetectSteadyState', False),
 ('IsGeneralTransient', True),
 ('IsHalfPeriodicTransient', False),
 ('SaveFieldsType', 'None'),
 ('CacheSaveKind', 'Count'),
 ('NumberSolveSteps', 1),
 ('RangeStart', '0s'),
 ('RangeEnd', '0.1s')])
```

#### Magnetostatic

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False)])),
 ('MaximumPasses', 10),
 ('MinimumPasses', 2),
 ('MinimumConvergedPasses', 1),
 ('PercentRefinement', 30),
 ('SolveFieldOnly', False),
 ('PercentError', 1),
 ('SolveMatrixAtLast', True),
 ('UseIterativeSolver', False),
 ('RelativeResidual', 1e-06),
 ('NonLinearResidual', 0.001),
 ('SmoothBHCurve', False),
 ('MuOption', OrderedDict([('MuNonLinearBH', True)]))])
```

**Electrostatic**

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([(('ImportMesh', False)])),
 ('MaximumPasses', 10),
 ('MinimumPasses', 2),
 ('MinimumConvergedPasses', 1),
 ('PercentRefinement', 30),
 ('SolveFieldOnly', False),
 ('PercentError', 1),
 ('SolveMatrixAtLast', True),
 ('UseIterativeSolver', False),
 ('RelativeResidual', 1e-06),
 ('NonLinearResidual', 0.001)])
```

**EddyCurrent**

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([(('ImportMesh', False)])),
 ('MaximumPasses', 6),
 ('MinimumPasses', 1),
 ('MinimumConvergedPasses', 1),
 ('PercentRefinement', 30),
 ('SolveFieldOnly', False),
 ('PercentError', 1),
 ('SolveMatrixAtLast', True),
 ('UseIterativeSolver', False),
 ('RelativeResidual', 1e-05),
 ('NonLinearResidual', 0.0001),
 ('SmoothBHCurve', False),
 ('Frequency', '60Hz'),
 ('HasSweepSetup', False),
 ('SweepRanges',
 OrderedDict([
 ('Subrange',
 OrderedDict([
 ('SweepSetupType',
 OrderedDict([
 ('LinearStep',),
 ('StartValue',
 '1e-08GHz'),
 ('StopValue',
 '1e-06GHz'),
 ('StepSize',
 '1e-08GHz')]))]))),
 ('UseHighOrderShapeFunc', False),
 ('UseMuLink', False)])])
```

**ElectricTransient**

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([(('ImportMesh', False)])),
 ('Tolerance', 0.005),
 ('ComputePowerLoss', False),
 ('Data',
```

(continues on next page)

(continued from previous page)

```
OrderedDict([('SaveField', True),
 ('Stop', '100s'),
 ('InitialStep', '0.01s'),
 ('MaxStep', '5s'))),
 ('Initial Voltage', '0mV')])
```

## 2.18.4 Q3D templates and arguments

This section lists all setup templates with their default values and keys available in Q3D and 2D Extractor. Note that to use nested parameters, you can set a parameter using the `__` separator as shown in the following example.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Q3d

app = Q3d()
Any property of this setup can be found on this page.
setup = app.create_setup(AC__MaxPasses=6)
```

### Matrix

```
OrderedDict([('AdaptiveFreq', '1GHz'),
 ('SaveFields', False),
 ('Enabled', True),
 ('Cap',
 OrderedDict([('MaxPass', 10),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine', 30),
 ('AutoIncreaseSolutionOrder', True),
 ('SolutionOrder', 'High'),
 ('Solver Type', 'Iterative')]))),
 ('DC',
 OrderedDict([('SolveResOnly', False),
 ('Cond',
 OrderedDict([('MaxPass', 10),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine', 30)]))],
 ('Mult',
 OrderedDict([('MaxPass', 1),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine',
 30)]))))),
 ('AC',
 OrderedDict([('MaxPass', 10),
 ('MinPass', 1),
```

(continues on next page)

(continued from previous page)

```
('MinConvPass', 1),
('PerError', 1),
('PerRefine', 30))))])
```

**Close**

```
OrderedDict([('AdaptiveFreq', '1GHz'),
 ('SaveFields', True),
 ('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False)])),
 ('CGDataBlock',
 OrderedDict([
 ('MaxPass', 10),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine', 30),
 ('DataType', 'CG'),
 ('Included', True),
 ('UseParamConv', True),
 ('UseLossyParamConv', False),
 ('PerErrorParamConv', 1),
 ('UseLossConv', True)])),
 ('RLDataBlock',
 OrderedDict([
 ('MaxPass', 10),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine', 30),
 ('DataType', 'CG'),
 ('Included', True),
 ('UseParamConv', True),
 ('UseLossyParamConv', False),
 ('PerErrorParamConv', 1),
 ('UseLossConv', True)])),
 ('CacheSaveKind', 'Delta'),
 ('ConstantDelta', '0s')])]
```

**Open**

```
OrderedDict([('AdaptiveFreq', '1GHz'),
 ('SaveFields', True),
 ('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False)])),
 ('CGDataBlock',
 OrderedDict([
 ('MaxPass', 10),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine', 30),
 ('DataType', 'CG'),
 ('Included', True),
```

(continues on next page)

(continued from previous page)

```

 ('UseParamConv', True),
 ('UseLossyParamConv', False),
 ('PerErrorParamConv', 1),
 ('UseLossConv', True])),
 ('RLDataBlock',
 OrderedDict([('MaxPass', 10),
 ('MinPass', 1),
 ('MinConvPass', 1),
 ('PerError', 1),
 ('PerRefine', 30),
 ('DataType', 'CG'),
 ('Included', True),
 ('UseParamConv', True),
 ('UseLossyParamConv', False),
 ('PerErrorParamConv', 1),
 ('UseLossConv', True])),
 ('CacheSaveKind', 'Delta'),
 ('ConstantDelta', '0s')])
)

```

## 2.18.5 Icepak templates and arguments

This section lists all setup templates with their default values and keys available in Icepak. Note that Icepak parameters contain spaces. To use them as arguments of the "create\_setup" method, these same parameters have to be used without spaces. You can edit a setup after it is created. Here is an example:

```

from pyaedt import Icepak

app = Icepak()
Any property of this setup can be found on this page.
setup = app.create_setup(MaxIterations=5)

```

Available turbulent models are: "ZeroEquation", "TwoEquation", "EnhancedTwoEquation", "RNG", "EnhancedRNG", "RealizableTwoEquation", "EnhancedRealizableTwoEquation", "SpalartAllmaras", "kOmegaSST". **TransientFlowOnly**

```

OrderedDict([('Enabled', True),
 ('Flow Regime', 'Laminar'),
 ('Turbulent Model Eqn', 'ZeroEquation'),
 ('Include Temperature', False),
 ('Include Flow', True),
 ('Include Gravity', False),
 ('Include Solar', False),
 ('Solution Initialization - X Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Y Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Z Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Temperature',
 'AmbientTemp'),
 ('Solution Initialization - Turbulent Kinetic '

```

(continues on next page)

(continued from previous page)

```

'Energy',
'1m2_per_s2'),
('Solution Initialization - Turbulent Dissipation',
'Rate',
'1m2_per_s3'),
('Solution Initialization - Specific Dissipation',
'Rate',
'1diss_per_s'),
('Solution Initialization - Use Model Based Flow',
'Initialization',
False),
('Convergence Criteria - Flow', '0.001'),
('Convergence Criteria - Energy', '1e-07'),
('Convergence Criteria - Turbulent Kinetic Energy',
'0.001'),
('Convergence Criteria - Turbulent Dissipation Rate',
'0.001'),
('Convergence Criteria - Specific Dissipation Rate',
'0.001'),
('Convergence Criteria - Discrete Ordinates',
'1e-06'),
('IsEnabled', False),
('Radiation Model', 'Off'),
('Solar Radiation Model',
'Solar Radiation Calculator'),
('Solar Radiation - Scattering Fraction', '0'),
('Solar Radiation - Day', 1),
('Solar Radiation - Month', 1),
('Solar Radiation - Hours', 0),
('Solar Radiation - Minutes', 0),
('Solar Radiation - GMT', '0'),
('Solar Radiation - Latitude', '0'),
('Solar Radiation - Latitude Direction', 'East'),
('Solar Radiation - Longitude', '0'),
('Solar Radiation - Longitude Direction', 'North'),
('Solar Radiation - Ground Reflectance', '0'),
('Solar Radiation - Sunshine Fraction', '0'),
('Solar Radiation - North X', '0'),
('Solar Radiation - North Y', '0'),
('Solar Radiation - North Z', '1'),
('Under-relaxation - Pressure', '0.3'),
('Under-relaxation - Momentum', '0.7'),
('Under-relaxation - Temperature', '1'),
('Under-relaxation - Turbulent Kinetic Energy',
'0.8'),
('Under-relaxation - Turbulent Dissipation Rate',
'0.8'),
('Under-relaxation - Specific Dissipation Rate',
'0.8'),
('Discretization Scheme - Pressure', 'Standard'),
('Discretization Scheme - Momentum', 'First'),
('Discretization Scheme - Temperature', 'First'),

```

(continues on next page)

(continued from previous page)

```
('Secondary Gradient', False),
('Discretization Scheme - Turbulent Kinetic Energy',
 'First'),
('Discretization Scheme - Turbulent Dissipation '
 'Rate',
 'First'),
('Discretization Scheme - Specific Dissipation Rate',
 'First'),
('Discretization Scheme - Discrete Ordinates',
 'First'),
('Linear Solver Type - Pressure', 'V'),
('Linear Solver Type - Momentum', 'flex'),
('Linear Solver Type - Temperature', 'F'),
('Linear Solver Type - Turbulent Kinetic Energy',
 'flex'),
('Linear Solver Type - Turbulent Dissipation Rate',
 'flex'),
('Linear Solver Type - Specific Dissipation Rate',
 'flex'),
('Linear Solver Termination Criterion - Pressure',
 '0.1'),
('Linear Solver Termination Criterion - Momentum',
 '0.1'),
('Linear Solver Termination Criterion - Temperature',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Kinetic Energy',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Dissipation Rate',
 '0.1'),
('Linear Solver Termination Criterion - Specific '
 'Dissipation Rate',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Pressure',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Momentum',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Temperature',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Kinetic Energy',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Dissipation Rate',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Specific Dissipation Rate',
 '0.1'),
```

(continues on next page)

(continued from previous page)

```
('Linear Solver Stabilization - Pressure', 'None'),
('Linear Solver Stabilization - Temperature', 'None'),
('Coupled pressure-velocity formulation', False),
('Frozen Flow Simulation', False),
('Start Time', '0s'),
('Stop Time', '20s'),
('Time Step', '1s'),
('Iterations per Time Step', 20),
('Import Start Time', False),
('Copy Fields From Source', False),
('SaveFieldsType', 'Every N Steps'),
('N Steps', '10'),
('Enable Control Program', False),
('Control Program Name', '')])
```

**TransientTemperatureOnly**

```
OrderedDict([
 ('Enabled', True),
 ('Flow Regime', 'Laminar'),
 ('Turbulent Model Eqn', 'ZeroEquation'),
 ('Include Temperature', True),
 ('Include Flow', False),
 ('Include Gravity', False),
 ('Include Solar', False),
 ('Solution Initialization - X Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Y Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Z Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Temperature',
 'AmbientTemp'),
 ('Solution Initialization - Turbulent Kinetic Energy',
 '1m2_per_s2'),
 ('Solution Initialization - Turbulent Dissipation Rate',
 '1m2_per_s3'),
 ('Solution Initialization - Specific Dissipation Rate',
 '1diss_per_s'),
 ('Solution Initialization - Use Model Based Flow Initialization',
 False),
 ('Convergence Criteria - Flow', '0.001'),
 ('Convergence Criteria - Energy', '1e-07'),
 ('Convergence Criteria - Turbulent Kinetic Energy',
 '0.001'),
 ('Convergence Criteria - Turbulent Dissipation Rate',
 '0.001'),
 ('Convergence Criteria - Specific Dissipation Rate',
 '0.001'),
```

(continues on next page)

(continued from previous page)

```
('Convergence Criteria - Discrete Ordinates',
 '1e-06'),
('IsEnabled', False),
('Radiation Model', 'Off'),
('Solar Radiation Model',
 'Solar Radiation Calculator'),
('Solar Radiation - Scattering Fraction', '0'),
('Solar Radiation - Day', 1),
('Solar Radiation - Month', 1),
('Solar Radiation - Hours', 0),
('Solar Radiation - Minutes', 0),
('Solar Radiation - GMT', '0'),
('Solar Radiation - Latitude', '0'),
('Solar Radiation - Latitude Direction', 'East'),
('Solar Radiation - Longitude', '0'),
('Solar Radiation - Longitude Direction', 'North'),
('Solar Radiation - Ground Reflectance', '0'),
('Solar Radiation - Sunshine Fraction', '0'),
('Solar Radiation - North X', '0'),
('Solar Radiation - North Y', '0'),
('Solar Radiation - North Z', '1'),
('Under-relaxation - Pressure', '0.3'),
('Under-relaxation - Momentum', '0.7'),
('Under-relaxation - Temperature', '1'),
('Under-relaxation - Turbulent Kinetic Energy',
 '0.8'),
('Under-relaxation - Turbulent Dissipation Rate',
 '0.8'),
('Under-relaxation - Specific Dissipation Rate',
 '0.8'),
('Discretization Scheme - Pressure', 'Standard'),
('Discretization Scheme - Momentum', 'First'),
('Discretization Scheme - Temperature', 'First'),
('Secondary Gradient', False),
('Discretization Scheme - Turbulent Kinetic Energy',
 'First'),
('Discretization Scheme - Turbulent Dissipation '
 'Rate',
 'First'),
('Discretization Scheme - Specific Dissipation Rate',
 'First'),
('Discretization Scheme - Discrete Ordinates',
 'First'),
('Linear Solver Type - Pressure', 'V'),
('Linear Solver Type - Momentum', 'flex'),
('Linear Solver Type - Temperature', 'F'),
('Linear Solver Type - Turbulent Kinetic Energy',
 'flex'),
('Linear Solver Type - Turbulent Dissipation Rate',
 'flex'),
('Linear Solver Type - Specific Dissipation Rate',
 'flex'),
```

(continues on next page)

(continued from previous page)

```

 ('Linear Solver Termination Criterion - Pressure',
 '0.1'),
 ('Linear Solver Termination Criterion - Momentum',
 '0.1'),
 ('Linear Solver Termination Criterion - Temperature',
 '0.1'),
 ('Linear Solver Termination Criterion - Turbulent Kinetic Energy',
 '0.1'),
 ('Linear Solver Termination Criterion - Turbulent Dissipation Rate',
 '0.1'),
 ('Linear Solver Termination Criterion - Specific Dissipation Rate',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - Pressure',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - Momentum',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - Temperature',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - Turbulent Kinetic Energy',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - Turbulent Dissipation Rate',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - Specific Dissipation Rate',
 '0.1'),
 ('Linear Solver Stabilization - Pressure', 'None'),
 ('Linear Solver Stabilization - Temperature', 'None'),
 ('Coupled pressure-velocity formulation', False),
 ('Frozen Flow Simulation', False),
 ('Start Time', '0s'),
 ('Stop Time', '20s'),
 ('Time Step', '1s'),
 ('Iterations per Time Step', 20),
 ('Import Start Time', False),
 ('Copy Fields From Source', False),
 ('SaveFieldsType', 'Every N Steps'),
 ('N Steps', '10'),
 ('Enable Control Program', False),
 ('Control Program Name', '')])

```

## TransientTemperatureAndFlow

```
OrderedDict([('Enabled', True),
 ('Flow Regime', 'Laminar')],
```

(continues on next page)

(continued from previous page)

```
('Turbulent Model Eqn', 'ZeroEquation'),
('Include Temperature', True),
('Include Flow', True),
('Include Gravity', False),
('Include Solar', False),
('Solution Initialization - X Velocity',
 '0m_per_sec'),
('Solution Initialization - Y Velocity',
 '0m_per_sec'),
('Solution Initialization - Z Velocity',
 '0m_per_sec'),
('Solution Initialization - Temperature',
 'AmbientTemp'),
('Solution Initialization - Turbulent Kinetic Energy',
 '1m2_per_s2'),
('Solution Initialization - Turbulent Dissipation Rate',
 '1m2_per_s3'),
('Solution Initialization - Specific Dissipation Rate',
 '1diss_per_s'),
('Solution Initialization - Use Model Based Flow Initialization',
 False),
('Convergence Criteria - Flow', '0.001'),
('Convergence Criteria - Energy', '1e-07'),
('Convergence Criteria - Turbulent Kinetic Energy',
 '0.001'),
('Convergence Criteria - Turbulent Dissipation Rate',
 '0.001'),
('Convergence Criteria - Specific Dissipation Rate',
 '0.001'),
('Convergence Criteria - Discrete Ordinates',
 '1e-06'),
('.IsEnabled', False),
('Radiation Model', 'Off'),
('Solar Radiation Model',
 'Solar Radiation Calculator'),
('Solar Radiation - Scattering Fraction', '0'),
('Solar Radiation - Day', 1),
('Solar Radiation - Month', 1),
('Solar Radiation - Hours', 0),
('Solar Radiation - Minutes', 0),
('Solar Radiation - GMT', '0'),
('Solar Radiation - Latitude', '0'),
('Solar Radiation - Latitude Direction', 'East'),
('Solar Radiation - Longitude', '0'),
('Solar Radiation - Longitude Direction', 'North'),
('Solar Radiation - Ground Reflectance', '0'),
('Solar Radiation - Sunshine Fraction', '0'),
('Solar Radiation - North X', '0'),
```

(continues on next page)

(continued from previous page)

```

('Solar Radiation - North Y', '0'),
('Solar Radiation - North Z', '1'),
('Under-relaxation - Pressure', '0.3'),
('Under-relaxation - Momentum', '0.7'),
('Under-relaxation - Temperature', '1'),
('Under-relaxation - Turbulent Kinetic Energy',
 '0.8'),
('Under-relaxation - Turbulent Dissipation Rate',
 '0.8'),
('Under-relaxation - Specific Dissipation Rate',
 '0.8'),
('Discretization Scheme - Pressure', 'Standard'),
('Discretization Scheme - Momentum', 'First'),
('Discretization Scheme - Temperature', 'First'),
('Secondary Gradient', False),
('Discretization Scheme - Turbulent Kinetic Energy',
 'First'),
('Discretization Scheme - Turbulent Dissipation '
 'Rate',
 'First'),
('Discretization Scheme - Specific Dissipation Rate',
 'First'),
('Discretization Scheme - Discrete Ordinates',
 'First'),
('Linear Solver Type - Pressure', 'V'),
('Linear Solver Type - Momentum', 'flex'),
('Linear Solver Type - Temperature', 'F'),
('Linear Solver Type - Turbulent Kinetic Energy',
 'flex'),
('Linear Solver Type - Turbulent Dissipation Rate',
 'flex'),
('Linear Solver Type - Specific Dissipation Rate',
 'flex'),
('Linear Solver Termination Criterion - Pressure',
 '0.1'),
('Linear Solver Termination Criterion - Momentum',
 '0.1'),
('Linear Solver Termination Criterion - Temperature',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Kinetic Energy',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Dissipation Rate',
 '0.1'),
('Linear Solver Termination Criterion - Specific '
 'Dissipation Rate',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Pressure',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '

```

(continues on next page)

(continued from previous page)

```

'Momentum',
'0.1'),
('Linear Solver Residual Reduction Tolerance - '
'Temperature',
'0.1'),
('Linear Solver Residual Reduction Tolerance - '
'Turbulent Kinetic Energy',
'0.1'),
('Linear Solver Residual Reduction Tolerance - '
'Turbulent Dissipation Rate',
'0.1'),
('Linear Solver Residual Reduction Tolerance - '
'Specific Dissipation Rate',
'0.1'),
('Linear Solver Stabilization - Pressure', 'None'),
('Linear Solver Stabilization - Temperature', 'None'),
('Coupled pressure-velocity formulation', False),
('Frozen Flow Simulation', False),
('Start Time', '0s'),
('Stop Time', '20s'),
('Time Step', '1s'),
('Iterations per Time Step', 20),
('Import Start Time', False),
('Copy Fields From Source', False),
('SaveFieldsType', 'Every N Steps'),
('N Steps', '10'),
('Enable Control Program', False),
('Control Program Name', '')])

```

**SteadyFlowOnly**

```

OrderedDict([
('Enabled', True),
('Flow Regime', 'Laminar'),
('Turbulent Model Eqn', 'ZeroEquation'),
('Include Flow', True),
('Include Gravity', False),
('Solution Initialization - X Velocity',
'0m_per_sec'),
('Solution Initialization - Y Velocity',
'0m_per_sec'),
('Solution Initialization - Z Velocity',
'0m_per_sec'),
('Solution Initialization - Temperature',
'AmbientTemp'),
('Solution Initialization - Turbulent Kinetic '
'Energy',
'1m2_per_s2'),
('Solution Initialization - Turbulent Dissipation '
'Rate',
'1m2_per_s3'),
('Solution Initialization - Specific Dissipation '
'Rate',
'1m2_per_s3')])

```

(continues on next page)

(continued from previous page)

```

 '1diss_per_s'),
('Convergence Criteria - Flow', '0.001'),
('Convergence Criteria - Energy', '1e-07'),
('Convergence Criteria - Turbulent Kinetic Energy',
 '0.001'),
('Convergence Criteria - Turbulent Dissipation Rate',
 '0.001'),
('Convergence Criteria - Specific Dissipation Rate',
 '0.001'),
('Convergence Criteria - Discrete Ordinates',
 '1e-06'),
('.IsEnabled', False),
('Radiation Model', 'Off'),
('Under-relaxation - Pressure', '0.7'),
('Under-relaxation - Momentum', '0.3'),
('Under-relaxation - Temperature', '1'),
('Under-relaxation - Turbulent Kinetic Energy',
 '0.8'),
('Under-relaxation - Turbulent Dissipation Rate',
 '0.8'),
('Under-relaxation - Specific Dissipation Rate',
 '0.8'),
('Discretization Scheme - Pressure', 'Standard'),
('Discretization Scheme - Momentum', 'First'),
('Discretization Scheme - Temperature', 'First'),
('Secondary Gradient', False),
('Discretization Scheme - Turbulent Kinetic Energy',
 'First'),
('Discretization Scheme - Turbulent Dissipation '
 'Rate',
 'First'),
('Discretization Scheme - Specific Dissipation Rate',
 'First'),
('Discretization Scheme - Discrete Ordinates',
 'First'),
('Linear Solver Type - Pressure', 'V'),
('Linear Solver Type - Momentum', 'flex'),
('Linear Solver Type - Temperature', 'F'),
('Linear Solver Type - Turbulent Kinetic Energy',
 'flex'),
('Linear Solver Type - Turbulent Dissipation Rate',
 'flex'),
('Linear Solver Type - Specific Dissipation Rate',
 'flex'),
('Linear Solver Termination Criterion - Pressure',
 '0.1'),
('Linear Solver Termination Criterion - Momentum',
 '0.1'),
('Linear Solver Termination Criterion - Temperature',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Kinetic Energy',

```

(continues on next page)

(continued from previous page)

```

 '0.1'),
 ('Linear Solver Termination Criterion - Turbulent '
 'Dissipation Rate',
 '0.1'),
 ('Linear Solver Termination Criterion - Specific '
 'Dissipation Rate',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Pressure',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Momentum',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Temperature',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Kinetic Energy',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Dissipation Rate',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Specific Dissipation Rate',
 '0.1'),
 ('Linear Solver Stabilization - Pressure', 'None'),
 ('Linear Solver Stabilization - Temperature', 'None'),
 ('Frozen Flow Simulation', False),
 ('Sequential Solve of Flow and Energy Equations',
 False),
 ('Convergence Criteria - Max Iterations', 100)])

```

**SteadyTemperatureOnly**

```

OrderedDict([
 ('Enabled', True),
 ('Flow Regime', 'Laminar'),
 ('Turbulent Model Eqn', 'ZeroEquation'),
 ('Include Temperature', True),
 ('Include Gravity', False),
 ('Solution Initialization - X Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Y Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Z Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Temperature',
 'AmbientTemp'),
 ('Solution Initialization - Turbulent Kinetic '
 'Energy',
 '1m2_per_s2'),
 ('Solution Initialization - Turbulent Dissipation '
 'Rate',
 '0.1')
])

```

(continues on next page)

(continued from previous page)

```

 '1m2_per_s3'),
 ('Solution Initialization - Specific Dissipation',
 'Rate',
 '1diss_per_s'),
 ('Convergence Criteria - Flow', '0.001'),
 ('Convergence Criteria - Energy', '1e-07'),
 ('Convergence Criteria - Turbulent Kinetic Energy',
 '0.001'),
 ('Convergence Criteria - Turbulent Dissipation Rate',
 '0.001'),
 ('Convergence Criteria - Specific Dissipation Rate',
 '0.001'),
 ('Convergence Criteria - Discrete Ordinates',
 '1e-06'),
 ('IsEnabled', False),
 ('Radiation Model', 'Off'),
 ('Under-relaxation - Pressure', '0.7'),
 ('Under-relaxation - Momentum', '0.3'),
 ('Under-relaxation - Temperature', '1'),
 ('Under-relaxation - Turbulent Kinetic Energy',
 '0.8'),
 ('Under-relaxation - Turbulent Dissipation Rate',
 '0.8'),
 ('Under-relaxation - Specific Dissipation Rate',
 '0.8'),
 ('Discretization Scheme - Pressure', 'Standard'),
 ('Discretization Scheme - Momentum', 'First'),
 ('Discretization Scheme - Temperature', 'Second'),
 ('Secondary Gradient', False),
 ('Discretization Scheme - Turbulent Kinetic Energy',
 'First'),
 ('Discretization Scheme - Turbulent Dissipation',
 'Rate',
 'First'),
 ('Discretization Scheme - Specific Dissipation Rate',
 'First'),
 ('Discretization Scheme - Discrete Ordinates',
 'First'),
 ('Linear Solver Type - Pressure', 'V'),
 ('Linear Solver Type - Momentum', 'flex'),
 ('Linear Solver Type - Temperature', 'F'),
 ('Linear Solver Type - Turbulent Kinetic Energy',
 'flex'),
 ('Linear Solver Type - Turbulent Dissipation Rate',
 'flex'),
 ('Linear Solver Type - Specific Dissipation Rate',
 'flex'),
 ('Linear Solver Termination Criterion - Pressure',
 '0.1'),
 ('Linear Solver Termination Criterion - Momentum',
 '0.1'),
 ('Linear Solver Termination Criterion - Temperature',
 '0.1')
)

```

(continues on next page)

(continued from previous page)

```

 '0.1'),
 ('Linear Solver Termination Criterion - Turbulent '
 'Kinetic Energy',
 '0.1'),
 ('Linear Solver Termination Criterion - Turbulent '
 'Dissipation Rate',
 '0.1'),
 ('Linear Solver Termination Criterion - Specific '
 'Dissipation Rate',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Pressure',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Momentum',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Temperature',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Kinetic Energy',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Dissipation Rate',
 '0.1'),
 ('Linear Solver Residual Reduction Tolerance - '
 'Specific Dissipation Rate',
 '0.1'),
 ('Linear Solver Stabilization - Pressure', 'None'),
 ('Linear Solver Stabilization - Temperature', 'None'),
 ('Sequential Solve of Flow and Energy Equations',
 False),
 ('Convergence Criteria - Max Iterations', 100)))

```

**SteadyTemperatureAndFlow**

```

OrderedDict([
 ('Enabled', True),
 ('Flow Regime', 'Laminar'),
 ('Turbulent Model Eqn', 'ZeroEquation'),
 ('Include Temperature', True),
 ('Include Flow', True),
 ('Include Gravity', False),
 ('Solution Initialization - X Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Y Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Z Velocity',
 '0m_per_sec'),
 ('Solution Initialization - Temperature',
 'AmbientTemp'),
 ('Solution Initialization - Turbulent Kinetic '
 'Energy'),
])

```

(continues on next page)

(continued from previous page)

```

 '1m2_per_s2'),
 ('Solution Initialization - Turbulent Dissipation '
 'Rate',
 '1m2_per_s3'),
 ('Solution Initialization - Specific Dissipation '
 'Rate',
 '1diss_per_s'),
 ('Convergence Criteria - Flow', '0.001'),
 ('Convergence Criteria - Energy', '1e-07'),
 ('Convergence Criteria - Turbulent Kinetic Energy',
 '0.001'),
 ('Convergence Criteria - Turbulent Dissipation Rate',
 '0.001'),
 ('Convergence Criteria - Specific Dissipation Rate',
 '0.001'),
 ('Convergence Criteria - Discrete Ordinates',
 '1e-06'),
 ('IsEnabled', False),
 ('Radiation Model', 'Off'),
 ('Under-relaxation - Pressure', '0.7'),
 ('Under-relaxation - Momentum', '0.3'),
 ('Under-relaxation - Temperature', '1'),
 ('Under-relaxation - Turbulent Kinetic Energy',
 '0.8'),
 ('Under-relaxation - Turbulent Dissipation Rate',
 '0.8'),
 ('Under-relaxation - Specific Dissipation Rate',
 '0.8'),
 ('Discretization Scheme - Pressure', 'Standard'),
 ('Discretization Scheme - Momentum', 'First'),
 ('Discretization Scheme - Temperature', 'Second'),
 ('Secondary Gradient', False),
 ('Discretization Scheme - Turbulent Kinetic Energy',
 'First'),
 ('Discretization Scheme - Turbulent Dissipation '
 'Rate',
 'First'),
 ('Discretization Scheme - Specific Dissipation Rate',
 'First'),
 ('Discretization Scheme - Discrete Ordinates',
 'First'),
 ('Linear Solver Type - Pressure', 'V'),
 ('Linear Solver Type - Momentum', 'flex'),
 ('Linear Solver Type - Temperature', 'F'),
 ('Linear Solver Type - Turbulent Kinetic Energy',
 'flex'),
 ('Linear Solver Type - Turbulent Dissipation Rate',
 'flex'),
 ('Linear Solver Type - Specific Dissipation Rate',
 'flex'),
 ('Linear Solver Termination Criterion - Pressure',
 '0.1'),

```

(continues on next page)

(continued from previous page)

```
('Linear Solver Termination Criterion - Momentum',
 '0.1'),
('Linear Solver Termination Criterion - Temperature',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Kinetic Energy',
 '0.1'),
('Linear Solver Termination Criterion - Turbulent '
 'Dissipation Rate',
 '0.1'),
('Linear Solver Termination Criterion - Specific '
 'Dissipation Rate',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Pressure',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Momentum',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Temperature',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Kinetic Energy',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Turbulent Dissipation Rate',
 '0.1'),
('Linear Solver Residual Reduction Tolerance - '
 'Specific Dissipation Rate',
 '0.1'),
('Linear Solver Stabilization - Pressure', 'None'),
('Linear Solver Stabilization - Temperature', 'None'),
('Frozen Flow Simulation', False),
('Sequential Solve of Flow and Energy Equations',
 False),
('Convergence Criteria - Max Iterations', 100]))
```

## 2.18.6 Mechanical templates and arguments

This section lists all setup templates with their default values and keys available in Mechanical.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Mechanical

app = Mechanical()
Any property of this setup can be found on this page.
setup = app.create_setup(MaxModes=6)
```

MechTerm

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False])),
 ('Solver', 'Program Controlled'),
 ('Stepping', 'Program Controlled')])
```

**MechModal**

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False])),
 ('Max Modes', 6),
 ('Limit Search', True),
 ('Range Max', '100MHz'),
 ('Range Min', '0Hz'),
 ('Solver', 'Program Controlled')])
```

**MechStructural**

```
OrderedDict([('Enabled', True),
 ('MeshLink', OrderedDict([('ImportMesh', False])),
 ('Solver', 'Program Controlled'),
 ('Stepping', 'Program Controlled')])
```

**2.18.7 Circuit templates and arguments**

This section lists all setup templates with their default values and keys available in Circuit.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Hfss

hfss = Hfss()
Any property of this setup can be found on this page.
setup = hfss.create_setup()
setup.props["AdaptMultipleFreqs"] = True
setup.update()
```

**NexximLNA**

```
OrderedDict([('DataBlockID', 16),
 ('OptionName', 'Default Options'),
 ('AdditionalOptions', ''),
 ('AlterBlockName', ''),
 ('FilterText', ''),
 ('AnalysisEnabled', 1),
 ('OutputQuantities', OrderedDict()),
 ('NoiseOutputQuantities', OrderedDict()),
 ('Name', 'LinearFrequency'),
 ('LinearFrequencyData',
 [False, 0.1, False, '', False]),
 ('SweepDefinition',
 OrderedDict([('Variable', 'Freq'),
```

(continues on next page)

(continued from previous page)

```
('Data', 'LINC 1GHz 5GHz 501'),
('OffsetF1', False),
('Synchronize', 0))))
```

### NexximDC

```
OrderedDict([('DataBlockID', 15),
 ('OptionName', 'Default Options'),
 ('AdditionalOptions', ''),
 ('AlterBlockName', ''),
 ('FilterText', ''),
 ('AnalysisEnabled', 1),
 ('OutputQuantities', OrderedDict()),
 ('NoiseOutputQuantities', OrderedDict()),
 ('Name', 'LinearFrequency')])
```

### NexximTransient

```
OrderedDict([('DataBlockID', 10),
 ('OptionName', 'Default Options'),
 ('AdditionalOptions', ''),
 ('AlterBlockName', ''),
 ('FilterText', ''),
 ('AnalysisEnabled', 1),
 ('OutputQuantities', OrderedDict()),
 ('NoiseOutputQuantities', OrderedDict()),
 ('Name', 'LinearFrequency'),
 ('TransientData', ['0.1ns', '10ns']),
 ('TransientNoiseData',
 [False, '', '', 0, 1, 0, False, 1]),
 ('TransientOtherData', ['default'])])
```

### NexximQuickEye

```
OrderedDict([('DataBlockID', 28),
 ('OptionName', 'Default Options'),
 ('AdditionalOptions', ''),
 ('AlterBlockName', ''),
 ('FilterText', ''),
 ('AnalysisEnabled', 1),
 ('OutputQuantities', OrderedDict()),
 ('NoiseOutputQuantities', OrderedDict()),
 ('Name', 'QuickEyeAnalysis'),
 ('QuickEyeAnalysis',
 [False, '1e-9', False, '0', '', True]))])
```

### NexximVerifEye

```
OrderedDict([('DataBlockID', 27),
 ('OptionName', 'Default Options'),
 ('AdditionalOptions', '')])
```

(continues on next page)

(continued from previous page)

```
('AlterBlockName', ''),
('FilterText', ''),
('AnalysisEnabled', 1),
('OutputQuantities', OrderedDict()),
('NoiseOutputQuantities', OrderedDict()),
('Name', 'VerifEyeAnalysis'),
('VerifEyeAnalysis',
 [False, '1e-9', False, '0', '', True]))
```

**NexximAMI**

```
OrderedDict([
 ('DataBlockID', 29),
 ('OptionName', 'Default Options'),
 ('AdditionalOptions', ''),
 ('AlterBlockName', ''),
 ('FilterText', ''),
 ('AnalysisEnabled', 1),
 ('OutputQuantities', OrderedDict()),
 ('NoiseOutputQuantities', OrderedDict()),
 ('Name', 'AMIAutomation'),
 ('InputType', 1),
 ('DataBlockSize', 10000),
 ('AMIAutomation', [32, False, False]))]
```

## 2.18.8 Twin Builder templates and arguments

This section lists all setup templates with their default values and keys available in Twin Builder.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Hfss

hfss = Hfss()
Any property of this setup can be found on this page.
setup = hfss.create_setup()
setup.props["AdaptMultipleFreqs"] = True
setup.update()
```

**TR**

```
OrderedDict()
```

## 2.18.9 RMXprt templates and arguments

This section lists all setup templates with their default values and keys available in RMXprt.

You can edit a setup after it is created. Here is an example:

```
from pyaedt import Hfss

hfss = Hfss()
Any property of this setup can be found on this page.
setup = hfss.create_setup()
setup.props["AdaptMultipleFreqs"] = True
setup.update()
```

**GRM**

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('RatedPowerFactor', '0.8'),
 ('Frequency', '60Hz'),
 ('CapacitivePowerFactor', False)])
```

**DFIG**

```
OrderedDict([('Enabled', True),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('OperationType', 'Wind Generator'),
 ('LoadType', 'InfiniteBus'),
 ('RatedPowerFactor', '0.8'),
 ('Frequency', '60Hz'),
 ('CapacitivePowerFactor', False)])
```

**TPIM**

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('Frequency', '60Hz'),
 ('WindingConnection', 0)])
```

**TPSM**

```
OrderedDict([('Enabled', True),
 ('RatedOutputPower', '100'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('OperationType', 'Generator'),
 ('LoadType', 'InfiniteBus'),
 ('RatedPowerFactor', 0.8),
 ('WindingConnection', False),
 ('ExciterEfficiency', 90),
 ('StartingFieldResistance', '0ohm'),
 ('InputExcitingCurrent', False),
 ('ExcitingCurrent', '0A')])
```

**BLDC**

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel')])
```

**ASSM**

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel')])
```

**PMDC**

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel')])
```

**SRM**

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
```

(continues on next page)

(continued from previous page)

```
('RatedSpeed', '1000rpm'),
('OperatingTemperature', '75cel'))
```

## LSSM

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('WindingConnection', False)])
```

## UNIM

```
OrderedDict([('Enabled', True),
 ('OperationType', 'Motor'),
 ('LoadType', 'ConstPower'),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('Frequency', '60Hz')])
```

## DCM

```
OrderedDict([('Enabled', True),
 ('RatedOutputPower', '1kW'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('OperationType', 'Generator'),
 ('LoadType', 'InfiniteBus'),
 ('FieldExcitingType', False),
 ('DeterminedbyRatedSpeed', False),
 ('ExcitingVoltage', '100V'),
 ('SeriesResistance', '1ohm')])
```

## CPSM

```
OrderedDict([('Enabled', True),
 ('RatedOutputPower', '100'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('OperationType', 'Generator'),
 ('LoadType', 'InfiniteBus'),
 ('RatedPowerFactor', '0.8'),
 ('InputExcitingCurrent', False),
 ('ExcitingCurrent', '0A')])
```

**NSSM**

```
OrderedDict([('Enabled', True),
 ('RatedOutputPower', '100'),
 ('RatedVoltage', '100V'),
 ('RatedSpeed', '1000rpm'),
 ('OperatingTemperature', '75cel'),
 ('OperationType', 'Generator'),
 ('LoadType', 'InfiniteBus'),
 ('RatedPowerFactor', 0.8),
 ('WindingConnection', False),
 ('ExciterEfficiency', 90),
 ('StartingFieldResistance', '0ohm'),
 ('InputExcitingCurrent', False),
 ('ExcitingCurrent', '0A')])
```

## 2.19 Cable modeling

The **Cable Modeling** module includes several methods to work with the **Cable Modeling HFSS Beta** feature:

- `create_cable` to create all available types of cables: bundle, straight wire and twisted pair.
- `update_cable_properties` to update all cables properties for all cable types.
- `update_shielding` to update only the shielding jacket type for bundle cable.
- `remove_cables` to remove cables.
- `add_cable_to_bundle` to add a cable or a list of cables to a bundle.
- `create_clock_source` to create a clock source.
- `update_clock_source` to update a clock source.
- `remove_source` to remove a source.
- `remove_all_sources` to remove all sources.
- `create_pwl_source` to create a pwl source.
- `create_pwl_source_from_file` to create a pwl source from file.
- `update_pwl_source` to update a pwl source.
- `create_cable_harness` to create a cable harness.

They are accessible through:

|                                  |                                     |
|----------------------------------|-------------------------------------|
| <code>CableModeling.Cable</code> | Contains all common Cable features. |
|----------------------------------|-------------------------------------|

## 2.19.1 pyaedt.modules.CableModeling.Cable

```
class pyaedt.modules.CableModeling.Cable(app,json_file_name=None,working_dir=None)
```

Contains all common Cable features.

### Parameters

**app**  
[pyaedt.hfss.Hfss]

**json\_file\_name**  
[str, dict, optional] Full path to either the JSON file or dictionary containing the cable information.

**working\_dir**  
[str, optional] Working directory.

### Examples

```
>>> from pyaedt import Hfss
>>> from pyaedt.modules.CableModeling import Cable
>>> hfss = Hfss()
>>> cable_class = Cable(hfss)
```

```
__init__(app,json_file_name=None,working_dir=None)
```

### Methods

|                               |                                                                               |
|-------------------------------|-------------------------------------------------------------------------------|
| add_cable_to_bundle()         | Add a cable to an existing cable bundle.                                      |
| create_cable()                | Create a cable.                                                               |
| create_cable_harness()        | Create cable harness.                                                         |
| create_clock_source()         | Create a clock source.                                                        |
| create_pwl_source()           | Create a pwl source.                                                          |
| create_pwl_source_from_file() | Create a pwl source from file.                                                |
| remove_all_sources()          | Remove all sources.                                                           |
| remove_cables()               | Remove a list of cables.                                                      |
| remove_source()               | Remove source.                                                                |
| update_cable_properties()     | Update cable properties for all cable types.                                  |
| update_clock_source()         | Update clock source.                                                          |
| update_pwl_source()           | Update pwl source.                                                            |
| update_shielding()            | Create jacket type when cable type is bundle and jacket type is braid shield. |

Cable bundle creation example:

```
from pyaedt import Hfss
from pyaedt.generic.DataHandlers import json_to_dict
from pyaedt.modules.CableModeling import Cable

hfss = Hfss(projectname=project_path, specified_version="2023.1",
 non_graphical=False, new_desktop_session=True,
 close_on_exit=True, student_version=False)
```

(continues on next page)

(continued from previous page)

```
This call returns a dictionary out of the JSON file
cable_props = json_to_dict(json_path)
This example shows how to manually change from script the cable properties
cable_props["Add_Cable"] = "True"
cable_props["Cable_prop"]["CableType"] = "bundle"
cable_props["Cable_prop"]["IsJacketTypeInsulation"] = "True"
cable_props["CableManager"]["Definitions"]["CableBundle"]["BundleParams"][
 "InsulationJacketParams"][
 "InsThickness"
] = "3.66mm"
cable_props["CableManager"]["Definitions"]["CableBundle"]["BundleParams"][
 "InsulationJacketParams"][
 "JacketMaterial"
] = "pec"
cable_props["CableManager"]["Definitions"]["CableBundle"]["BundleParams"][
 "InsulationJacketParams"][
 "InnerDiameter"
] = "2.88mm"
cable_props["CableManager"]["Definitions"]["CableBundle"]["BundleAttribs"]["Name"] =
 "Bundle_Cable_Insulation"
This call returns the Cable class
cable = Cable(hfss, cable_props)
This call creates the cable bundle
cable.create_cable()
```

Clock source creation example:

```
from pyaedt import Hfss
from pyaedt.generic.DataHandlers import json_to_dict
from pyaedt.modules.CableModeling import Cable

hfss = Hfss(projectname=project_path, specified_version="2023.1",
 non_graphical=False, new_desktop_session=True,
 close_on_exit=True, student_version=False)
This call returns a dictionary out of the JSON file
cable_props = json_to_dict(json_path)
This example shows how to manually change from script the clock source properties
cable_props["Add_Cable"] = "False"
cable_props["Update_Cable"] = "False"
cable_props["Add_CablesToBundle"] = "False"
cable_props["Remove_Cable"] = "False"
cable_props["Add_Source"] = "True"
cable_props["Source_prop"]["AddClockSource"] = "True"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["ClockSignalParams"]["Period"] =
 "40us"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["ClockSignalParams"][
 "LowPulseVal"] = "0.1V"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["ClockSignalParams"][
 "HighPulseVal"] = "2V"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["ClockSignalParams"]["Risetime"] =
 "5us"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["ClockSignalParams"]["Falltime"]
```

(continues on next page)

(continued from previous page)

```

↳ "] = "10us"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["ClockSignalParams"][
 ↳ "PulseWidth"] = "23us"
cable_props["CableManager"]["TDSources"]["ClockSourceDef"]["TDSourceAttribs"]["Name"] =
 ↳ "clock_test_1"
This call returns the Cable class
cable = Cable(hfss, cable_props)
This call creates the clock source
cable.create_clock_source()

```

Cable harness creation example:

```

from pyaedt import Hfss
from pyaedt.generic.DataHandlers import json_to_dict
from pyaedt.modules.CableModeling import Cable

hfss = Hfss(projectname=project_path, specified_version="2023.1",
 non_graphical=False, new_desktop_session=True,
 close_on_exit=True, student_version=False)
This call returns a dictionary out of the JSON file
cable_props = json_to_dict(json_path)
This example shows how to manually change from script the cable harness properties
cable_props["Add_Cable"] = "False"
cable_props["Update_Cable"] = "False"
cable_props["Add_CablesToBundle"] = "False"
cable_props["Remove_Cable"] = "False"
cable_props["Add_Source"] = "False"
cable_props["Update_Source"] = "False"
cable_props["Remove_Source"] = "False"
cable_props["Add_CableHarness"] = "True"
cable_props["CableHarness_prop"]["Name"] = "cable_harness_test"
cable_props["CableHarness_prop"]["Bundle"] = "New_updated_name_cable_bundle_insulation"
cable_props["CableHarness_prop"]["TwistAngleAlongRoute"] = "20deg"
cable_props["CableHarness_prop"]["Polyline"] = "Polyline1"
cable_props["CableHarness_prop"]["AutoOrient"] = "False"
cable_props["CableHarness_prop"]["XAxis"] = "Undefined"
cable_props["CableHarness_prop"]["XAxisOrigin"] = ["0mm", "0mm", "0mm"]
cable_props["CableHarness_prop"]["XAxisEnd"] = ["0mm", "0mm", "0mm"]
cable_props["CableHarness_prop"]["ReverseYAxisDirection"] = "True"
cable_props["CableHarness_prop"]["CableTerminationsToInclude"][0]["CableName"] =
 ↳ "straight_wire_cable"
cable_props["CableHarness_prop"]["CableTerminationsToInclude"][1]["CableName"] =
 ↳ "straight_wire_cable1"
cable_props["CableHarness_prop"]["CableTerminationsToInclude"][2]["CableName"] =
 ↳ "straight_wire_cable2"
This call returns the Cable class
cable = Cable(hfss, cable_props)
This call creates the cable harness
cable.create_cable_harness()

```

## EXAMPLES

End-to-end examples show how you can use PyAEDT. If PyAEDT 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.

---

### 3.1 EDB examples

EDB is a powerful API for efficiently controlling PCB data. You can either use EDB standalone or embedded in HFSS 3D Layout in AEDT. The EDB class is now part of the PyEDB package, which is currently installed with PyAEDT and backward-compatible with PyAEDT. All EDB related examples have been moved to the [Examples page](#) in the PyEDB documentation. These examples use EDB (Electronics Database) with PyAEDT.

```
Launch the latest installed version of AEDB.
import pyaedt
edb = pyaedt.Edb("mylayout.aedb")

You can also launch EDB directly from PyEDB.

import pyedb
edb = pyedb.Edb("mylayout.aedb")
```

### 3.2 HFSS 3D Layout examples

These examples use PyAEDT to show some end-to-end workflows for HFSS 3D Layout. It includes model generation, setup, meshing, and post-processing.

### 3.3 General model and setup examples

These examples use PyAEDT to show some general model and simulation setup features inside AEDT.

### 3.4 HFSS examples

These examples use PyAEDT to show some end-to-end workflows for HFSS 3D. This includes model generation, setup, meshing, and postprocessing.

### 3.5 SBR+ examples

These examples use PyAEDT to show some end-to-end workflows for HFSS SBR+. This includes model generation, setup, meshing, and postprocessing.

### 3.6 Maxwell examples

These examples use PyAEDT to show some end-to-end workflows for Maxwell 2D and Maxwell 3D. This includes model generation, setup, meshing, and postprocessing. Examples cover different Maxwell solution types (Eddy Current, Magnetostatic, and Transient).

### 3.7 Icepak examples

These examples use PyAEDT to show end-to-end workflows for Icepak. This includes schematic generation, setup, and thermal postprocessing.

### 3.8 2D Extractor and Q3D Extractor examples

These examples use PyAEDT to show some end-to-end workflows for 2D Extractor and Q3D Extractor. This includes model generation, setup, and thermal postprocessing.

### 3.9 Multiphysics examples

These examples use PyAEDT to create some multiphysics workflows. They might use an electromagnetic tool like HFSS or Maxwell and a thermal or structural tool like Icepak or Mechanical.

## 3.10 Circuit examples

These examples use PyAEDT to show some end-to-end workflows for Circuit. This includes schematic generation, setup, and postprocessing.

## 3.11 EMIT examples

These examples use PyAEDT to show some end-to-end workflows for EMIT. This includes schematic generation, setup, and postprocessing.

## 3.12 Twin Builder examples

These examples use PyAEDT to show some end-to-end workflows for Twin Builder. This includes schematic generation, setup, and postprocessing.

### 3.12.1 EDB examples

EDB is a powerful API for efficiently controlling PCB data. You can either use EDB standalone or embedded in HFSS 3D Layout in AEDT. The EDB class is now part of the PyEDB package, which is currently installed with PyAEDT and backward-compatible with PyAEDT. All EDB related examples have been moved to the [Examples page](#) in the PyEDB documentation. These examples use EDB (Electronics Database) with PyAEDT.

```
Launch the latest installed version of AEDB.
import pyaedt
edb = pyaedt.Edb("mylayout.aedb")

You can also launch EDB directly from PyEDB.

import pyedb
edb = pyedb.Edb("mylayout.aedb")
```

### 3.12.2 HFSS 3D Layout examples

These examples use PyAEDT to show some end-to-end workflows for HFSS 3D Layout. It includes model generation, setup, meshing, and post-processing.

#### HFSS 3D Layout: SIwave DCIR analysis in HFSS 3D Layout

This example shows how you can use configure HFSS 3D Layout for SIwave DCIR analysis.

```
import os
import tempfile
import pyaedt
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Configure EDB for DCIR analysis

Copy example into temporary folder

```
temp_dir = tempfile.gettempdir()
dst_dir = os.path.join(temp_dir, pyaedt.generate_unique_name("pyaedt_dcir"))
os.mkdir(dst_dir)
local_path = pyaedt.downloads.download_aedb(dst_dir)
```

Load example board into EDB

```
appedb = pyaedt.Edb(local_path, edbversion=aedt_version)
```

Create pin group on VRM positive pins

```
gnd_name = "GND"
appedb.siwave.create_pin_group_on_net(
 reference_designator="U3A1",
 net_name="BST_V3P3_S5",
 group_name="U3A1-BST_V3P3_S5")
```

```
('U3A1-BST_V3P3_S5', <pyedb.dotnet.edb_core.edb_data.sources.PinGroup object at
 ↪0x00000258EA1C8070>)
```

Create pin group on VRM negative pins

```
appedb.siwave.create_pin_group_on_net(
 reference_designator="U3A1",
 net_name="GND",
 group_name="U3A1-GND")
```

```
('U3A1-GND', <pyedb.dotnet.edb_core.edb_data.sources.PinGroup object at
 ↪0x00000258EA1C9CC0>)
```

Create voltage source between VRM positive and negative pin groups

```
appedb.siwave.create_voltage_source_on_pin_group(
 pos_pin_group_name="U3A1-BST_V3P3_S5",
 neg_pin_group_name="U3A1-GND",
 magnitude=3.3,
 name="U3A1-BST_V3P3_S5")
)
```

```
True
```

Create pin group on sink component positive pins

```
appedb.siwave.create_pin_group_on_net(
 reference_designator="U2A5",
 net_name="V3P3_S5",
 group_name="U2A5-V3P3_S5")
```

```
('U2A5-V3P3_S5', <pyedb.dotnet.edb_core.edb_data.sources.PinGroup object at
<0x00000258EA273F10>)
```

Create pin group on sink component negative pins

```
appedb.siwave.create_pin_group_on_net(
 reference_designator="U2A5",
 net_name="GND",
 group_name="U2A5-GND")

~~~~~
Create place current source between sink component positive and negative pin groups
appedb.siwave.create_current_source_on_pin_group(
 pos_pin_group_name="U2A5-V3P3_S5",
 neg_pin_group_name="U2A5-GND",
 magnitude=1,
 name="U2A5-V3P3_S5"
)
```

```
True
```

Add SIwave DCIR analysis

```
appedb.siwave.add_siwave_dc_analysis(name="my_setup")
```

```
<pyedb.dotnet.edb_core.edb_data.siwave_simulation_setup_data.SiwaveDCSimulationSetup
<object at 0x00000258EA250130>
```

## Save and close EDB

Save and close EDB.

```
appedb.save_edb()
appedb.close_edb()
```

```
True
```

## Analysis DCIR in AEDT

Launch AEDT and import the configured EDB and analysis DCIR

```
desktop = pyaedt.Desktop(aedt_version, non_graphical=False, new_desktop_session=True)
hfss3dl = pyaedt.Hfss3dLayout(local_path)
hfss3dl.analyze()
hfss3dl.save_project()
```

```
True
```

## Get element data

Get current source

```
current_source = hfss3dl.get_dcir_element_data_current_source(setup="my_setup")
print(current_source)

~~~~~
Get via information

via = hfss3dl.get_dcir_element_data_via(setup="my_setup")
print(via)
```

```
Voltage
U2A5-V3P3_S5 3.294553
X
J1-6<BOTTOM, TOP> 0.003632
J1A6-3<BOTTOM, TOP> 0.034595
J1B2-2<BOTTOM, GND> 0.007493
J1B2-2<GND, TOP> 0.007493
J2A1-4<BOTTOM, TOP> 0.026975
...
...
via_4873<GND, TOP> 0.056947
via_4874<BOTTOM, GND> 0.086106
via_4874<GND, TOP> 0.086106
via_4875<BOTTOM, GND> 0.082550
via_4875<GND, TOP> 0.082550

[883 rows x 1 columns]
```

## Get voltage

Get voltage from dcir solution data

```
voltage = hfss3dl.get_dcir_solution_data(setup="my_setup", show="Sources", category=
 "Voltage")
print({expression: voltage.data_magnitude(expression) for expression in voltage.
 expressions})
```

```
{'SeriesRV(U3A1-BST_V3P3_S5)': [1.000632289474], 'V(U2A5-V3P3_S5)': [3294.5533607909997]}
```

## Close AEDT

```
desktop.release_desktop()
```

```
True
```

**Total running time of the script:** (2 minutes 48.669 seconds)

## HFSS 3D Layout: PCB and EDB in 3D layout

This example shows how you can use HFSS 3D Layout combined with EDB to interact with a 3D layout.

```
import os
import tempfile
import pyaedt

tmpfold = tempfile.gettempdir()
temp_folder = os.path.join(tmpfold, pyaedt.generate_unique_name("Example"))
if not os.path.exists(temp_folder):
 os.makedirs(temp_folder)
print(temp_folder)
```

```
D:\Temp\Example_SSDWFL
```

## Copy example into temporary folder

Copy an example into the temporary folder.

```
targetfile = pyaedt.downloads.download_aedb()
print(targetfile)
aedb_file = targetfile[:-12] + "aedt"
```

```
D:\Temp\PyAEDTEXamples\edb/Galileo.aedb\edb.def
```

## Set non-graphical mode

Set non-graphical mode. You can set non\_graphical either to True or False.

```
non_graphical = False
NewThread = True
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Initialize AEDT and launch HFSS 3D Layout

Initialize AEDT and launch HFSS 3D Layout. The h3d object contains the pyaedt.Edb class query methods.

```
d = pyaedt.launch_desktop(aedt_version, non_graphical, NewThread)
if os.path.exists(aedt_file):
 os.remove(aedt_file)
h3d = pyaedt.Hfss3dLayout(targetfile)
h3d.save_project(os.path.join(temp_folder, "edb_demo.aedt"))
```

```
True
```

## Print boundaries

Print boundaries from the setups object.

```
h3d.boundaries
```

```
[]
```

## Hide all nets

Hide all nets.

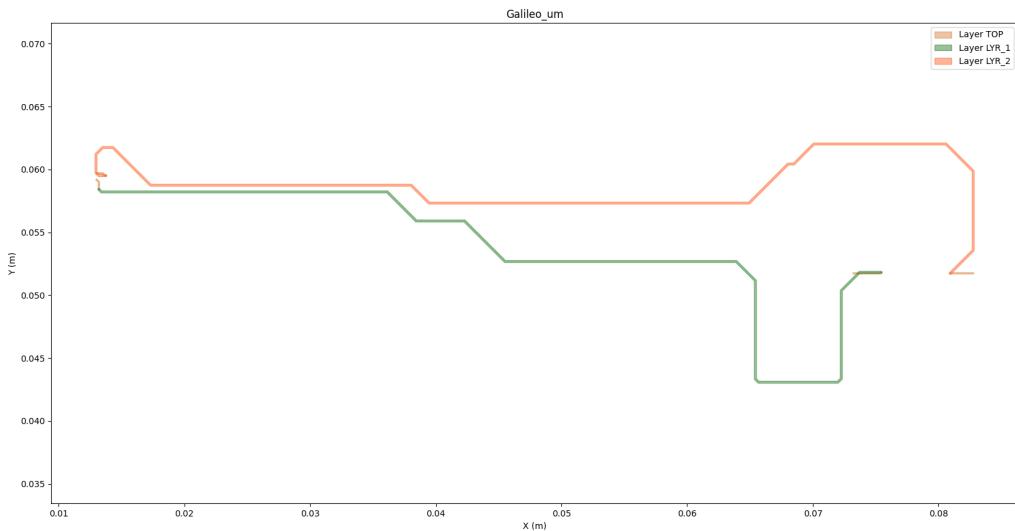
```
h3d.modeler.change_net_visibility(visible=False)
```

```
True
```

## Show only two nets

Show only two specified nets.

```
h3d.modeler.change_net_visibility(["A0_GPIO", "A0_MUX"], visible=True)
edb = h3d.modeler.edb
edb.nets.plot(["A0_GPIO", "A0_MUX"])
```



## Show all layers

Show all layers.

```
for layer in h3d.modeler.layers.all_signal_layers:
 layer.is_visible = True
```

## Change layer color

Change the layer color.

```
layer = h3d.modeler.layers.layers[h3d.modeler.layers.layer_id("TOP")]
layer.set_layer_color(0, 255, 0)
h3d.modeler.fit_all()
```

## Disable component visibility

Disable component visibility for "TOP" and "BOTTOM". The `pyaedt.modules.LayerStackup.Layer.update_stackup_layer()` method applies modifications to the layout.

```
top = h3d.modeler.layers.layers[h3d.modeler.layers.layer_id("TOP")]
top.is_visible_component = False

bot = h3d.modeler.layers.layers[h3d.modeler.layers.layer_id("BOTTOM")]
bot.is_visible_component = False
```

## Fit all

Fit all so that you can visualize all.

```
h3d.modeler.fit_all()
```

## Close AEDT

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

```
h3d.close_project()
d.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 3.209 seconds)

## HFSS 3D Layout: parametric via analysis

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

### Perform required imports

Perform required imports.

```
import pyaedt
import os
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = True
```

## Launch AEDT

Launch AEDT 2023 R2 in graphical mode.

```
h3d = pyaedt.Hfss3dLayout(specified_version=aedt_version, new_desktop_session=True, non_
graphical=non_graphical)
```

## Set up variables

Set up all parametric variables to use in the layout.

```
h3d["viatotrace"] = "5mm"
h3d["viatovia"] = "10mm"
h3d["w1"] = "1mm"
h3d["sp"] = "0.5mm"
h3d["len"] = "50mm"
```

## Add stackup layers

Add stackup layers.

```
h3d.modeler.layers.add_layer(layer="GND", layer_type="signal", thickness="0",_
isnegative=True)
h3d.modeler.layers.add_layer(layer="diel", layer_type="dielectric", thickness="0.2mm",_
material="FR4_epoxy")
h3d.modeler.layers.add_layer(layer="TOP", layer_type="signal", thickness="0.035mm",_
elevation="0.2mm")
```

```
<pyaedt.modules.LayerStackup.Layer object at 0x00000258E71B2770>
```

## Create signal net and ground planes

Create a signal net and ground planes.

```
h3d.modeler.create_line(layer="TOP",
center_line_coordinates=[[0, 0], [len, 0]],
lw=w1,
name="microstrip",
net="microstrip")
h3d.modeler.create_rectangle(layer="TOP", origin=[0, "-w1/2-sp"], sizes=[len, "-w1/2-
sp-20mm"])
h3d.modeler.create_rectangle(layer="TOP", origin=[0, "w1/2+sp"], sizes=[len, "w1/
2+sp+20mm"])
```

```
<pyaedt.modeler.pcb.object3dlayout.Rect3dLayout object at 0x00000258E71B1F00>
```

## Create vias

Create vias with parametric positions.

```
h3d.modeler.create_via(x="viatovia", y="-viatotrace", name="via1")
h3d.modeler.create_via(x="viatovia", y="viatotrace", name="via2")
h3d.modeler.create_via(x="2*viatovia", y="-viatotrace")
h3d.modeler.create_via(x="2*viatovia", y="viatotrace")
h3d.modeler.create_via(x="3*viatovia", y="-viatotrace")
h3d.modeler.create_via(x="3*viatovia", y="viatotrace")
```

```
<pyaedt.modeler.pcb.object3dlayout.Pins3DLayout object at 0x00000258E71B1360>
```

## Create circuit ports

Create circuit ports.

```
h3d.create_edge_port("microstrip", 0)
h3d.create_edge_port("microstrip", 2)
```

```
<pyaedt.modules.Boundary.BoundaryObject3dLayout object at 0x00000258E71B2080>
```

## Create setup and sweep

Create a setup and a sweep.

```
setup = h3d.create_setup()
h3d.create_linear_count_sweep(setup=setup.name, unit="GHz", start_frequency=3, stop_
 ↴frequency=7,
 num_of_freq_points=1001, save_fields=False, sweep_type=
 ↴"Interpolating",
 interpolation_tol_percent=1, interpolation_max_
 ↴solutions=255, use_q3d_for_dc=False)
```

```
<pyaedt.modules.SolveSweeps.SweepHFSS3DLayout object at 0x00000258E708B1F0>
```

## Solve and plot results

Solve and plot the results.

```
h3d.analyze()
traces = h3d.get_traces_for_plot(first_element_filter="Port1")
h3d.post.create_report(traces, variations=h3d.available_variations.nominal_w_values_dict)
```

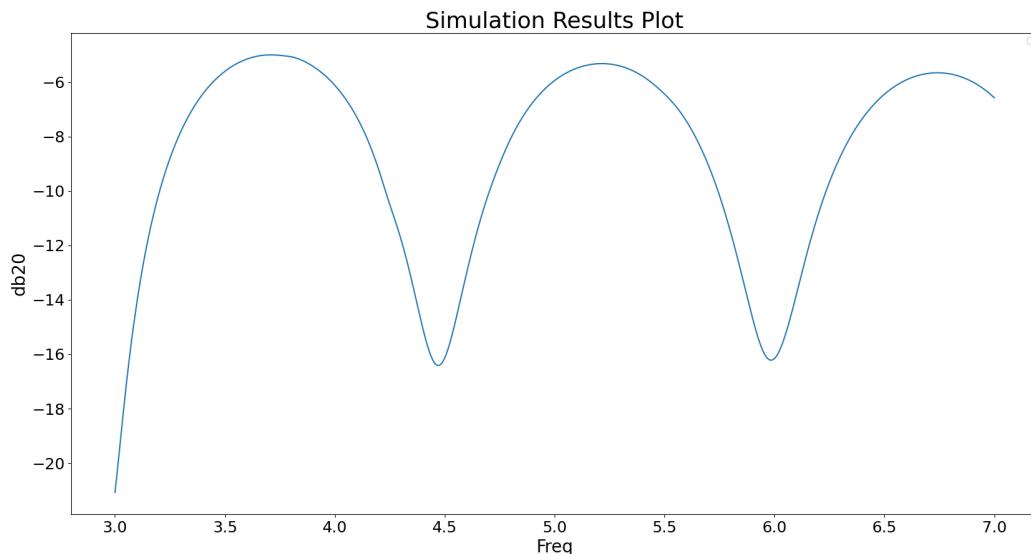
```
<pyaedt.modules.report_templates.Standard object at 0x00000258F6A19C60>
```

## Create report outside AEDT

Create a report using Matplotlib.

```
traces = h3d.get_traces_for_plot(first_element_filter="Port1", category="S")

solutions = h3d.post.get_solution_data(expressions=traces)
solutions.plot(formula="db20")
```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Close AEDT

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

```
h3d.release_desktop()
```

```
True
```

**Total running time of the script:** (2 minutes 51.044 seconds)

## HFSS: 3D Components

This example shows how you can use PyAEDT to place 3D Components in Hfss and in Hfss 3D Layout.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

### Common Properties

Set common properties.

```
trace_width = 0.6
trace_length = 30
diel_height = "121mil"
sig_height = "5mil"
max_steps = 3
freq = "3GHz"
new_session = True
```

## 3D Component Definition

File to be used in the example

```
component3d = pyaedt.downloads.download_file("component_3d", "SMA_RF_Jack.a3dcomp",)
```

### Hfss Example

This example will create a stackup in Hfss place a 3d component, build a ground plane, a trace, create excitation and solve it in Hfss.

## Launch Hfss

Launch HFSS application

```
hfss = pyaedt.Hfss(new_desktop_session=True, specified_version=aedt_version, non_
graphical=non_graphical)

hfss.solution_type = "Terminal"
```

## Insert 3d Component

To insert a 3d component we need to read parameters and then import in Hfss.

```
comp_param = hfss.get_components3d_vars(component3d)
hfss.modeler.insert_3d_component(component3d, comp_param)
```

```
<pyaedt.modeler.cad.components_3d.UserDefinedComponent object at 0x00000258F25686D0>
```

## Add a new Stackup

Pyaedt has a Stackup class which allows to parametrize stacked structures.

```
stackup = hfss.add_stackup_3d()
s1 = stackup.add_signal_layer("L1", thickness=sig_height)
d1 = stackup.add_dielectric_layer("D1", thickness=diel_height)
g1 = stackup.add_ground_layer("G1", thickness=sig_height)
```

## Define stackup extensions

Define stackup elevation and size. Defines also the stackup origin.

```
stackup.start_position = "-131mil"
stackup.dielectric_width = "20mm"
stackup.dielectric_length = "40mm"
stackup.dielectric_y_position = "-dielectric_width/2"
stackup.dielectric_x_position = "-dielectric_length/4"
```

## Padstack Definition

Padstacks are needed to create a clearance around 3d component since intersections are not allowed. There will be 1 padstack for Gnd and 1 for pin.

```
p1 = stackup.add_padstack("gnd_via", material="cloned_copper")
p1.set_start_layer("L1")
p1.set_stop_layer("G1")
p1.set_all_antipad_value(1.3)
p1.set_all_pad_value(0)
p1.num_sides = 8
```

(continues on next page)

(continued from previous page)

```
p1.add_via(-3.2, -3.2)
p1.add_via(-3.2, 3.2)
p1.add_via(3.2, -3.2)
p1.add_via(3.2, 3.2)
p2 = stackup.add_padstack("signal_via", material="cloned_copper")

p2.set_start_layer("L1")
p2.set_stop_layer("G1")
p2.set_all_antipad_value(0.7)
p2.set_all_pad_value(0)
p2.padstacks_by_layer["L1"].pad_radius = 0.3048
p2.add_via(0, 0)
```

<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258F2790520>

## Trace Definition

The trace will connect the pin to the port on layer L1.

```
t1 = s1.add_trace(trace_width, trace_length)
rect1 = hfss.modeler.create_rectangle(orientation=hfss.PLANE.YZ,
 origin=[0.75*dielectric_length, -5*_
+ t1.width.name, "0mm"],
 sizes=[15* + t1.width.name, "-3*" +_
stackup.thickness.name])
p1 = hfss.wave_port(assignment=rect1, reference="G1", name="P1")
```

## Set Simulation Boundaries

Define regione and simulation boundaries.

```
hfss.change_material_override(True)
region = hfss.modeler.create_region([0, 0, 0, 0, 0, 100])
sheets = [i for i in region.faces]
hfss.assign_radiation_boundary_to_faces(sheets)
```

<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F2790E50>

## Create Setup

Iterations will be reduced to reduce simulation time.

```
setup1 = hfss.create_setup()
sweep1 = hfss.create_linear_count_sweep(setup1.name, "GHz", 0.01, 8, 1601, sweep_type=
+ "Interpolating")
setup1.props["Frequency"] = freq
setup1.props["MaximumPasses"] = max_steps
```

## Solve Setup

Save the project first and then solve the setup.

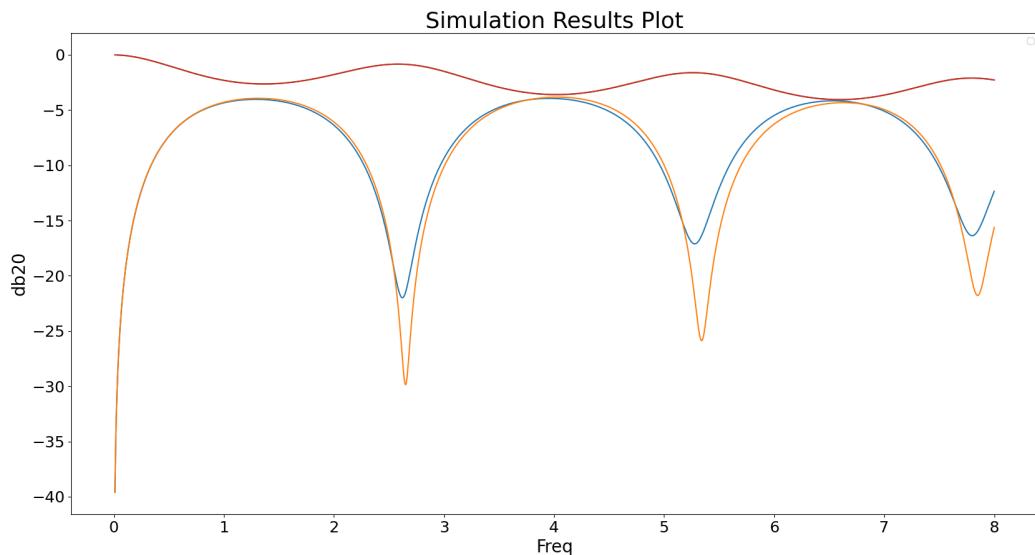
```
hfss.save_project()
hfss.analyze()
```

```
True
```

## Plot results

Plot the results when analysis is completed.

```
traces = hfss.get_traces_for_plot(category="S")
solutions = hfss.post.get_solution_data(traces)
solutions.plot(traces, formula="db20")
```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Hfss 3D Layout Example

Previous example will be repeated this time in Hfss 3d Layout. Small differences are expected in layout but results should be similar.

### Launch Hfss3dLayout

Launch HFSS3dLayout application

```
h3d = pyaedt.Hfss3dLayout()
```

### Add stackup layers

Add stackup layers.

```
l1 = h3d.modeler.layers.add_layer("L1", "signal", thickness=sig_height)
h3d.modeler.layers.add_layer("diel", "dielectric", thickness=diel_height, material="FR4_"
 ↪epoxy")
h3d.modeler.layers.add_layer("G1", "signal", thickness=sig_height, isnegative=True)
```

```
<pyaedt.modules.LayerStackup.Layer object at 0x00000258F2A07460>
```

### Place 3d Component

Place a 3d component by specifying the .a3dcomp file path.

```
comp = h3d.modeler.place_3d_component(
 component_path=component3d, number_of_terminals=1, placement_layer="G1", component_
 ↪name="my_connector",
 pos_x=0.000, pos_y=0.000
)
```

### Create signal net and ground planes

Create a signal net and ground planes.

```
h3d["len"] = str(trace_length) + "mm"
h3d["w1"] = str(trace_width) + "mm"

line = h3d.modeler.create_line("L1", [[0, 0], ["len", 0]], lw="w1", name="microstrip", ↪
 ↪net="microstrip")
h3d.create_edge_port(line, h3d.modeler[line.name].top_edge_x, is_wave_port=True, wave_
 ↪horizontal_extension=15)
```

```
<pyaedt.modules.Boundary.BoundaryObject3dLayout object at 0x00000258F252A110>
```

## Create void on Ground plane for pin

Create a void.

```
h3d.modeler.create_circle("G1", 0, 0, 0.5)
```

```
<pyaedt.modeler.pcb.object3dlayout.Circle3dLayout object at 0x00000258F2A04D60>
```

## Create Setup

Iterations will be reduced to reduce simulation time.

```
h3d.set_meshing_settings(mesh_method="PhiPlus", enable_intersections_check=False)
h3d.edit_hfss_extents(diel_extent_horizontal_padding="0.2", air_vertical_positive_
 ↪padding="0",
 air_vertical_negative_padding="2", airbox_values_as_dim=False)
setup1 = h3d.create_setup()
sweep1 = h3d.create_linear_count_sweep(setup1.name,
 "GHz",
 0.01,
 8,
 1601,
 sweep_type="Interpolating")
setup1.props["AdaptiveSettings"]["SingleFrequencyDataList"]["AdaptiveFrequencyData"][
 ↪"AdaptiveFrequency"] = freq
setup1.props["AdaptiveSettings"]["SingleFrequencyDataList"]["AdaptiveFrequencyData"][
 ↪"MaxPasses"] = max_steps
```

## Solve Setup

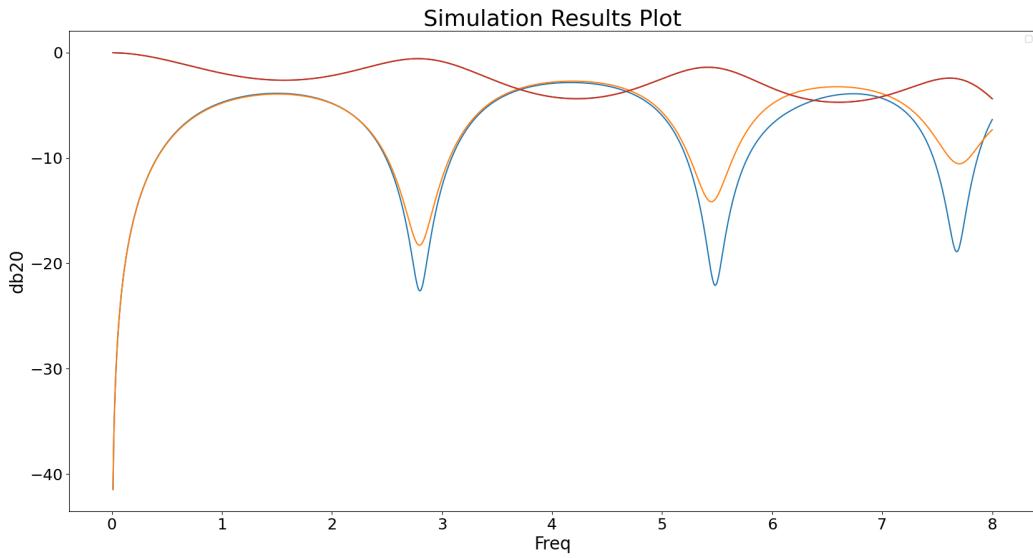
```
h3d.analyze()
```

```
True
```

## Plot results

```
traces = h3d.get_traces_for_plot(category="S")
solutions = h3d.post.get_solution_data(traces)
solutions.plot(traces, formula="db20")

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



```
No artists with labels found to put in legend. Note that artists whose label start with _
an underscore are ignored when legend() is called with no argument.
```

```
True
```

**Total running time of the script:** (4 minutes 17.403 seconds)

### 3.12.3 General model and setup examples

These examples use PyAEDT to show some general model and simulation setup features inside AEDT.

#### General: configuration files

This example shows how you can use PyAEDT to export configuration files and re-use them to import in a new project. A configuration file is supported by these applications:

- HFSS
- 2D Extractor and Q3D Extractor
- Maxwell
- Icepak (in AEDT)
- Mechanical (in AEDT)

The following sections are covered:

- Variables
- Mesh operations (except Icepak)
- Setup and optimetrics
- Material properties
- Object properties

- Boundaries and excitations

When a boundary is attached to a face, the tool tries to match it with a FaceByPosition on the same object name on the target design. If, for any reason, this face position has changed or the object name in the target design has changed, the boundary fails to apply.

## Perform required imports

Perform required imports from PyAEDT.

```
import os
import pyaedt
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set non\_graphical either to True or False.

```
non_graphical = False
```

## Open project

Download the project, open it, and save it to the temporary folder.

```
project_full_name = pyaedt.downloads.download_icepak(pyaedt.generate_unique_folder_
name(folder_name="Graphic_Card"))

ipk = pyaedt.Icepak(projectname=project_full_name, specified_version=aedt_version,
new_desktop_session=True, non_graphical=non_graphical)
ipk.autosave_disable()
```

```
True
```

## Create source blocks

Create a source block on the CPU and memories.

```
ipk.create_source_block(object_name="CPU", input_power="25W")
ipk.create_source_block(object_name=["MEMORY1", "MEMORY1_1"], input_power="5W")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F6A3AAD0>
```

## Assign boundaries

Assign the opening and grille.

```
region = ipk.modeler["Region"]
ipk.assign_openings(air_faces=region.bottom_face_x.id)
ipk.assign_grille(air_faces=region.top_face_x.id, free_area_ratio=0.8)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F6A6E860>
```

## Create setup

Create the setup. Properties can be set up from the `setup` object with getters and setters. They dont have to perfectly match the property syntax.

```
setup1 = ipk.create_setup()
setup1["FlowRegime"] = "Turbulent"
setup1["Max Iterations"] = 5
setup1["Solver Type Pressure"] = "flex"
setup1["Solver Type Temperature"] = "flex"
ipk.save_project(r"C:\temp\Graphic_card.aedt")
```

```
True
```

## Export project to step file

Export the current project to the step file.

```
filename = ipk.design_name
file_path = os.path.join(ipk.working_directory, filename + ".step")
ipk.export_3d_model(file_name=filename, file_path=ipk.working_directory, file_format=".step",
 object_list=[], removed_objects=[])
```

```
True
```

## Export configuration files

Export the configuration files. You can optionally disable the export and import sections. Supported formats are json and toml files

```
conf_file = ipk.configurations.export_config(os.path.join(ipk.working_directory, "config.toml"))
ipk.close_project()
```

```
True
```

## Create project

Create an Icepak project and import the step.

```
app = pyaedt.Icepak(projectname="new_proj_Ipk")
app.modeler.import_3d_cad(file_path)
```

```
True
```

## Import and apply configuration file

Import and apply the configuration file. You can apply all or part of the JSON file that you import using options in the configurations object.

```
out = app.configurations.import_config(conf_file)
app.configurations.results.global_import_success
```

```
True
```

## Close project

Close the project.

```
app.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 18.116 seconds)

## General: coordinate system creation

This example shows how you can use PyAEDT to create and modify coordinate systems in the modeler.

### Perform required imports

Perform required imports

```
import os
import pyaedt
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Launch AEDT in graphical mode

Launch AEDT 2023 R2 in graphical mode.

```
d = pyaedt.launch_desktop(specified_version=aedt_version, non_graphical=non_graphical,
 ↪new_desktop_session=True)
```

## Insert HFSS design

Insert an HFSS design with the default name.

```
hfss = pyaedt.Hfss(projectname=pyaedt.generate_unique_project_name(folder_name=
 ↪"CoordSysDemo"))
```

## Create coordinate system

The coordinate system is centered on the global origin and has the axis aligned to the global coordinate system. The new coordinate system is saved in the object `cs1`.

```
cs1 = hfss.modeler.create_coordinate_system()
```

## Modify coordinate system

The `cs1` object exposes properties and methods to manipulate the coordinate system. The origin can be changed.

```
cs1["OriginX"] = 10
cs1.props["OriginY"] = 10
cs1.props["OriginZ"] = 10

Pointing vectors can be changed

ypoint = [0, -1, 0]
cs1.props["YAxisXvec"] = ypoint[0]
cs1.props["YAxisYvec"] = ypoint[1]
cs1.props["YAxisZvec"] = ypoint[2]
```

## Rename coordinate system

Rename the coordinate system.

```
cs1.rename("newCS")
```

```
True
```

## Change coordinate system mode

Use the `change_cs_mode` method to change the mode. Options are 0 for axis/position, 1 for Euler angle ZXZ, and 2 for Euler angle ZYZ. Here 1 sets Euler angle ZXZ as the mode.

```
cs1.change_cs_mode(1)

In the new mode, these properties can be edited
cs1.props["Phi"] = "10deg"
cs1.props["Theta"] = "22deg"
cs1.props["Psi"] = "30deg"
```

## Delete coordinate system

Delete the coordinate system.

```
cs1.delete()
```

```
True
```

## Create coordinate system by defining axes

Create a coordinate system by defining the axes. During creation, you can specify all coordinate system properties.

```
cs2 = hfss.modeler.create_coordinate_system(
 name="CS2", origin=[1, 2, 3.5], mode="axis", x_pointing=[1, 0, 1], y_pointing=[0, -1,
 ↪ 0]
)
```

## Create coordinate system by defining Euler angles

Create a coordinate system by defining Euler angles.

```
cs3 = hfss.modeler.create_coordinate_system(name="CS3", origin=[2, 2, 2], mode="zyz",
 ↪ phi=10, theta=20, psi=30)
```

## Create coordinate system by defining view

Create a coordinate system by defining the view. Options are "iso", "XY", "XZ", and "XY". Here "iso" is specified. The axes are set automatically.

```
cs4 = hfss.modeler.create_coordinate_system(name="CS4", origin=[1, 0, 0], reference_cs=
 ↪"CS3", mode="view", view="iso")
```

## Create coordinate system by defining axis and angle rotation

Create a coordinate system by defining the axis and angle rotation. When you specify the axis and angle rotation, this data is automatically translated to Euler angles.

```
cs5 = hfss.modeler.create_coordinate_system(name="CS5", mode="axisrotation", u=[1, 0, 0],
 ↪ theta=123)
```

## Create face coordinate system

Face coordinate systems are bound to an object face. First create a box and then define the face coordinate system on one of its faces. To create the reference face for the face coordinate system, you must specify starting and ending points for the axis.

```
box = hfss.modeler.create_box([0, 0, 0], [2, 2, 2])
face = box.faces[0]
fcs1 = hfss.modeler.create_face_coordinate_system(
 face=face, origin=face.edges[0], axis_position=face.edges[1], name="FCS1"
)
```

## Create face coordinate system centered on face

Create a face coordinate system centered on the face with the X axis pointing to the edge vertex.

```
fcs2 = hfss.modeler.create_face_coordinate_system(
 face=face, origin=face, axis_position=face.edges[0].vertices[0], name="FCS2"
)
```

## Swap X and Y axes of face coordinate system

Swap the X axis and Y axis of the face coordinate system. The X axis is the pointing `axis_position` by default. You can optionally select the Y axis.

```
fcs3 = hfss.modeler.create_face_coordinate_system(face=face, origin=face, axis_
 ↪ position=face.edges[0], axis="Y")

Axis can also be changed after coordinate system creation
fcs3.props["WhichAxis"] = "X"
```

## Apply a rotation around Z axis

Apply a rotation around the Z axis. The Z axis of a face coordinate system is always orthogonal to the face. A rotation can be applied at definition. Rotation is expressed in degrees.

```
fcs4 = hfss.modeler.create_face_coordinate_system(face=face, origin=face, axis_
 ↪position=face.edges[1], rotation=10.3)

Rotation can also be changed after coordinate system creation
fcs4.props["ZRotationAngle"] = "3deg"
```

## Apply offset to X and Y axes of face coordinate system

Apply an offset to the X axis and Y axis of a face coordinate system. The offset is in respect to the face coordinate system itself.

```
fcs5 = hfss.modeler.create_face_coordinate_system(
 face=face, origin=face, axis_position=face.edges[2], offset=[0.5, 0.3]
)

The offset can also be changed after the coordinate system is created.
fcs5.props["XOffset"] = "0.2mm"
fcs5.props["YOffset"] = "0.1mm"
```

## Create coordinate system relative to face coordinate system

Create a coordinate system relative to a face coordinate system. Coordinate systems and face coordinate systems interact with each other.

```
face = box.faces[1]
fcs6 = hfss.modeler.create_face_coordinate_system(face=face, origin=face, axis_
 ↪position=face.edges[0])
cs_fcs = hfss.modeler.create_coordinate_system(
 name="CS_FCS", origin=[0, 0, 0], reference_cs=fcs6.name, mode="view", view="iso"
)
```

## Create object coordinate system

Create object coordinate system with origin on face

```
obj_cs = hfss.modeler.create_object_coordinate_system(assignment=box, origin=box.
 ↪faces[0], x_axis=box.edges[0],
 y_axis=[0, 0, 0], name="box_obj_cs"
 ↪)
obj_cs.rename("new_obj_cs")
```

True

## Create object coordinate system

Create object coordinate system with origin on edge

```
obj_cs_1 = hfss.modeler.create_object_coordinate_system(assignment=box.name, origin=box.
edges[0], x_axis=[1, 0, 0],
y_axis=[0, 1, 0], name="obj_cs_1")
obj_cs_1.set_as_working_cs()
```

```
True
```

## Create object coordinate system

Create object coordinate system with origin specified on point

```
obj_cs_2 = hfss.modeler.create_object_coordinate_system(assignment=box.name, origin=[0,
0.8, 0], x_axis=[1, 0, 0],
y_axis=[0, 1, 0], name="obj_cs_2")
new_obj_cs_2 = hfss.modeler.duplicate_coordinate_system_to_global(obj_cs_2)
obj_cs_2.delete()
```

```
True
```

## Create object coordinate system

Create object coordinate system with origin on vertex

```
obj_cs_3 = hfss.modeler.create_object_coordinate_system(assignment=box.name, origin=box.
vertices[1],
x_axis=box.faces[2], y_axis=box.
faces[4], name="obj_cs_3")
obj_cs_3.props["MoveToEnd"] = False
obj_cs_3.update()
```

```
True
```

## Get all coordinate systems

Get all coordinate systems.

```
css = hfss.modeler.coordinate_systems
names = [i.name for i in css]
print(names)
```

```
['obj_cs_3', 'obj_cs_1', 'new_obj_cs', 'CS_FCS', 'Face_CS_MVHJ1M', 'Face_CS_EINHI2',
'Face_CS_L468T2', 'Face_CS_MW9HWL', 'FCS2', 'FCS1', 'CS5', 'CS4', 'CS3', 'CS2']
```

## Select coordinate system

Select an existing coordinate system.

```
css = hfss.modeler.coordinate_systems
cs_selected = css[0]
cs_selected.delete()
```

```
True
```

## Get point coordinate under another coordinate system

Get a point coordinate under another coordinate system. A point coordinate can be translated in respect to any coordinate system.

```
hfss.modeler.create_box([-10, -10, -10], [20, 20, 20], "Box1")
p = hfss.modeler["Box1"].faces[0].vertices[0].position
print("Global: ", p)
p2 = hfss.modeler.global_to_cs(p, "CS5")
print("CS5 : ", p2)
```

```
Global: [-10.0, -10.0, 10.0]
CS5 : [-10.0, 13.8330960296045, 2.940315329304]
```

## Close AEDT

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

```
d.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 54.229 seconds)

## General: optimetrics setup

This example shows how you can use PyAEDT to create a project in HFSS and create all optimetrics setups.

### Perform required imports

Perform required imports.

```
import pyaedt
import os
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Initialize object and create variables

Initialize the `Hfss` object and create two needed design variables, `w1` and `w2`.

```
hfss = pyaedt.Hfss(specified_version=aedt_version, new_desktop_session=True, non_
graphical=non_graphical)
hfss["w1"] = "1mm"
hfss["w2"] = "100mm"
```

## Create waveguide with sheets on it

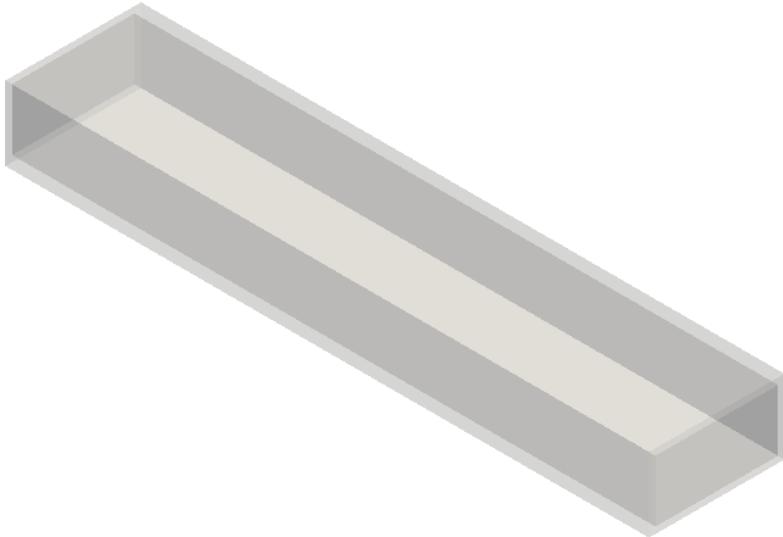
Create one of the standard waveguide structures and parametrize it. You can also create rectangles of waveguide openings and assign ports later.

```
wg1, p1, p2 = hfss.modeler.create_waveguide(
 [0, 0, 0],
 hfss.AXIS.Y,
 "WG17",
 wg_thickness="w1",
 wg_length="w2",
 create_sheets_on_openings=True,
)

model = hfss.plot(show=False)

model.show_grid = False
model.plot(os.path.join(hfss.working_directory, "Image.jpg"))
```

```
WG17_AMFP MJ
NewObject_HXL30Z_ObjectFromFace1
NewObject_HXL30Z_ObjectFromFace2
```



True

### Create wave ports on sheets

Create two wave ports on the sheets.

```
hfss.wave_port(p1, integration_line=hfss.AxisDir.ZPos, name="1")
hfss.wave_port(p2, integration_line=hfss.AxisDir.ZPos, name="2")
```

False

### Create setup and frequency sweep

Create a setup and a frequency sweep to use as the base for optimetrics setups.

```
setup = hfss.create_setup()
hfss.create_linear_step_sweep(
 setup_name=setup.name,
 unit="GHz",
 start_frequency=1,
```

(continues on next page)

(continued from previous page)

```
 stop_frequency=5,
 step_size=.1,
 sweep_name="Sweep1",
 save_fields=True
)
```

```
False
```

## Optimetrics analysis

### Create parametrics analysis

Create a simple optimetrics parametrics analysis with output calculations.

```
sweep = hfss.parametrics.add("w2", 90, 200, 5)
sweep.add_variation("w1", .1, 2, 10)
sweep.add_calculation(calculation="dB(S(1,1))", ranges={"Freq": "2.5GHz"})
sweep.add_calculation(calculation="dB(S(1,1))", ranges={"Freq": "2.6GHz"})
```

```
False
```

### Create sensitivity analysis

Create an optimetrics sensitivity analysis with output calculations.

```
sweep2 = hfss.optimizations.add(calculation="dB(S(1,1))", ranges={"Freq": "2.5GHz"},
 optim_type="Sensitivity")
sweep2.add_variation("w1", .1, 3, .5)
sweep2.add_calculation(calculation="dB(S(1,1))", ranges={"Freq": "2.6GHz"})
```

```
False
```

### Create optimization based on goals and calculations

Create an optimization analysis based on goals and calculations.

```
sweep3 = hfss.optimizations.add(calculation="dB(S(1,1))", ranges={"Freq": "2.5GHz"})
sweep3.add_variation("w1", .1, 3, .5)
sweep3.add_goal(calculation="dB(S(1,1))", ranges={"Freq": "2.6GHz"})
sweep3.add_goal(calculation="dB(S(1,1))", ranges={"Freq": ("2.6GHz", "5GHz")})
sweep3.add_goal(
 calculation="dB(S(1,1))",
 ranges={"Freq": ("2.6GHz", "5GHz")},
 condition="Maximize",
)
```

```
False
```

### Create DX optimization based on a goal and calculation

Create a DX (DesignXplorer) optimization based on a goal and a calculation.

```
sweep4 = hfss.optimizations.add(calculation="dB(S(1,1))", ranges={"Freq": "2.5GHz"},
 optim_type="DesignExplorer")
sweep4.add_goal(calculation="dB(S(1,1))", ranges={"Freq": "2.6GHz"})
```

```
False
```

### Create DOE based on a goal and calculation

Create a DOE (Design of Experiments) based on a goal and a calculation.

```
sweep5 = hfss.optimizations.add(calculation="dB(S(1,1))", ranges={"Freq": "2.5GHz"},
 optim_type="DXDOE")
```

### Create DOE based on a goal and calculation

Create a DOE based on a goal and a calculation.

```
region = hfss.modeler.create_region()
hfss.assign_radiation_boundary_to_objects(region)
hfss.insert_infinite_sphere(name="Infinite_1")
sweep6 = hfss.optimizations.add(
 calculation="RealizedGainTotal",
 solution=hfss.nominal_adaptive,
 ranges={"Freq": "5GHz", "Theta": ["0deg", "10deg", "20deg"], "Phi": "0deg"},
 context="Infinite_1",
)
```

### Close AEDT

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

```
hfss.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 59.998 seconds)

## General: polyline creation

This example shows how you can use PyAEDT to create and manipulate polylines.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

### Create Maxwell 3D object

Create a `pyaedt.maxwell.Maxwell3d` object and set the unit type to "mm".

```
M3D = pyaedt.Maxwell3d(solution_type="Transient", designname="test_polyline_3D",
 ↪specified_version=aedt_version,
 new_desktop_session=True, non_graphical=non_graphical,)
M3D.modeler.model_units = "mm"
prim3D = M3D.modeler
```

### Define variables

Define two design variables as parameters for the polyline objects.

```
M3D["p1"] = "100mm"
M3D["p2"] = "71mm"
```

## Input data

Input data. All data for the polyline functions can be entered as either floating point values or strings. Floating point values are assumed to be in model units (M3D.modeler.model\_units).

```
test_points = [[{"0mm", "p1", "0mm"}, ["-p1", "0mm", "0mm"], ["-p1/2", "-p1/2", "0mm"], [←"0mm", "0mm", "0mm"]]
```

## Polyline primitives

The following examples are for creating polyline primitives.

```
Create line primitive
~~~~~
Create a line primitive. The basic polyline command takes a list of positions
([X, Y, Z] coordinates) and creates a polyline object with one or more
segments. The supported segment types are ``Line``, ``Arc`` (3 points),
``AngularArc`` (center-point + angle), and ``Spline``.

P = prim3D.create_polyline(points=test_points[0:2], name="PL01_line")

print("Created Polyline with name: {}".format(prim3D.objects[P.id].name))
print("Segment types : {}".format([s.type for s in P.segment_types]))
print("primitive id = {}".format(P.id))
```

```
Created Polyline with name: PL01_line
Segment types : ['Line']
primitive id = 6
```

## Create arc primitive

Create an arc primitive. The parameter `position_list` must contain at least three position values. The first three position values are used.

```
P = prim3D.create_polyline(points=test_points[0:3], segment_type="Arc", name="PL02_arc")

print("Created object with id {} and name {}".format(P.id, prim3D.objects[P.id].name))
```

```
Created object with id 11 and name PL02_arc.
```

## Create spline primitive

Create a spline primitive. Defining the segment using a `PolylineSegment` object allows you to provide additional input parameters for the spine, such as the number of points (in this case 4). The parameter `position_list` must contain at least four position values.

```
P = prim3D.create_polyline(points=test_points, segment_type=prim3D.polyline_segment(
 ←"Spline", num_points=4,
 name="PL03_spline_4pt")
```

## Create center-point arc primitive

Create a center-point arc primitive. A center-point arc segment is defined by a starting point, a center point, and an angle of rotation around the center point. The rotation occurs in a plane parallel to the XY, YZ, or ZX plane of the active coordinate system. The starting point and the center point must therefore have one coordinate value (X, Y, or Z) with the same value.

Here `start_point` and `center_point` have a common Z coordinate, "0mm". The curve is therefore rotated in the XY plane with Z = "0mm".

```
start_point = [100, 100, 0]
center_point = [0, 0, 0]
P = prim3D.create_polyline(points=[start_point],
 segment_type=prim3D.polyline_segment("AngularArc", arc_
 ↵center=center_point,
 ↵arc_angle="30deg"), name=
 ↵"PL04_center_point_arc")
```

Here `start_point` and `center_point` have the same values for the Y and Z coordinates, so the plane of rotation could be either XY or ZX. For these special cases when the rotation plane is ambiguous, you can specify the plane explicitly.

```
start_point = [100, 0, 0]
center_point = [0, 0, 0]
P = prim3D.create_polyline(points=[start_point],
 segment_type=prim3D.polyline_segment("AngularArc", arc_
 ↵center=center_point,
 ↵arc_angle="30deg", arc_
 ↵plane="XY"),
 name="PL04_center_point_arc_rot_XY")
P = prim3D.create_polyline(points=[start_point],
 segment_type=prim3D.polyline_segment("AngularArc", arc_
 ↵center=center_point,
 ↵arc_angle="30deg", arc_
 ↵plane="ZX"),
 name="PL04_center_point_arc_rot_ZX")
```

## Compound polylines

You can use a list of points in a single command to create a multi-segment polyline.

By default, if no specification of the type of segments is given, all points are connected by line segments.

```
P = prim3D.create_polyline(points=test_points, name="PL06_segmented_compound_line")
```

You can specify the segment type with the parameter `segment_type`. In this case, you must specify that the four input points in `position_list` are to be connected as a line segment followed by a 3-point arc segment.

```
P = prim3D.create_polyline(points=test_points, segment_type=["Line", "Arc"], name="PL05_
 ↵compound_line_arc")
```

The parameter `close_surface` ensures that the polyline starting point and ending point are the same. If necessary, you can add an additional line segment to achieve this.

```
P = prim3D.create_polyline(points=test_points, close_surface=True, name="PL07_segmented_
↪compound_line_closed")
```

The parameter `cover_surface=True` also performs the modeler command `cover_surface`. Note that specifying `cover_surface=True` automatically results in the polyline being closed.

```
P = prim3D.create_polyline(points=test_points, cover_surface=True, name="SPL01_segmented_
↪compound_line")
```

## Compound lines

The following examples are for inserting compound lines.

### Insert line segment

Insert a line segment starting at vertex 1 `["100mm", "0mm", "0mm"]` of an existing polyline and ending at some new point `["90mm", "20mm", "0mm"]`. By numerical comparison of the starting point with the existing vertices of the original polyline object, it is determined automatically that the segment is inserted after the first segment of the original polyline.

```
P = prim3D.create_polyline(points=test_points, close_surface=True, name="PL08_segmented_
↪compound_insert_segment")

p2 = P.points[1]
insert_point = ["-100mm", "20mm", "0mm"]

P.insert_segment(points=[insert_point, p2])
```

True

### Insert compound line with insert curve

Insert a compound line starting a line segment at vertex 1 `["100mm", "0mm", "0mm"]` of an existing polyline and end at some new point `["90mm", "20mm", "0mm"]`. By numerical comparison of the starting point, it is determined automatically that the segment is inserted after the first segment of the original polyline.

```
P = prim3D.create_polyline(points=test_points, close_surface=False, name="PL08_segmented_
↪compound_insert_arc")

start_point = P.vertex_positions[1]
insert_point1 = ["90mm", "20mm", "0mm"]
insert_point2 = [40, 40, 0]

P.insert_segment(points=[start_point, insert_point1, insert_point2], segment="Arc")
```

True

## Insert compound line at end of a center-point arc

Insert a compound line at the end of a center-point arc (`type="AngularArc"`). This is a special case.

Step 1: Draw a center-point arc.

```
start_point = [2200.0, 0.0, 1200.0]
arc_center_1 = [1400, 0, 800]
arc_angle_1 = "43.47deg"

P = prim3D.create_polyline(points=[start_point],
 segment_type=prim3D.polyline_segment(type="AngularArc", arc_
 ↪angle=arc_angle_1,
 ↪name="First_Arc"),
 arc_center=arc_center_1),
```

Step 2: Insert a line segment at the end of the arc with a specified end point.

```
start_of_line_segment = P.end_point
end_of_line_segment = [3600, 200, 30]

P.insert_segment(points=[start_of_line_segment, end_of_line_segment])
```

True

Step 3: Append a center-point arc segment to the line object.

```
arc_angle_2 = "39.716deg"
arc_center_2 = [3400, 200, 3800]

P.insert_segment(points=[end_of_line_segment],
 segment=prim3D.polyline_segment(type="AngularArc", arc_center=arc_
 ↪center_2, arc_angle=arc_angle_2))
```

True

You can use the compound polyline definition to complete all three steps in a single step.

```
prim3D.create_polyline(points=[start_point, end_of_line_segment], segment_type=[

 prim3D.polyline_segment(type="AngularArc", arc_angle="43.47deg", arc_center=arc_
 ↪center_1),
 prim3D.polyline_segment(type="Line"),
 prim3D.polyline_segment(type="AngularArc", arc_angle=arc_angle_2, arc_center=arc_
 ↪center_2),
], name="Compound_Polyline_One_Command")
```

```
<pyaedt.modeler.cad.polylines.Polyline object at 0x00000258F252A110>
```

## Insert two 3-point arcs forming a circle and covered

Insert two 3-point arcs forming a circle and covered. Note that the last point of the second arc segment is not defined in the position list.

```
P = prim3D.create_polyline(
 points=[[34.1004, 14.1248, 0], [27.646, 16.7984, 0], [24.9725, 10.3439, 0], [31.4269,
 ↪ 7.6704, 0]],
 segment_type=["Arc", "Arc"], cover_surface=True, close_surface=True, name="Rotor_
 ↪ Subtract_25_0", material="vacuum")
```

Here is an example of a complex polyline where the number of points is insufficient to populate the requested segments. This results in an `IndexError` that PyAEDT catches silently. The return value of the command is `False`, which can be caught at the app level. While this example might not be so useful in a Jupyter Notebook, it is important for unit tests.

```
MDL_points = [
 ["67.1332mm", "2.9901mm", "0mm"],
 ["65.9357mm", "2.9116mm", "0mm"],
 ["65.9839mm", "1.4562mm", "0mm"],
 ["66mm", "0mm", "0mm"],
 ["99mm", "0mm", "0mm"],
 ["98.788mm", "6.4749mm", "0mm"],
 ["98.153mm", "12.9221mm", "0mm"],
 ["97.0977mm", "19.3139mm", "0mm"],
]

MDL_segments = ["Line", "Arc", "Line", "Arc", "Line"]
return_value = prim3D.create_polyline(MDL_points, segment_type=MDL_segments, name="MDL_
 ↪ Polyline")
assert return_value # triggers an error at the application error
```

Here is an example that provides more points than the segment list requires. This is valid usage. The remaining points are ignored.

```
MDL_segments = ["Line", "Arc", "Line", "Arc"]

P = prim3D.create_polyline(MDL_points, segment_type=MDL_segments, name="MDL_Polyline")
```

## Save project

Save the project.

```
project_dir = r"C:\temp"
project_name = "Polylines"
project_file = os.path.join(project_dir, project_name + ".aedt")

M3D.save_project(project_file)

M3D.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 59.169 seconds)

### 3.12.4 HFSS examples

These examples use PyAEDT to show some end-to-end workflows for HFSS 3D. This includes model generation, setup, meshing, and postprocessing.

#### HFSS: component antenna array

This example shows how you can use PyAEDT to create an example using a 3D component file. It sets up the analysis, solves it, and uses postprocessing functions to create plots using Matplotlib and PyVista without opening the HFSS user interface. This examples runs only on Windows using CPython.

##### Perform required imports

Perform required imports.

```
import os
import pyaedt
from pyaedt.modules.solutions import FfdSolutionData
```

##### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

##### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

##### Download 3D component

Download the 3D component that is needed to run the example.

```
example_path = pyaedt.downloads.download_3dcomponent()
```

## Launch HFSS and save project

Launch HFSS and save the project.

```
project_name = pyaedt.generate_unique_project_name(project_name="array")
hfss = pyaedt.Hfss(projectname=project_name,
 specified_version=aedt_version,
 designname="Array_Simple",
 non_graphical=non_graphical,
 new_desktop_session=True)

print("Project name " + project_name)
```

```
Project name D:\Temp\pyaedt_prj_67S\array.aedt
```

## Read array definition from JSON file

Read the array definition from a JSON file. A JSON file can contain all information needed to import and set up a full array in HFSS.

If a 3D component is not available in the design, it is loaded into the dictionary from the path that you specify. The following code edits the dictionary to point to the location of the A3DCOMP file.

```
dict_in = pyaedt.general_methods.read_json(os.path.join(example_path, "array_simple.json"))
dict_in["Circ_Patch_5GHz1"] = os.path.join(example_path, "Circ_Patch_5GHz.a3dcomp")
dict_in["cells"][(3, 3)] = {"name": "Circ_Patch_5GHz1"}
array = hfss.add_3d_component_array_from_json(dict_in)
```

## Modify cells

Make center element passive and rotate corner elements.

```
array.cells[1][1].is_active = False
array.cells[0][0].rotation = 90
array.cells[0][2].rotation = 90
array.cells[2][0].rotation = 90
array.cells[2][2].rotation = 90
```

## Set up simulation

Set up a simulation and analyze it.

```
setup = hfss.create_setup()
setup.props["Frequency"] = "5GHz"
setup.props["MaximumPasses"] = 3

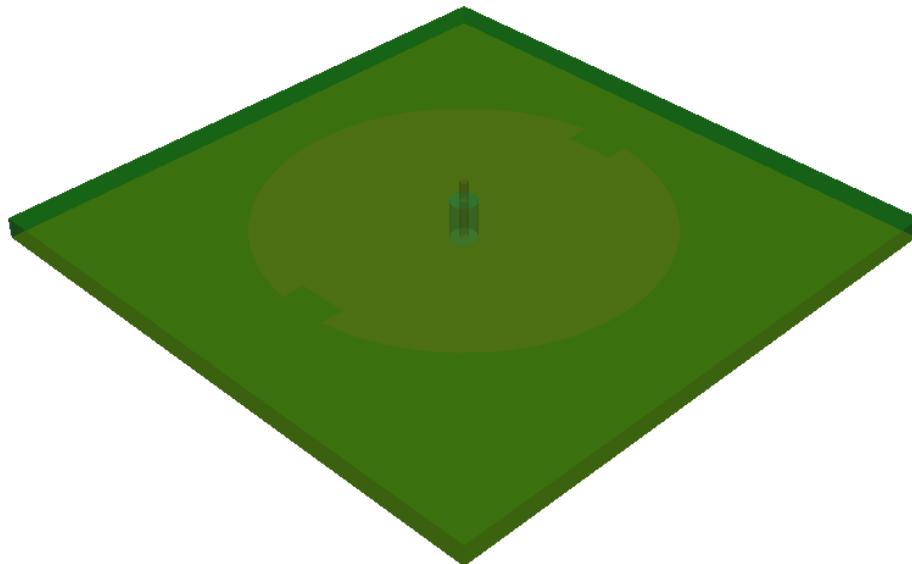
hfss.analyze(cores=4)
```

```
True
```

### Get far field data

Get far field data. After the simulation completes, the far field data is generated port by port and stored in a data class.

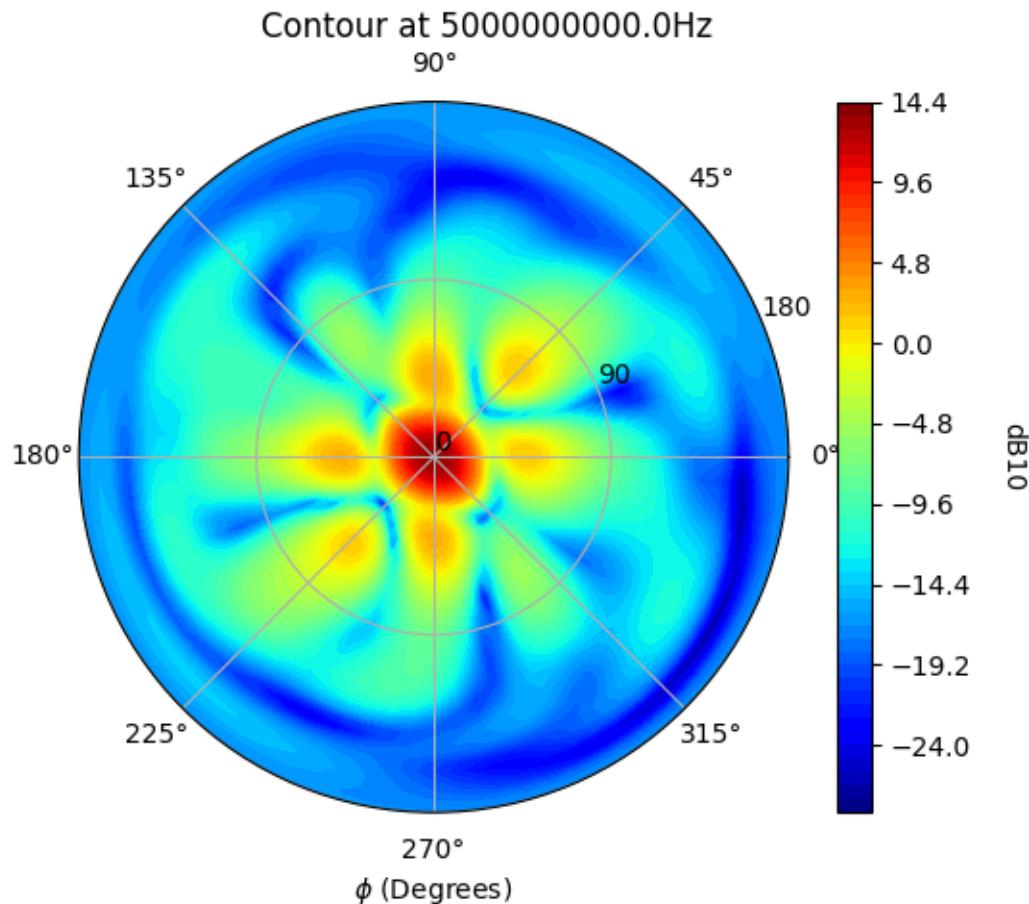
```
ffdata = hfss.get_antenna_ffd_solution_data(frequencies=[5e9], setup=hfss.nominal_
 ↪adaptive, sphere="Infinite Sphere1")
```



### Generate contour plot

Generate a contour plot. You can define the Theta scan and Phi scan.

```
ffdata.plot_farfield_contour(quantity='RealizedGain', title='Contour at {}Hz'.
 ↪format(ffdata.frequency))
```



<Figure size 640x480 with 2 Axes>

Release AEDT

Release AEDT. Far field post-processing can be performed without AEDT because the data is stored.

```
eep_file = ffdata.eep_files
frequencies = ffdata.frequencies
working_directory = hfss.working_directory

hfss.release_desktop()
```

True

## Load far field data

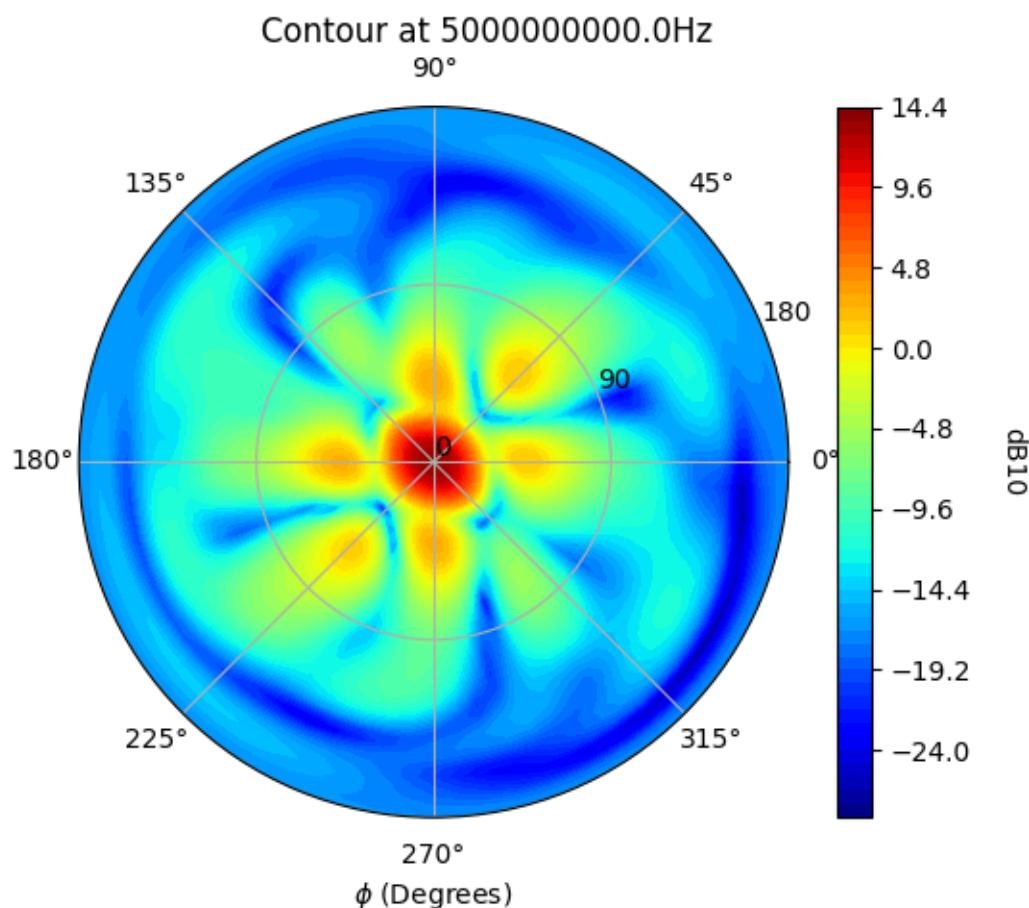
Load far field data stored.

```
ffdata = FfdSolutionData(frequencies=frequencies[0], eep_files=eep_file[0])
```

## Generate contour plot

Generate a contour plot. You can define the Theta scan and Phi scan.

```
ffdata.plot_farfield_contour(quantity='RealizedGain', title='Contour at {}Hz'.
 .format(ffdata.frequency))
```



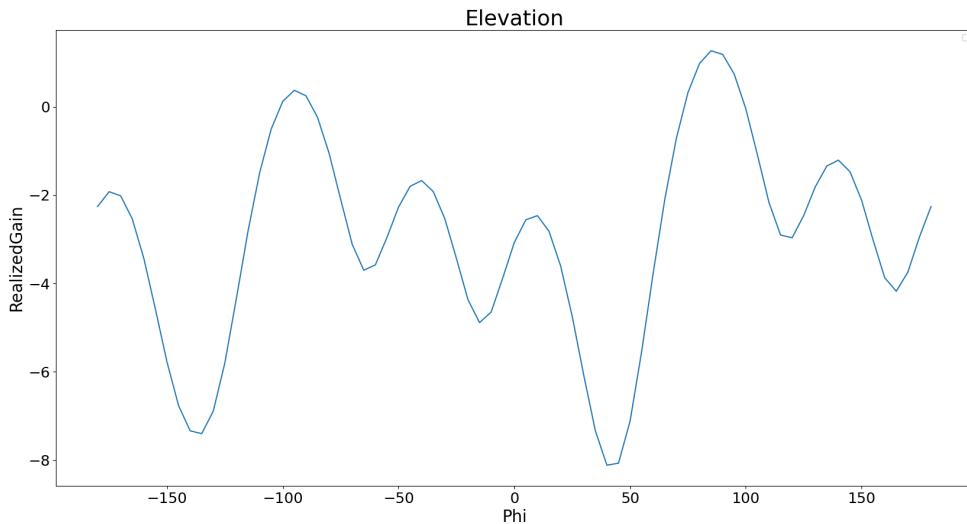
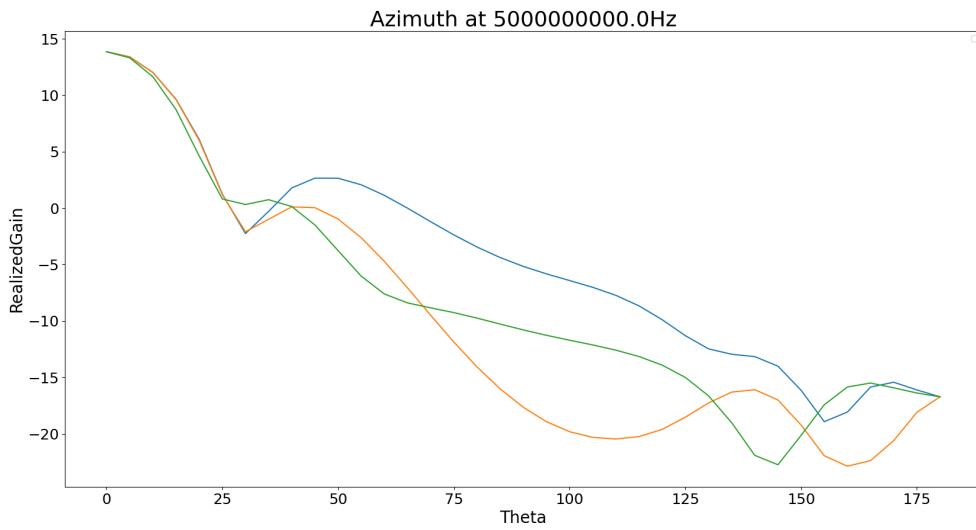
```
<Figure size 640x480 with 2 Axes>
```

## Generate 2D cutout plots

Generate 2D cutout plots. You can define the Theta scan and Phi scan.

```
ffdata.plot_2d_cut(quantity='RealizedGain', primary_sweep='theta', secondary_sweep_
˓→value=[-180, -75, 75],
 title='Azimuth at {}Hz'.format(ffdata.frequency), quantity_format=
˓→"dB10")

ffdata.plot_2d_cut(quantity='RealizedGain', primary_sweep="phi", secondary_sweep_
˓→value=30, title='Elevation',
 quantity_format="dB10")
```



No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.

(continues on next page)

(continued from previous page)

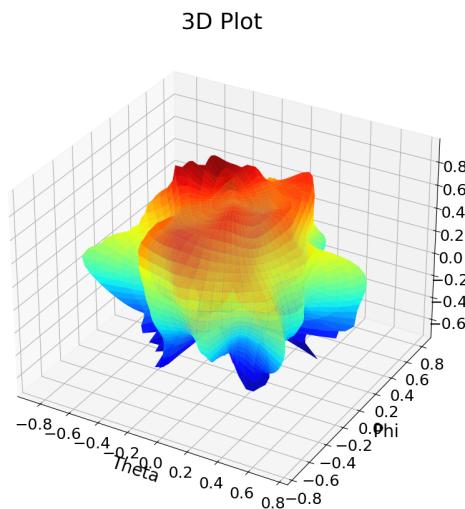
No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Generate 3D polar plots in Matplotlib

Generate 3D polar plots in Matplotlib. You can define the Theta scan and Phi scan.

```
ffdata.polar_plot_3d(quantity='RealizedGain')
```



<Figure size 2000x1000 with 1 Axes>

**Total running time of the script:** (1 minutes 50.802 seconds)

## Create a 3D Component and reuse it

Summary of the workflow 1. Create an antenna using PyAEDT and HFSS 3D Modeler (same can be done with EDB and HFSS 3D Layout) 2. Store the object as a 3D Component on the disk 3. Reuse the 3D component in another project 4. Parametrize and optimize target design

### Perform required imports

Perform required imports.

```
import os
import tempfile
from pyaedt import Hfss
from pyaedt.generic.general_methods import generate_unique_name
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Launch HFSS

PyAEDT can initialize a new session of Electronics Desktop or connect to an existing one. Once Desktop is connected, a new HFSS session is started and a design is created.

```
hfss = Hfss(specified_version=aedt_version, new_desktop_session=True, close_on_exit=True)
```

### Variables

PyAEDT can create and store all variables available in AEDT (Design, Project, Post Processing)

```
hfss["thick"] = "0.1mm"
hfss["width"] = "1mm"
```

### Modeler

PyAEDT supports all modeler functionalities available in the Desktop. Objects can be created, deleted and modified using all available boolean operations. History is also fully accessible to PyAEDT.

```
substrate = hfss.modeler.create_box(["-width", "-width", "-thick"], ["2*width", "2*width", "thick"], name="sub", material="FR4_epoxy")

patch = hfss.modeler.create_rectangle("XY", [-width/2, -width/2, 0mm], [width, width], name="patch1")
```

(continues on next page)

(continued from previous page)

```
via1 = hfss.modeler.create_cylinder(2, ["-width/8", "-width/4", "-thick"], "0.01mm",
→ "thick", name="via_inner",
 material="copper")

via_outer = hfss.modeler.create_cylinder(2, ["-width/8", "-width/4", "-thick"], "0.025mm
→ ", "thick", name="via_teflon",
 material="Teflon_based")
```

## Boundaries

Most of HFSS boundaries and excitations are already available in PyAEDT. User can assign easily a boundary to a face or to an object by taking benefits of Object-Oriented Programming (OOP) available in PyAEDT.

```
hfss.assign_perfecte_to_sheets(patch)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F6A4F1F0>
```

## Advanced Modeler functions

Thanks to Python capabilities a lot of additional functionalities have been added to the Modeler of PyAEDT. in this example there is a property to retrieve automatically top and bottom faces of an objects.

```
side_face = [i for i in via_outer.faces if i.id not in [via_outer.top_face_z.id, via_
→ outer.bottom_face_z.id]]

hfss.assign_perfecte_to_sheets(side_face)
hfss.assign_perfecte_to_sheets(substrate.bottom_face_z)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F6A4EEF0>
```

## Create Wave Port

Wave port can be assigned to a sheet or to a face of an object.

```
hfss.wave_port(via_outer.bottom_face_z, name="P1")
```

```
False
```

## Create 3D Component

Once the model is ready a 3D Component can be created. Multiple options are available to partially select objects, cs, boundaries and mesh operations. Furthermore, encrypted 3d comp can be created too.

```
component_path = os.path.join(tempfile.gettempdir(), generate_unique_name("component_test"
 ↪")+" .aedbcomp")
hfss.modeler.create_3dcomponent(component_path, "patch_antenna")
```

```
True
```

## Multiple project management

PyAEDT allows to control multiple projects, design and solution type at the same time.

```
hfss2 = Hfss(projectname="new_project", designname="new_design")
```

## Insert of 3d component

The 3d component can be inserted without any additional info. All needed info will be read from the file itself.

```
hfss2.modeler.insert_3d_component(component_path)
```

```
<pyaedt.modeler.cad.components_3d.UserDefinedComponent object at 0x00000258F8AA0760>
```

## 3D Component Parameters

All 3d Component parameters are available and can be parametrized.

```
hfss2.modeler.user_defined_components["patch_antenna1"].parameters
hfss2["p_thick"] = "1mm"
hfss2.modeler.user_defined_components["patch_antenna1"].parameters["thick"] = "p_thick"
```

## Multiple 3d Components

There is no limit to the number of 3D components that can be added on the same design. They can be the same or linked to different files.

```
hfss2.modeler.create_coordinate_system(origin=[20, 20, 10], name="Second_antenna")
ant2 = hfss2.modeler.insert_3d_component(component_path, coordinate_system="Second_
 ↪antenna")
```

## Move components

The component can be moved by changing its position or moving the relative coordinate system.

```
hfss2.modeler.coordinate_systems[0].origin = [10, 10, 3]
```

## Boundaries

Most of HFSS boundaries and excitations are already available in PyAEDT. User can assign easily a boundary to a face or to an object by taking benefits of

```
hfss2.modeler.create_air_region(30, 30, 30, 30, 30, 30)
hfss2.assign_radiation_boundary_to_faces(hfss2.modeler["Region"].faces)

Create Setup and Optimetrics
Once project is ready to be solved, a setup and parametrics analysis can be created
with PyAEDT.
All setup parameters can be edited.

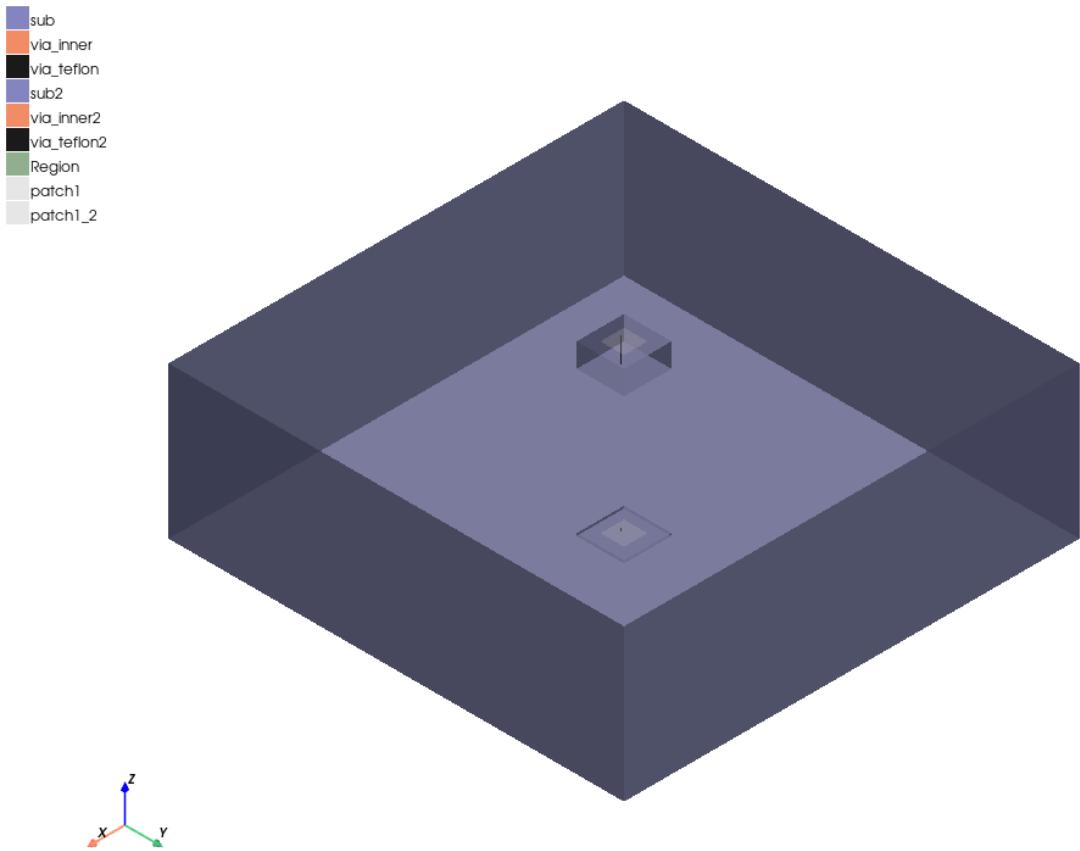
setup1 = hfss2.create_setup()

optim = hfss2.parametrics.add("p_thick", "0.2mm", "1.5mm", step=14)
```

## Save project

Save the project.

```
hfss2.modeler.fit_all()
hfss2.plot(show=False, export_path=os.path.join(hfss.working_directory, "Image.jpg"),
plot_air_objects=True)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258F8AA1450>
```

## Close AEDT

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

```
hfss2.save_project(os.path.join(tempfile.gettempdir(), generate_unique_name("parametrized"
 ↪")+" .aedt"))
hfss2.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 0.151 seconds)

## HFSS: flex cable CPWG

This example shows how you can use PyAEDT to create a flex cable CPWG (coplanar waveguide with ground).

### Perform required imports

Perform required imports.

```
import os
from math import radians, sin, cos, sqrt
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

### Launch AEDT

Launch AEDT 2023 R2 in graphical mode.

```
hfss = pyaedt.Hfss(specified_version=aedt_version,
 solution_type="DrivenTerminal",
 new_desktop_session=True,
 non_graphical=non_graphical)
hfss.change_material_override(True)
hfss.change_automatically_use_causal_materials(True)
hfss.create_open_region("100GHz")
hfss.modeler.model_units = "mil"
hfss.mesh.assign_initial_mesh_from_slider(applycurvilinear=True)
```

```
True
```

## Create variables

Create input variables for creating the flex cable CPWG.

```
total_length = 300
theta = 120
r = 100
width = 3
height = 0.1
spacing = 1.53
gnd_width = 10
gnd_thickness = 2

xt = (total_length - r * radians(theta)) / 2
```

## Create bend

Create the bend. The `create_bending` method creates a list of points for the bend based on the curvature radius and extension.

```
def create_bending(radius, extension=0):
 position_list = [(-xt, 0, -radius), (0, 0, -radius)]

 for i in [radians(i) for i in range(theta)] + [radians(theta + 0.00000001)]:
 position_list.append((radius * sin(i), 0, -radius * cos(i)))

 x1, y1, z1 = position_list[-1]
 x0, y0, z0 = position_list[-2]

 scale = (xt + extension) / sqrt((x1 - x0) ** 2 + (z1 - z0) ** 2)
 x, y, z = (x1 - x0) * scale + x0, 0, (z1 - z0) * scale + z0

 position_list[-1] = (x, y, z)
 return position_list
```

## Draw signal line

Draw a signal line to create a bent signal wire.

```
position_list = create_bending(r, 1)
line = hfss.modeler.create_polyline(points=position_list, material="copper", xsection_
 ↪type="Rectangle",
 xsection_width=height, xsection_height=width)
```

## Draw ground line

Draw a ground line to create two bent ground wires.

```
gnd_r = [(x, spacing + width / 2 + gnd_width / 2, z) for x, y, z in position_list]
gnd_l = [(x, -y, z) for x, y, z in gnd_r]

gnd_objs = []
for gnd in [gnd_r, gnd_l]:
 x = hfss.modeler.create_polyline(points=gnd, material="copper", xsection_type=
 "Rectangle", xsection_width=height,
 xsection_height=gnd_width)
 x.color = (255, 0, 0)
 gnd_objs.append(x)
```

## Draw dielectric

Draw a dielectric to create a dielectric cable.

```
position_list = create_bending(r + (height + gnd_thickness) / 2)

fr4 = hfss.modeler.create_polyline(points=position_list, material="FR4_epoxy", xsection_
 type="Rectangle",
 xsection_width=gnd_thickness, xsection_height=width +
 2 * spacing + 2 * gnd_width)
```

## Create bottom metals

Create the bottom metals.

```
position_list = create_bending(r + height + gnd_thickness, 1)

bot = hfss.modeler.create_polyline(points=position_list, material="copper", xsection_
 type="Rectangle",
 xsection_width=height, xsection_height=width + 2 *
 spacing + 2 * gnd_width)
```

## Create port interfaces

Create port interfaces (PEC enclosures).

```
port_faces = []
for face, blockname in zip([fr4.top_face_z, fr4.bottom_face_x], ["b1", "b2"]):
 xc, yc, zc = face.center
 positions = [i.position for i in face.vertices]

 port_sheet_list = [((x - xc) * 10 + xc, (y - yc) + yc, (z - zc) * 10 + zc) for x, y, z
 in positions]
 s = hfss.modeler.create_polyline(port_sheet_list, cover_surface=True, close_
```

(continues on next page)

(continued from previous page)

```

 ↵surface=True)
 center = [round(i, 6) for i in s.faces[0].center]

 port_block = hfss.modeler.thicken_sheet(s.name, -5)
 port_block.name = blockname
 for f in port_block.faces:

 if [round(i, 6) for i in f.center] == center:
 port_faces.append(f)

 port_block.material_name = "PEC"

 for i in [line, bot] + gnd_objs:
 i.subtract([port_block], True)

print(port_faces)

```

[7391]  
[7391, 7586]

## Create boundary condition

Creates a Perfect E boundary condition.

```

boundary = []
for face in [fr4.top_face_y, fr4.bottom_face_y]:
 s = hfss.modeler.create_object_from_face(face)
 boundary.append(s)
hfss.assign_perfecte_to_sheets(s)

```

## Create ports

Creates ports.

```

for s, port_name in zip(port_faces, ["1", "2"]):
 reference = [i.name for i in gnd_objs + boundary + [bot]] + ["b1", "b2"]

hfss.wave_port(s.id, reference=reference, name=port_name)

```

## Create setup and sweep

Create the setup and sweep.

```

setup = hfss.create_setup("setup1")
setup["Frequency"] = "2GHz"
setup.props["MaximumPasses"] = 10
setup.props["MinimumConvergedPasses"] = 2
hfss.create_linear_count_sweep(setup="setup1", units="GHz", start_frequency=1e-1, stop_

```

(continues on next page)

(continued from previous page)

```
 ↵frequency=4,
 num_of_freq_points=101, name="sweep1", save_fields=False, ↵
 ↵sweep_type="Interpolating")
```

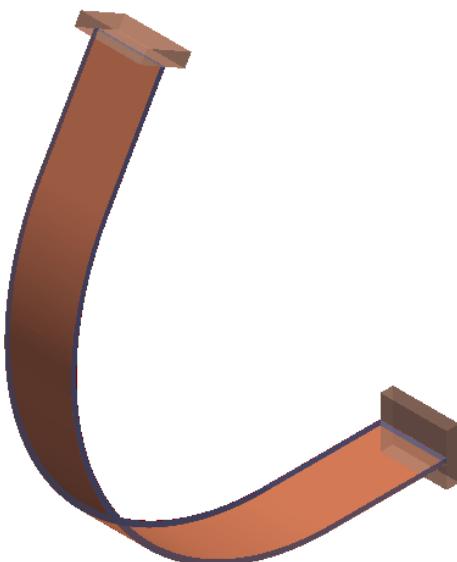
```
<pyaedt.modules.SolveSweeps.SweepHFSS object at 0x00000258F2A186D0>
```

## Plot model

Plot the model.

```
my_plot = hfss.plot(show=False, plot_air_objects=False)
my_plot.show_axes = False
my_plot.show_grid = False
my_plot.plot(
 os.path.join(hfss.working_directory, "Image.jpg"),
)
```

NewObject\_HTNQPM  
NewObject\_J61UIM  
NewObject\_KGVC3Y  
NewObject\_ONX0RV  
NewObject\_3XCBSD  
b1  
b2  
NewObject\_ONX0RV\_ObjectFromFace1  
NewObject\_ONX0RV\_ObjectFromFace2



```
True
```

## Analyze and release

Uncomment the `hfss.analyze` command if you want to analyze the model and release AEDT.

```
hfss.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 8.152 seconds)

## HFSS: choke

This example shows how you can use PyAEDT to create a choke setup in HFSS.

### Perform required imports

Perform required imports.

```
import json
import os
import pyaedt

project_name = pyaedt.generate_unique_project_name(rootname=r"C:\Data\Support\Test",
 ↴folder_name="choke", project_name="choke")
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

### Launch HFSS

Launches HFSS 2023 R2 in graphical mode.

```
hfss = pyaedt.Hfss(projectname=project_name,
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True,
 solution_type="Terminal")
```

## Rules and information of use

The dictionary values contain the different parameter values of the core and the windings that compose the choke. You must not change the main structure of the dictionary. The dictionary has many primary keys, including "Number of Windings", "Layer", and "Layer Type", that have dictionaries as values. The keys of these dictionaries are secondary keys of the dictionary values, such as "1", "2", "3", "4", and "Simple".

You must not modify the primary or secondary keys. You can modify only their values. You must not change the data types for these keys. For the dictionaries from "Number of Windings" through "Wire Section", values must be Boolean. Only one value per dictionary can be True. If all values are True, only the first one remains set to True. If all values are False, the first value is chosen as the correct one by default. For the dictionaries from "Core" through "Inner Winding", values must be strings, floats, or integers.

Descriptions follow for primary keys:

- "Number of Windings": Number of windings around the core
- "Layer": Number of layers of all windings
- "Layer Type": Whether layers of a winding are linked to each other
- "Similar Layer": Whether layers of a winding have the same number of turns and same spacing between turns
- "Mode": When there are only two windows, whether they are in common or differential mode
- "Wire Section": Type of wire section and number of segments
- "Core": Design of the core
- "Outer Winding": Design of the first layer or outer layer of a winding and the common parameters for all layers
- "Mid Winding": Turns and turns spacing (Coil Pit) for the second or mid layer if it is necessary
- "Inner Winding": Turns and turns spacing (Coil Pit) for the third or inner layer if it is necessary
- "Occupation(%)": An informative parameter that is useless to modify

The following parameter values work. You can modify them if you want.

```
values = {
 "Number of Windings": {"1": False, "2": True, "3": False, "4": False},
 "Layer": {"Simple": False, "Double": True, "Triple": False},
 "Layer Type": {"Separate": False, "Linked": True},
 "Similar Layer": {"Similar": False, "Different": True},
 "Mode": {"Differential": False, "Common": True},
 "Wire Section": {"None": False, "Hexagon": True, "Octagon": False, "Circle": False},
 "Core": {
 "Name": "Core",
 "Material": "ferrite",
 "Inner Radius": 20,
 "Outer Radius": 30,
 "Height": 10,
 "Chamfer": 0.8,
 },
 "Outer Winding": {
 "Name": "Winding",
 "Material": "copper",
 "Inner Radius": 20,
 "Outer Radius": 30,
 }
}
```

(continues on next page)

(continued from previous page)

```

 "Height": 10,
 "Wire Diameter": 1.5,
 "Turns": 20,
 "Coil Pit(deg)": 0.1,
 "Occupation)": 0,
},
"Mid Winding": {"Turns": 25, "Coil Pit(deg)": 0.1, "Occupation)": 0},
"Inner Winding": {"Turns": 4, "Coil Pit(deg)": 0.1, "Occupation)": 0},
}

```

## Convert dictionary to JSON file

Convert a dictionary to a JSON file. You must supply the path of the JSON file as an argument.

```

json_path = os.path.join(hfss.working_directory, "choke_example.json")

with open(json_path, "w") as outfile:
 json.dump(values, outfile)

```

## Verify parameters of JSON file

Verify parameters of the JSON file. The `check_choke_values` method takes the JSON file path as an argument and does the following:

- Checks if the JSON file is correctly written (as explained in the rules)
- Checks in equations on windings parameters to avoid having unintended intersections

```

dictionary_values = hfss.modeler.check_choke_values(json_path, create_another_file=False)
print(dictionary_values)

```

```

[True, {'Number of Windings': {'1': False, '2': True, '3': False, '4': False}, 'Layer': {
 'Simple': False, 'Double': True, 'Triple': False}, 'Layer Type': {'Separate': False,
 'Linked': True}, 'Similar Layer': {'Similar': False, 'Different': True}, 'Mode': {
 'Differential': False, 'Common': True}, 'Wire Section': {'None': False, 'Hexagon': True,
 'Octagon': False, 'Circle': False}, 'Core': {'Name': 'Core', 'Material': 'ferrite',
 'Inner Radius': 20, 'Outer Radius': 30, 'Height': 10, 'Chamfer': 0.8}, 'Outer Winding': {
 'Name': 'Winding', 'Material': 'copper', 'Inner Radius': 17.525, 'Outer Radius': 32.475,
 'Height': 14.95, 'Wire Diameter': 1.5, 'Turns': 20, 'Coil Pit(deg)': 2.699,
 'Occupation()': 59.97777777777774}, 'Mid Winding': {'Turns': 25, 'Coil Pit(deg)': 2.466,
 'Occupation()': 68.50000000000001}, 'Inner Winding': {'Turns': 4, 'Coil Pit(deg)': 0.1,
 'Occupation()': 0}}]

```

## Create choke

Create the choke. The `create_choke` method takes the JSON file path as an argument.

```
list_object = hfss.modeler.create_choke(json_path)
print(list_object)
core = list_object[1]
first_winding_list = list_object[2]
second_winding_list = list_object[3]
```

```
[<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258F6A2D360>, <pyaedt.modeler.
<cad.object3d.Object3d object at 0x00000258F6A2D060>, [<pyaedt.modeler.cad.polylines.
<Polyline object at 0x00000258F6A2E530>, [[-9.105371275769, 16.875213744732, -14.95], [-9.105371275769, 16.875213744732, 4.541726188958], [-9.714742044896, 18.004575927489, 5.825], [-15.133715602029, 25.37093294107, 5.825], [-15.791114589875, 26.473030144081, 4.541726188958], [-15.791114589875, 26.473030144081, -4.541726188958], [-15.133715602029, 25.37093294107, -5.825], [-11.2266867204, 17.102703663736, -5.825], [-10.522477108869, 16.029912678905, -4.541726188958], [-10.522477108869, 16.029912678905, 4.541726188958], [-11.2266867204, 17.102703663736, 5.825], [-17.258906742015, 23.975898821395, 5.825], [-18.008622682363, 25.017396628832, 4.541726188958], [-17.258906742015, 23.975898821395, -5.825], [-12.655496321042, 16.074183655653, -5.825], [-11.861662630843, 15.065908025475, 4.541726188958], [-12.655496321042, 16.074183655653, 5.825], [-19.256293433567, 22.403320053593, 5.825], [-20.0927745824, 23.376506038775, 4.541726188958], [-20.0927745824, 23.376506038775, -4.541726188958], [-19.256293433567, 22.403320053593, -5.825], [-13.990590325955, 14.926632227628, 5.825], [-21.111084768533, 20.664841787906, 5.825], [-22.028136874184, 21.562509382079, 4.541726188958], [-22.028136874184, 21.562509382079, -4.541726188958], [-21.111084768533, 20.664841787906, -5.825], [-15.222082190442, 13.668547147192, -5.825], [-14.267255815306, 12.811168428391, -4.541726188958], [-14.267255815306, 12.811168428391, 4.541726188958], [-15.222082190442, 13.668547147192, 5.825], [-22.809545775916, 18.773337681929, -5.825], [-16.340852557127, 12.309244697941, -5.825], [-15.315849747489, 11.537130124616, 4.541726188958], [-16.340852557127, 12.309244697941, 5.825], [-24.339099131124, 16.742814569501, 5.825], [-25.396374129191, 17.470111793866, 4.541726188958], [-25.396374129191, 17.470111793866, -4.541726188958], [-24.339099131124, 16.742814569501, -5.825], [-17.338616785984, 10.85879069125, -5.825], [-16.25102782092, 10.177657872206, 4.541726188958], [-17.338616785984, 10.85879069125, 5.825], [-25.688418292686, 14.588308738278, -5.825], [-26.804306857601, 15.222015598585, -4.541726188958], [-25.688418292686, 14.588308738278, 5.825], [-18.207986303167, 9.327925927625, -5.825], [-17.065864920372, 8.74281873995, -4.541726188958], [-17.065864920372, 8.74281873995, 5.825], [-26.847511376774, 12.325774584072, 5.825], [-28.013750208625, 12.861198398624, -4.541726188958], [-26.847511376774, 12.325774584072, -5.825], [-18.942523314366, 7.727986659686, -5.825], [-17.754327071178, 7.24323789818, -4.541726188958], [-17.754327071178, 7.24323789818, 4.541726188958], [-18.942523314366, 7.727986659686, 5.825], [-27.807795148393, 9.971966466468, 5.825], [-29.015748096995, 10.405142352304, 4.541726188958], [-29.
```

(continues on next page)

(continued from previous page)

↵ 015748096995, 10.405142352304, -4.541726188958], [-27.807795148393, 9.971966466468, -5.  
 ↵ 825], [-19.536788477508, 6.070820645736, -5.825], [-18.311316121598, 5.690019938005, -  
 ↵ 4.541726188958], [-18.311316121598, 5.690019938005, 4.541726188958], [-19.536788477508,  
 ↵ 6.070820645736, 5.825], [-28.562158581358, 7.544314640622, 5.825], [-29.802880598069,  
 ↵ 7.872034873984, 4.541726188958], [-29.802880598069, 7.872034873984, -4.541726188958],  
 ↵ [-28.562158581358, 7.544314640622, -5.825], [-19.986381181793, 4.368699415575, -5.825],  
 ↵ [-18.732707495294, 4.094666640371, -4.541726188958], [-18.732707495294, 4.  
 ↵ 094666640371, 4.541726188958], [-19.986381181793, 4.368699415575, 5.825], [-29.  
 ↵ 105015516366, 5.060796183947, 5.825], [-30.369318893333, 5.280633953898, 4.  
 ↵ 541726188958], [-30.369318893333, 5.280633953898, -4.541726188958], [-29.105015516366,  
 ↵ 5.060796183947, -5.825], [-20.287972134791, 2.634227398221, -5.825], [-19.015380734353,  
 ↵ 2.468991803874, -4.541726188958], [-19.015380734353, 2.468991803874, 4.541726188958],  
 ↵ [-20.287972134791, 2.634227398221, 5.825], [-29.43234602723, 2.539801873523, 5.825], [-  
 ↵ 30.710868433561, 2.650129252793, 4.541726188958], [-30.710868433561, 2.650129252793, -  
 ↵ 4.541726188958], [-29.43234602723, 2.539801873523, -5.825], [-20.43932801628, 0.  
 ↵ 88024858448, -5.825], [-19.157242606687, 0.825033762051, -4.541726188958], [-19.  
 ↵ 157242606687, 0.825033762051, 4.541726188958], [-20.43932801628, 0.88024858448, 5.825],  
 ↵ [-29.541726188958, 0.0, 5.825], [-30.825, 0.0, 4.541726188958], [-30.825, 0.0, -4.  
 ↵ 541726188958], [-29.541726188958, 0.0, -5.825], [-20.43932801628, -0.88024858448, -5.  
 ↵ 825], [-19.157242606687, -0.825033762051, -4.541726188958], [-19.157242606687, -0.  
 ↵ 825033762051, 4.541726188958], [-20.43932801628, -0.88024858448, 5.825], [-29.  
 ↵ 43234602723, -2.539801873523, 5.825], [-30.710868433561, -2.650129252793, 4.  
 ↵ 541726188958], [-30.710868433561, -2.650129252793, -4.541726188958], [-29.43234602723,  
 ↵ -2.539801873523, -5.825], [-20.287972134791, -2.634227398221, -5.825], [-19.  
 ↵ 015380734353, -2.468991803874, -4.541726188958], [-19.015380734353, -2.468991803874, 4.  
 ↵ 541726188958], [-20.287972134791, -2.634227398221, 5.825], [-29.105015516366, -5.  
 ↵ 060796183947, 5.825], [-30.369318893333, -5.280633953898, 4.541726188958], [-30.  
 ↵ 369318893333, -5.280633953898, -4.541726188958], [-29.105015516366, -5.060796183947, -  
 ↵ 5.825], [-19.986381181793, -4.368699415575, -5.825], [-18.732707495294, -4.  
 ↵ 094666640371, -4.541726188958], [-18.732707495294, -4.094666640371, 4.541726188958], [-  
 ↵ 19.986381181793, -4.368699415575, 5.825], [-28.562158581358, -7.544314640622, 5.825],  
 ↵ [-29.802880598069, -7.872034873984, 4.541726188958], [-29.802880598069, -7.  
 ↵ 872034873984, -4.541726188958], [-28.562158581358, -7.544314640622, -5.825], [-19.  
 ↵ 536788477508, -6.070820645736, -5.825], [-18.311316121598, -5.690019938005, -4.  
 ↵ 541726188958], [-18.311316121598, -5.690019938005, 4.541726188958], [-19.536788477508,  
 ↵ -6.070820645736, 5.825], [-27.807795148393, -9.971966466468, 5.825], [-29.015748096995,  
 ↵ -10.405142352304, 4.541726188958], [-29.015748096995, -10.405142352304, -4.  
 ↵ 541726188958], [-27.807795148393, -9.971966466468, -5.825], [-18.942523314366, -7.  
 ↵ 727986659686, -5.825], [-17.754327071178, -7.24323789818, -4.541726188958], [-17.  
 ↵ 754327071178, -7.24323789818, 4.541726188958], [-18.942523314366, -7.727986659686, 5.  
 ↵ 825], [-26.847511376774, -12.325774584072, 5.825], [-28.013750208625, -12.861198398624,  
 ↵ 4.541726188958], [-28.013750208625, -12.861198398624, -4.541726188958], [-26.  
 ↵ 847511376774, -12.325774584072, -5.825], [-18.207986303167, -9.327925927625, -5.825],  
 ↵ [-17.065864920372, -8.74281873995, -4.541726188958], [-17.065864920372, -8.74281873995,  
 ↵ 4.541726188958], [-18.207986303167, -9.327925927625, 5.825], [-25.688418292686, -14.  
 ↵ 588308738278, 5.825], [-26.804306857601, -15.222015598585, 4.541726188958], [-26.  
 ↵ 804306857601, -15.222015598585, -4.541726188958], [-25.688418292686, -14.588308738278,  
 ↵ -5.825], [-17.338616785984, -10.858790691249, -5.825], [-16.25102782092, -10.  
 ↵ 177657872206, -4.541726188958], [-16.25102782092, -10.177657872206, 4.541726188958], [-  
 ↵ 17.338616785984, -10.85879069125, 5.825], [-24.339099131124, -16.742814569501, 5.825],  
 ↵ [-25.396374129191, -17.470111793866, 4.541726188958], [-25.396374129191, -17.  
 ↵ 470111793866, -4.541726188958], [-24.339099131124, -16.742814569501, -5.825], [-16.

(continues on next page)

(continued from previous page)

↵ 340852557127, -12.309244697941, -5.825], [-15.315849747489, -11.537130124616, -4.  
 ↵ 541726188958], [-15.315849747489, -11.537130124616, 4.541726188958], [-16.340852557127,  
 ↵ -12.309244697941, 5.825], [-22.809545775916, -18.773337681929, 5.825], [-23.  
 ↵ 800377948308, -19.588839539842, 4.541726188958], [-23.800377948308, -19.588839539842, -  
 ↵ 4.541726188958], [-22.809545775916, -18.773337681929, -5.825], [-15.222082190442, -13.  
 ↵ 668547147192, -5.825], [-14.267255815306, -12.811168428391, -4.541726188958], [-14.  
 ↵ 267255815306, -12.811168428391, 4.541726188958], [-15.222082190442, -13.668547147192,  
 ↵ 5.825], [-21.111084768533, -20.664841787906, 5.825], [-22.028136874184, -21.  
 ↵ 562509382079, 4.541726188958], [-22.028136874184, -21.562509382079, -4.541726188958],  
 ↵ [-21.111084768533, -20.664841787906, -5.825], [-13.990590325955, -14.926632227628, -5.  
 ↵ 825], [-13.113010998777, -13.990338364241, -4.541726188958], [-13.113010998777, -13.  
 ↵ 990338364241, 4.541726188958], [-13.990590325955, -14.926632227628, 5.825], [-19.  
 ↵ 256293433567, -22.403320053593, 5.825], [-20.0927745824, -23.376506038775, 4.  
 ↵ 541726188958], [-20.0927745824, -23.376506038775, -4.541726188958], [-19.256293433567,  
 ↵ -22.403320053593, -5.825], [-12.655496321042, -16.074183655653, -5.825], [-11.  
 ↵ 861662630843, -15.065908025475, -4.541726188958], [-11.861662630843, -15.065908025475,  
 ↵ 4.541726188958], [-12.655496321042, -16.074183655653, 5.825], [-17.258906742015, -23.  
 ↵ 975898821395, 5.825], [-18.008622682363, -25.017396628832, 4.541726188958], [-18.  
 ↵ 008622682363, -25.017396628832, -4.541726188958], [-17.258906742015, -23.975898821395,  
 ↵ -5.825], [-11.2266867204, -17.102703663736, -5.825], [-10.522477108869, -16.  
 ↵ 029912678905, -4.541726188958], [-10.522477108869, -16.029912678905, 4.541726188958],  
 ↵ [-11.2266867204, -17.102703663736, 5.825], [-15.133715602029, -25.37093294107, 5.825],  
 ↵ [-15.791114589875, -26.473030144081, 4.541726188958], [-15.791114589875, -26.  
 ↵ 473030144081, -4.541726188958], [-15.133715602029, -25.37093294107, -5.825], [-9.  
 ↵ 714742044896, -18.004575927489, -5.825], [-9.105371275769, -16.875213744732, -4.  
 ↵ 541726188958], [-9.105371275769, -16.875213744732, 5.225178566873], [-9.714742044896,  
 ↵ 18.004575927489, 7.475], [-15.589268932331, -25.89471207708, 7.475], [-16.749661460472,  
 ↵ -27.822193766121, 5.225178566873], [-16.749661460472, -27.822193766121, -5.  
 ↵ 225178566873], [-15.589268932331, -25.89471207708, -7.475], [-10.985728205065, -16.  
 ↵ 442546594627, -7.475], [-9.73585968626, -14.571844810094, -5.225178566873], [-9.  
 ↵ 73585968626, -14.571844810094, 5.225178566873], [-10.985728205065, -16.442546594627, 7.  
 ↵ 475], [-17.956142197799, -24.313337425615, 7.475], [-19.292713741404, -26.123108955334,  
 ↵ 5.225178566873], [-19.292713741404, -26.123108955334, -5.225178566873], [-17.  
 ↵ 956142197799, -24.313337425615, -7.475], [-12.483818155695, -15.336161415673, -7.475],  
 ↵ [-11.063508913009, -13.591335310843, -5.225178566873], [-11.063508913009, -13.  
 ↵ 591335310843, 5.225178566873], [-12.483818155695, -15.336161415673, 7.475], [-20.  
 ↵ 163753424001, -22.516315579052, 7.475], [-21.664649259082, -24.192325177232, 5.  
 ↵ 225178566873], [-21.664649259082, -24.192325177232, -5.225178566873], [-20.  
 ↵ 163753424001, -22.516315579052, -7.475], [-13.871182856134, -14.093752122261, -7.475],  
 ↵ [-12.293030325246, -12.490277435773, -5.225178566873], [-12.293030325246, -12.  
 ↵ 490277435773, 5.225178566873], [-13.871182856134, -14.093752122261, 7.475], [-22.  
 ↵ 192522198737, -20.519585226265, 7.475], [-23.844430126671, -22.046967522412, 5.  
 ↵ 225178566873], [-23.844430126671, -22.046967522412, -5.225178566873], [-22.  
 ↵ 192522198737, -20.519585226265, -7.475], [-15.135517072389, -12.72633826611, -7.475],  
 ↵ [-13.413518680339, -11.278437019915, -5.225178566873], [-13.413518680339, -11.  
 ↵ 278437019915, 5.225178566873], [-15.135517072389, -12.72633826611, 7.475], [-24.  
 ↵ 024454353285, -18.340856371127, 7.475], [-25.812722773391, -19.706064244899, 5.  
 ↵ 225178566873], [-25.812722773391, -19.706064244899, -5.225178566873], [-24.  
 ↵ 024454353285, -18.340856371127, -7.475], [-16.26560678959, -11.246048127164, -7.475],  
 ↵ [-14.415035804573, -9.96656248427, -5.225178566873], [-14.415035804573, -9.96656248427,  
 ↵ 5.225178566873], [-16.26560678959, -11.246048127164, 7.475], [-25.643301561725, -15.  
 ↵ 999453253591, 7.475], [-27.55206942366, -17.190377991011, 5.225178566873], [-27.

(continues on next page)

(continued from previous page)

↵ 55206942366, -17.190377991011, -5.225178566873], [-25.643301561725, -15.999453253591, -  
 ↵ 7.475], [-17.251428674722, -9.66601114178, -7.475], [-15.288698739805, -8.566289502667,  
 ↵ -5.225178566873], [-15.288698739805, -8.566289502667, 5.225178566873], [-17.  
 ↵ 251428674722, -9.66601114178, 7.475], [-27.034705455491, -13.516142953307, 7.475], [-  
 ↵ 29.047042938873, -14.522221645027, 5.225178566873], [-29.047042938873, -14.  
 ↵ 522221645027, -5.225178566873], [-27.034705455491, -13.516142953307, -7.475], [-18.  
 ↵ 084238978596, -8.000241451173, -7.475], [-16.026758530875, -7.090037799125, -5.  
 ↵ 225178566873], [-16.026758530875, -7.090037799125, 5.225178566873], [-18.084238978596,  
 ↵ -8.000241451173, 7.475], [-28.186324974955, -10.91295119597, 7.475], [-30.284383648435,  
 ↵ -11.725260425013, 5.225178566873], [-30.284383648435, -11.725260425013, -5.  
 ↵ 225178566873], [-28.186324974955, -10.91295119597, -7.475], [-18.756651088551, -6.  
 ↵ 263513602956, -7.475], [-16.622668954987, -5.550900991092, -5.225178566873], [-16.  
 ↵ 622668954987, -5.550900991092, 5.225178566873], [-18.756651088551, -6.263513602956, 7.  
 ↵ 475], [-29.087945828478, -8.212966996095, 7.475], [-31.253116956441, -8.824301984124,  
 ↵ 5.225178566873], [-31.253116956441, -8.824301984124, -5.225178566873], [-29.  
 ↵ 087945828478, -8.212966996095, -7.475], [-19.262701044026, -4.471231508266, -7.475], [-  
 ↵ 17.071144583437, -3.962530455577, -5.225178566873], [-17.071144583437, -3.962530455577,  
 ↵ 5.225178566873], [-19.262701044026, -4.471231508266, 7.475], [-29.731571088089, -5.  
 ↵ 440137868962, 7.475], [-31.94465068087, -5.845076379075, 5.225178566873], [-31.  
 ↵ 94465068087, -5.845076379075, -5.225178566873], [-29.731571088089, -5.440137868962, -7.  
 ↵ 475], [-19.597900433928, -2.639291816738, -7.475], [-17.368207660739, -2.339014248233,  
 ↵ -5.225178566873], [-17.368207660739, -2.339014248233, 5.225178566873], [-19.  
 ↵ 597900433928, -2.639291816738, 7.475], [-30.111492118256, -2.619057428082, 7.475], [-  
 ↵ 32.352851261964, -2.814007857349, 5.225178566873], [-32.352851261964, -2.814007857349,  
 ↵ -5.225178566873], [-30.111492118256, -2.619057428082, -7.475], [-19.759276206593, -0.  
 ↵ 783942921144, -7.475], [-17.511223385333, -0.694752149319, -5.225178566873], [-17.  
 ↵ 511223385333, -0.694752149319, 5.225178566873], [-19.759276206593, -0.783942921144, 7.  
 ↵ 475], [-30.22433920864, 0.225252747914, 7.475], [-32.474098163852, 0.242019512716, 5.  
 ↵ 225178566873], [-32.474098163852, 0.242019512716, -5.225178566873], [-30.22433920864,  
 ↵ 0.225252747914, -7.475], [-19.745397039283, 1.078359157762, -7.475], [-17.498923278961,  
 ↵ 0.955672055178, -5.225178566873], [-17.498923278961, 0.955672055178, 5.225178566873],  
 ↵ [-19.745397039283, 1.078359157762, 7.475], [-30.069111461746, 3.06756504421, 7.475], [-  
 ↵ 32.307315986892, 3.295900290227, 5.225178566873], [-32.307315986892, 3.295900290227, -  
 ↵ 5.225178566873], [-30.069111461746, 3.06756504421, -7.475], [-19.556386033299, 2.  
 ↵ 931096727954, -7.475], [-17.331416437442, 2.597619924464, -5.225178566873], [-17.  
 ↵ 331416437442, 2.597619924464, 5.225178566873], [-19.556386033299, 2.931096727954, 7.  
 ↵ 475], [-29.647185670387, 5.882669566189, 7.475], [-31.853984005939, 6.320548073498, 5.  
 ↵ 225178566873], [-31.853984005939, 6.320548073498, -5.225178566873], [-29.647185670387,  
 ↵ 5.882669566189, -7.475], [-19.193919622139, 4.757836929835, -7.475], [-17.010188563043,  
 ↵ 4.216528198615, -5.225178566873], [-17.010188563043, 4.216528198615, 5.225178566873],  
 ↵ [-19.193919622139, 4.757836929835, 7.475], [-28.962304106216, 8.645597738643, 7.475],  
 ↵ [-31.118123049907, 9.289135742945, 5.225178566873], [-31.118123049907, 9.289135742945,  
 ↵ -5.225178566873], [-28.962304106216, 8.645597738643, -7.475], [-18.661212702364, 6.  
 ↵ 542377487516, -7.475], [-16.538088787041, 5.798037967445, -5.225178566873], [-16.  
 ↵ 538088787041, 5.798037967445, 5.225178566873], [-18.661212702364, 6.542377487516, 7.  
 ↵ 475], [-28.020541327642, 11.3318437646, 7.475], [-30.106259839023, 12.175333404274, 5.  
 ↵ 225178566873], [-30.106259839023, 12.175333404274, -5.225178566873], [-28.020541327642,  
 ↵ 11.3318437646, -7.475], [-17.962990119069, 8.268890414939, -7.475], [-15.919304399347,  
 ↵ 7.328122026885, -5.225178566873], [-15.919304399347, 7.328122026885, 5.225178566873],  
 ↵ [-17.962990119069, 8.268890414939, 7.475], [-26.830250301518, 13.917581979542, 7.475],  
 ↵ [-28.827369096067, 14.953541921535, 5.225178566873], [-28.827369096067, 14.  
 ↵ 953541921535, -5.225178566873], [-26.830250301518, 13.917581979542, -7.475], [-17.

(continues on next page)

(continued from previous page)

```
→ 105444758857, 9.92206240223, -7.475], [-15.15932370933, 8.793209293298, -5.
→ 225178566873], [-15.15932370933, 8.793209293298, 5.225178566873], [-17.105444758857, 9.
→ 92206240223, 7.475], [-25.401988316491, 16.379878173181, 7.475], [-27.292793945051, 17.
→ 599119968709, 5.225178566873], [-27.292793945051, 17.599119968709, -5.225178566873], [-
→ 25.401988316491, 16.379878173181, -7.475], [-16.096182622021, 11.487230637129, -7.475], [
→ [-14.264887367244, 10.180305172235, -5.225178566873], [-14.264887367244, 10.
→ 180305172235, 5.225178566873], [-16.096182622021, 11.487230637129, 7.475], [-23.
→ 748423345103, 18.696893004485, 7.475], [-25.516145303356, 20.088602585994, 5.
→ 225178566873], [-25.516145303356, 20.088602585994, -5.225178566873], [-23.748423345103,
→ 18.696893004485, -7.475], [-14.944155361111, 12.950512856834, -7.475], [-13.
→ 243928578022, 11.477106814012, -5.225178566873], [-13.243928578022, 11.477106814012, 5.
→ 225178566873], [-14.944155361111, 12.950512856834, 7.475], [-21.884221685188, 20.
→ 848075705756, 7.475], [-23.513181159677, 22.399909302321, 5.225178566873], [-23.
→ 513181159677, 22.399909302321, -5.225178566873], [-23.513181159677, 22.399909302321, -
→ 14.95]], [<pyaedt.modeler.cad.polylines.Polyline object at 0x00000258F2A18970>, [[9.
→ 105371275769, 16.875213744732, -14.95], [9.105371275769, 16.875213744732, 4.
→ 541726188958], [9.714742044896, 18.004575927489, 5.825], [15.133715602029, 25.
→ 37093294107, 5.825], [15.791114589875, 26.473030144081, 4.541726188958], [15.
→ 791114589875, 26.473030144081, -4.541726188958], [15.133715602029, 25.37093294107, -5.
→ 825], [11.2266867204, 17.102703663736, -5.825], [10.522477108869, 16.029912678905, -4.
→ 541726188958], [10.522477108869, 16.029912678905, 4.541726188958], [11.2266867204, 17.
→ 102703663736, 5.825], [17.258906742015, 23.975898821395, 5.825], [18.008622682363, 25.
→ 017396628832, 4.541726188958], [18.008622682363, 25.017396628832, -4.541726188958],
→ [17.258906742015, 23.975898821395, -5.825], [12.655496321042, 16.074183655653, -5.825],
→ [11.861662630843, 15.065908025475, -4.541726188958], [11.861662630843, 15.
→ 065908025475, 4.541726188958], [12.655496321042, 16.074183655653, 5.825], [19.
→ 256293433567, 22.403320053593, 5.825], [20.0927745824, 23.376506038775, 4.
→ 541726188958], [20.0927745824, 23.376506038775, -4.541726188958], [19.256293433567, 22.
→ 403320053593, -5.825], [13.990590325955, 14.926632227628, -5.825], [13.113010998777,
→ 13.990338364241, -4.541726188958], [13.113010998777, 13.990338364241, 4.541726188958],
→ [13.990590325955, 14.926632227628, 5.825], [21.111084768533, 20.664841787906, 5.825],
→ [22.028136874184, 21.562509382079, 4.541726188958], [22.028136874184, 21.562509382079,
→ -4.541726188958], [21.111084768533, 20.664841787906, -5.825], [15.222082190442, 13.
→ 668547147192, -5.825], [14.267255815306, 12.811168428391, -4.541726188958], [14.
→ 267255815306, 12.811168428391, 4.541726188958], [15.222082190442, 13.668547147192, 5.
→ 825], [22.809545775916, 18.773337681929, 5.825], [23.800377948308, 19.588839539842, 4.
→ 541726188958], [23.800377948308, 19.588839539842, -4.541726188958], [22.809545775916,
→ 18.773337681929, -5.825], [16.340852557127, 12.309244697941, -5.825], [15.315849747489,
→ 11.537130124616, -4.541726188958], [15.315849747489, 11.537130124616, 4.541726188958],
→ [16.340852557127, 12.309244697941, 5.825], [24.339099131124, 16.742814569501, 5.825],
→ [25.396374129191, 17.470111793866, 4.541726188958], [25.396374129191, 17.470111793866,
→ -4.541726188958], [24.339099131124, 16.742814569501, -5.825], [17.338616785984, 10.
→ 85879069125, -5.825], [16.25102782092, 10.177657872206, -4.541726188958], [16.
→ 25102782092, 10.177657872206, 4.541726188958], [17.338616785984, 10.85879069125, 5.
→ 825], [25.688418292686, 14.588308738278, 5.825], [26.804306857601, 15.222015598585, 4.
→ 541726188958], [26.804306857601, 15.222015598585, -4.541726188958], [25.688418292686,
→ 14.588308738278, -5.825], [18.207986303167, 9.327925927625, -5.825], [17.065864920372,
→ 8.74281873995, -4.541726188958], [17.065864920372, 8.74281873995, 4.541726188958], [18.
→ 207986303167, 9.327925927625, 5.825], [26.847511376774, 12.325774584072, 5.825], [28.
→ 013750208625, 12.861198398624, 4.541726188958], [28.013750208625, 12.861198398624, -4.
→ 541726188958], [26.847511376774, 12.325774584072, -5.825], [18.942523314366, 7.
→ 727986659686, -5.825], [17.754327071178, 7.24323789818, -4.541726188958], [17.
```

(continues on next page)

(continued from previous page)

↵ 754327071178, 7.24323789818, 4.541726188958], [18.942523314366, 7.727986659686, 5.825],  
 ↵ [27.807795148393, 9.971966466468, 5.825], [29.015748096995, 10.405142352304, 4.  
 ↵ 541726188958], [29.015748096995, 10.405142352304, -4.541726188958], [27.807795148393,  
 ↵ 9.971966466468, -5.825], [19.536788477508, 6.070820645736, -5.825], [18.311316121598,  
 ↵ 5.690019938005, -4.541726188958], [18.311316121598, 5.690019938005, 4.541726188958],  
 ↵ [19.536788477508, 6.070820645736, 5.825], [28.562158581358, 7.544314640622, 5.825],  
 ↵ [29.802880598069, 7.872034873984, 4.541726188958], [29.802880598069, 7.872034873984, -  
 ↵ 4.541726188958], [28.562158581358, 7.544314640622, -5.825], [19.986381181793, 4.  
 ↵ 368699415575, -5.825], [18.732707495294, 4.094666640371, -4.541726188958], [18.  
 ↵ 732707495294, 4.094666640371, 4.541726188958], [19.986381181793, 4.368699415575, 5.  
 ↵ 825], [29.105015516366, 5.060796183947, 5.825], [30.369318893333, 5.280633953898, 4.  
 ↵ 541726188958], [30.369318893333, 5.280633953898, -4.541726188958], [29.105015516366, 5.  
 ↵ 060796183947, -5.825], [20.287972134791, 2.634227398221, -5.825], [19.015380734353, 2.  
 ↵ 468991803874, -4.541726188958], [19.015380734353, 2.468991803874, 4.541726188958], [20.  
 ↵ 287972134791, 2.634227398221, 5.825], [29.43234602723, 2.539801873523, 5.825], [30.  
 ↵ 710868433561, 2.650129252793, 4.541726188958], [30.710868433561, 2.650129252793, -4.  
 ↵ 541726188958], [29.43234602723, 2.539801873523, -5.825], [20.43932801628, 0.  
 ↵ 88024858448, -5.825], [19.157242606687, 0.825033762051, -4.541726188958], [19.  
 ↵ 157242606687, 0.825033762051, 4.541726188958], [20.43932801628, 0.88024858448, 5.825],  
 ↵ [29.541726188958, 0.0, 5.825], [30.825, 0.0, 4.541726188958], [30.825, 0.0, -4.  
 ↵ 541726188958], [29.541726188958, 0.0, -5.825], [20.43932801628, -0.88024858448, -5.  
 ↵ 825], [19.157242606687, -0.825033762051, -4.541726188958], [19.157242606687, -0.  
 ↵ 825033762051, 4.541726188958], [20.43932801628, -0.88024858448, 5.825], [29.  
 ↵ 43234602723, -2.539801873523, 5.825], [30.710868433561, -2.650129252793, 4.  
 ↵ 541726188958], [30.710868433561, -2.650129252793, -4.541726188958], [29.43234602723, -  
 ↵ 2.539801873523, -5.825], [20.287972134791, -2.634227398221, -5.825], [19.015380734353,  
 ↵ -2.468991803874, -4.541726188958], [19.015380734353, -2.468991803874, 4.541726188958],  
 ↵ [20.287972134791, -2.634227398221, 5.825], [29.105015516366, -5.060796183947, 5.825],  
 ↵ [30.369318893333, -5.280633953898, 4.541726188958], [30.369318893333, -5.280633953898,  
 ↵ -4.541726188958], [29.105015516366, -5.060796183947, -5.825], [19.986381181793, -4.  
 ↵ 368699415575, -5.825], [18.732707495294, -4.094666640371, -4.541726188958], [18.  
 ↵ 732707495294, -4.094666640371, 4.541726188958], [19.986381181793, -4.368699415575, 5.  
 ↵ 825], [28.562158581358, -7.544314640622, 5.825], [29.802880598069, -7.872034873984, 4.  
 ↵ 541726188958], [29.802880598069, -7.872034873984, -4.541726188958], [28.562158581358, -  
 ↵ 7.544314640622, -5.825], [19.536788477508, -6.070820645736, -5.825], [18.311316121598,  
 ↵ -5.690019938005, -4.541726188958], [18.311316121598, -5.690019938005, 4.541726188958],  
 ↵ [19.536788477508, -6.070820645736, 5.825], [27.807795148393, -9.971966466468, 5.825],  
 ↵ [29.015748096995, -10.405142352304, 4.541726188958], [29.015748096995, -10.  
 ↵ 405142352304, -4.541726188958], [27.807795148393, -9.971966466468, -5.825], [18.  
 ↵ 942523314366, -7.727986659686, -5.825], [17.754327071178, -7.24323789818, -4.  
 ↵ 541726188958], [17.754327071178, -7.24323789818, 4.541726188958], [18.942523314366, -7.  
 ↵ 727986659686, 5.825], [26.847511376774, -12.325774584072, 5.825], [28.013750208625, -  
 ↵ 12.861198398624, 4.541726188958], [28.013750208625, -12.861198398624, -4.541726188958],  
 ↵ [26.847511376774, -12.325774584072, -5.825], [18.207986303167, -9.327925927625, -5.  
 ↵ 825], [17.065864920372, -8.74281873995, -4.541726188958], [17.065864920372, -8.  
 ↵ 74281873995, 4.541726188958], [18.207986303167, -9.327925927625, 5.825], [25.  
 ↵ 688418292686, -14.588308738278, 5.825], [26.804306857601, -15.222015598585, 4.  
 ↵ 541726188958], [26.804306857601, -15.222015598585, -4.541726188958], [25.688418292686,  
 ↵ -14.588308738278, -5.825], [17.338616785984, -10.858790691249, -5.825], [16.  
 ↵ 25102782092, -10.177657872206, -4.541726188958], [16.25102782092, -10.177657872206, 4.  
 ↵ 541726188958], [17.338616785984, -10.85879069125, 5.825], [24.339099131124, -16.  
 ↵ 742814569501, 5.825], [25.396374129191, -17.470111793866, 4.541726188958], [25.

(continues on next page)

(continued from previous page)

→ 396374129191, -17.470111793866, -4.541726188958], [24.339099131124, -16.742814569501, -5.825], [16.340852557127, -12.309244697941, -5.825], [15.315849747489, -11.537130124616, 4.541726188958], [16.340852557127, -12.309244697941, 5.825], [22.809545775916, -18.773337681929, 5.825], [23.800377948308, -19.588839539842, 4.541726188958], [23.800377948308, -19.588839539842, -4.541726188958], [22.809545775916, -18.773337681929, -5.825], [15.222082190442, -13.668547147192, -5.825], [14.267255815306, -12.811168428391, 4.541726188958], [15.222082190442, -13.668547147192, 5.825], [21.111084768533, -20.664841787906, 5.825], [22.028136874184, -21.562509382079, 4.541726188958], [22.028136874184, -21.562509382079, -4.541726188958], [21.111084768533, -20.664841787906, -5.825], [13.990590325955, -14.926632227628, -5.825], [13.113010998777, -13.990338364241, -4.541726188958], [13.990590325955, -14.926632227628, 5.825], [19.256293433567, -22.403320053593, 5.825], [20.0927745824, -23.376506038775, -4.541726188958], [19.256293433567, -22.403320053593, -5.825], [12.655496321042, -16.074183655653, -5.825], [11.861662630843, -15.065908025475, -4.541726188958], [11.861662630843, -15.065908025475, 4.541726188958], [12.655496321042, -16.074183655653, 5.825], [17.258906742015, -23.975898821395, -5.825], [11.2266867204, -17.102703663736, -5.825], [10.522477108869, -16.029912678905, 4.541726188958], [11.2266867204, -17.102703663736, 5.825], [15.133715602029, -25.37093294107, 5.825], [15.791114589875, -26.473030144081, 4.541726188958], [15.791114589875, -26.473030144081, -4.541726188958], [15.133715602029, -25.37093294107, -5.825], [9.714742044896, -18.004575927489, -5.825], [9.105371275769, -16.875213744732, -4.541726188958], [9.105371275769, -16.875213744732, -4.541726188958], [9.105371275769, -16.875213744732, 5.225178566873], [9.714742044896, -18.004575927489, 7.475], [15.589268932331, -25.89471207708, 7.475], [16.749661460472, -27.822193766121, 5.225178566873], [15.589268932331, -25.89471207708, -7.475], [10.985728205065, -16.442546594627, -7.475], [9.73585968626, -14.571844810094, -5.225178566873], [9.73585968626, -14.571844810094, 5.225178566873], [10.985728205065, -16.442546594627, 7.475], [17.956142197799, -24.313337425615, 7.475], [19.292713741404, -26.123108955334, 5.225178566873], [19.292713741404, -26.123108955334, -5.225178566873], [17.956142197799, -24.313337425615, -7.475], [12.483818155695, -15.336161415673, -7.475], [11.063508913009, -13.591335310843, -5.225178566873], [12.483818155695, -15.336161415673, 7.475], [20.163753424001, -22.516315579052, 7.475], [21.664649259082, -24.192325177232, 5.225178566873], [21.664649259082, -24.192325177232, -5.225178566873], [20.163753424001, -22.516315579052, -7.475], [13.871182856134, -14.093752122261, 7.475], [12.293030325246, -12.490277435773, 5.225178566873], [13.871182856134, -14.093752122261, 7.475], [22.192522198737, -20.519585226265, 7.475], [23.844430126671, -22.046967522412, 5.225178566873], [23.844430126671, -22.046967522412, -5.225178566873], [22.192522198737, -20.519585226265, -7.475], [15.135517072389, -12.72633826611, -7.475], [13.413518680339, -11.278437019915, -5.225178566873], [13.413518680339, -11.278437019915, 5.225178566873], [15.135517072389, -12.72633826611, 7.475], [24.024454353285, -18.340856371127, 7.475], [25.812722773391, -19.706064244899, -5.225178566873], [24.024454353285, -18.340856371127, -7.475], [16.26560678959, -11.246048127164, -7.475], [14.415035804573, -9.96656248427, -5.225178566873], [14.415035804573, -9.96656248427, 5.225178566873], [16.26560678959, -11.246048127164, 7.475], [25.643301561725, -15.999453253591, 7.475], [27.55206942366, -17.190377991011, 5.225178566873], [27.55206942366, -17.190377991011, -5.225178566873], [27.55206942366, -17.190377991011, -7.475]

(continues on next page)

(continued from previous page)

↵ 55206942366, -17.190377991011, -5.225178566873], [25.643301561725, -15.999453253591, -  
 ↵ 7.475], [17.251428674722, -9.66601114178, -7.475], [15.288698739805, -8.566289502667, -  
 ↵ 5.225178566873], [15.288698739805, -8.566289502667, 5.225178566873], [17.251428674722, -  
 ↵ -9.66601114178, 7.475], [27.034705455491, -13.516142953307, 7.475], [29.047042938873, -  
 ↵ 14.522221645027, 5.225178566873], [29.047042938873, -14.522221645027, -5.225178566873],  
 ↵ [27.034705455491, -13.516142953307, -7.475], [18.084238978596, -8.000241451173, -7.  
 ↵ 475], [16.026758530875, -7.090037799125, -5.225178566873], [16.026758530875, -7.  
 ↵ 090037799125, 5.225178566873], [18.084238978596, -8.000241451173, 7.475], [28.  
 ↵ 186324974955, -10.91295119597, 7.475], [30.284383648435, -11.725260425013, 5.  
 ↵ 225178566873], [30.284383648435, -11.725260425013, -5.225178566873], [28.186324974955, -  
 ↵ -10.91295119597, -7.475], [18.756651088551, -6.263513602956, -7.475], [16.622668954987,  
 ↵ -5.550900991092, -5.225178566873], [16.622668954987, -5.550900991092, 5.225178566873],  
 ↵ [18.756651088551, -6.263513602956, 7.475], [29.087945828478, -8.212966996095, 7.475],  
 ↵ [31.253116956441, -8.824301984124, 5.225178566873], [31.253116956441, -8.824301984124,  
 ↵ -5.225178566873], [29.087945828478, -8.212966996095, -7.475], [19.262701044026, -4.  
 ↵ 471231508266, -7.475], [17.071144583437, -3.962530455577, -5.225178566873], [17.  
 ↵ 071144583437, -3.962530455577, 5.225178566873], [19.262701044026, -4.471231508266, 7.  
 ↵ 475], [29.731571088089, -5.440137868962, 7.475], [31.94465068087, -5.845076379075, 5.  
 ↵ 225178566873], [31.94465068087, -5.845076379075, -5.225178566873], [29.731571088089, -  
 ↵ 5.440137868962, -7.475], [19.597900433928, -2.639291816738, -7.475], [17.368207660739,  
 ↵ -2.339014248233, -5.225178566873], [17.368207660739, -2.339014248233, 5.225178566873],  
 ↵ [19.597900433928, -2.639291816738, 7.475], [30.111492118256, -2.619057428082, 7.475],  
 ↵ [32.352851261964, -2.814007857349, 5.225178566873], [32.352851261964, -2.814007857349,  
 ↵ -5.225178566873], [30.111492118256, -2.619057428082, -7.475], [19.759276206593, -0.  
 ↵ 783942921144, -7.475], [17.511223385333, -0.694752149319, -5.225178566873], [17.  
 ↵ 511223385333, -0.694752149319, 5.225178566873], [19.759276206593, -0.783942921144, 7.  
 ↵ 475], [30.22433920864, 0.225252747914, 7.475], [32.474098163852, 0.242019512716, 5.  
 ↵ 225178566873], [32.474098163852, 0.242019512716, -5.225178566873], [30.22433920864, 0.  
 ↵ 225252747914, -7.475], [19.745397039283, 1.078359157762, -7.475], [17.498923278961, 0.  
 ↵ 955672055178, -5.225178566873], [17.498923278961, 0.955672055178, 5.225178566873], [19.  
 ↵ 745397039283, 1.078359157762, 7.475], [30.069111461746, 3.06756504421, 7.475], [32.  
 ↵ 307315986892, 3.295900290227, 5.225178566873], [32.307315986892, 3.295900290227, -5.  
 ↵ 225178566873], [30.069111461746, 3.06756504421, -7.475], [19.556386033299, 2.  
 ↵ 931096727954, -7.475], [17.331416437442, 2.597619924464, -5.225178566873], [17.  
 ↵ 331416437442, 2.597619924464, 5.225178566873], [19.556386033299, 2.931096727954, 7.  
 ↵ 475], [29.647185670387, 5.882669566189, 7.475], [31.853984005939, 6.320548073498, 5.  
 ↵ 225178566873], [31.853984005939, 6.320548073498, -5.225178566873], [29.647185670387, 5.  
 ↵ 882669566189, -7.475], [19.193919622139, 4.757836929835, -7.475], [17.010188563043, 4.  
 ↵ 216528198615, -5.225178566873], [17.010188563043, 4.216528198615, 5.225178566873], [19.  
 ↵ 193919622139, 4.757836929835, 7.475], [28.962304106216, 8.645597738643, 7.475], [31.  
 ↵ 118123049907, 9.289135742945, 5.225178566873], [31.118123049907, 9.289135742945, -5.  
 ↵ 225178566873], [28.962304106216, 8.645597738643, -7.475], [18.661212702364, 6.  
 ↵ 542377487516, -7.475], [16.538088787041, 5.798037967445, -5.225178566873], [16.  
 ↵ 538088787041, 5.798037967445, 5.225178566873], [18.661212702364, 6.542377487516, 7.  
 ↵ 475], [28.020541327642, 11.3318437646, 7.475], [30.106259839023, 12.175333404274, 5.  
 ↵ 225178566873], [30.106259839023, 12.175333404274, -5.225178566873], [28.020541327642,  
 ↵ 11.3318437646, -7.475], [17.962990119069, 8.268890414939, -7.475], [15.919304399347, 7.  
 ↵ 328122026885, -5.225178566873], [15.919304399347, 7.328122026885, 5.225178566873], [17.  
 ↵ 962990119069, 8.268890414939, 7.475], [26.830250301518, 13.917581979542, 7.475], [28.  
 ↵ 827369096067, 14.953541921535, 5.225178566873], [28.827369096067, 14.953541921535, -5.  
 ↵ 225178566873], [26.830250301518, 13.917581979542, -7.475], [17.105444758857, 9.  
 ↵ 92206240223, -7.475], [15.15932370933, 8.793209293298, -5.225178566873], [15.

(continues on next page)

(continued from previous page)

```

→ 15932370933, 8.793209293298, 5.225178566873], [17.105444758857, 9.92206240223, 7.475], ↵
→ [25.401988316491, 16.379878173181, 7.475], [27.292793945051, 17.599119968709, 5. ↵
→ 225178566873], [27.292793945051, 17.599119968709, -5.225178566873], [25.401988316491, ↵
→ 16.379878173181, -7.475], [16.096182622021, 11.487230637129, -7.475], [14.264887367244, ↵
→ 10.180305172235, -5.225178566873], [14.264887367244, 10.180305172235, 5.225178566873], ↵
→ [16.096182622021, 11.487230637129, 7.475], [23.748423345103, 18.696893004485, 7.475], ↵
→ [25.516145303356, 20.088602585994, 5.225178566873], [25.516145303356, 20.088602585994, ↵
→ -5.225178566873], [23.748423345103, 18.696893004485, -7.475], [14.944155361111, 12. ↵
→ 950512856834, -7.475], [13.243928578022, 11.477106814012, -5.225178566873], [13. ↵
→ 243928578022, 11.477106814012, 5.225178566873], [14.944155361111, 12.950512856834, 7. ↵
→ 475], [21.884221685188, 20.848075705756, 7.475], [23.513181159677, 22.399909302321, 5. ↵
→ 225178566873], [23.513181159677, 22.399909302321, -5.225178566873], [23.513181159677, ↵
→ 22.399909302321, -14.95]]]

```

## Create ground

Create a ground.

```

ground_radius = 1.2 * dictionary_values[1]["Outer Winding"]["Outer Radius"]
ground_position = [0, 0, first_winding_list[1][0][2] - 2]
ground = hfss.modeler.create_circle("XY", ground_position, ground_radius, name="GND", ↵
material="copper")
coat = hfss.assign_coating(ground, is_infinite_ground=True)

```

## Create lumped ports

Create lumped ports.

```

port_position_list = [
 [first_winding_list[1][0][0], first_winding_list[1][0][1], first_winding_ ↵
list[1][0][2] - 1],
 [first_winding_list[1][-1][0], first_winding_list[1][-1][1], first_winding_list[1][- ↵
1][2] - 1],
 [second_winding_list[1][0][0], second_winding_list[1][0][1], second_winding_ ↵
list[1][0][2] - 1],
 [second_winding_list[1][-1][0], second_winding_list[1][-1][1], second_winding_ ↵
list[1][-1][2] - 1],
]
port_dimension_list = [2, dictionary_values[1]["Outer Winding"]["Wire Diameter"]]
for position in port_position_list:
 sheet = hfss.modeler.create_rectangle("XZ", position, port_dimension_list, name= ↵
"sheet_port")
 sheet.move([-dictionary_values[1]["Outer Winding"]["Wire Diameter"] / 2, 0, -1])
 hfss.lumped_port(assignment=sheet.name, reference=[ground], ↵
name="port_" + str(port_position_list.index(position) + 1))

```

## Create mesh

Create the mesh.

```
cylinder_height = 2.5 * dictionary_values[1]["Outer Winding"]["Height"]
cylinder_position = [0, 0, first_winding_list[1][0][2] - 4]
mesh_operation_cylinder = hfss.modeler.create_cylinder("XY", cylinder_position, ground_
 ↵radius, cylinder_height,
 ↵ num_sides=36, name="mesh_cylinder
 ↵")
hfss.mesh.assign_length_mesh([mesh_operation_cylinder], maximum_length=15, maximum_
 ↵elements=None, name="choke_mesh")
```

```
<pyaedt.modules.Mesh.MeshOperation object at 0x00000258EA1CA530>
```

## Create boundaries

Create the boundaries. A region with openings is needed to run the analysis.

```
region = hfss.modeler.create_region(pad_percent=1000)
```

## Create setup

Create a setup with a sweep to run the simulation. Depending on your machines computing power, the simulation can take some time to run.

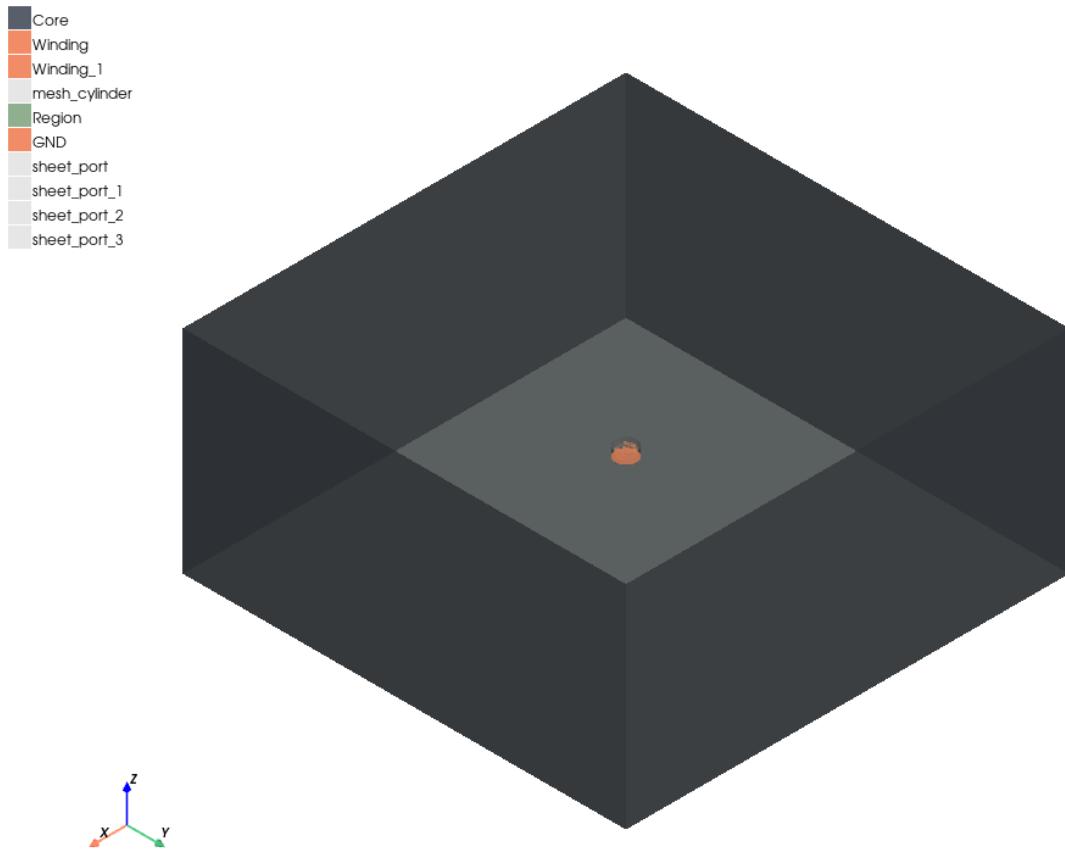
```
setup = hfss.create_setup("MySetup")
setup.props["Frequency"] = "50MHz"
setup["MaximumPasses"] = 10
hfss.create_linear_count_sweep(setup=setup.name, units="MHz", start_frequency=0.1, stop_
 ↵frequency=100,
 ↵ num_of_freq_points=100, name="sweep1", save_fields=False, ↵
 ↵sweep_type="Interpolating")
```

```
<pyaedt.modules.SolveSweeps.SweepHFSS object at 0x00000258EA1CAA40>
```

## Save project

Save the project.

```
hfss.modeler.fit_all()
hfss.plot(show=False, export_path=os.path.join(hfss.working_directory, "Image.jpg"),
 ↵plot_air_objects=True)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258EA1CBFA0>
```

## Close AEDT

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

```
hfss.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 20.714 seconds)

## HFSS: dipole antenna

This example shows how you can use PyAEDT to create a dipole antenna in HFSS and postprocess results.

### Perform required imports

Perform required imports.

```
import os
import pyaedt

project_name = pyaedt.generate_unique_project_name(project_name="dipole")
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

### Launch AEDT

Launch AEDT 2023 R2 in graphical mode.

```
d = pyaedt.launch_desktop(aedt_version, non_graphical=non_graphical, new_desktop_=
 session=True)
```

### Launch HFSS

Launch HFSS 2023 R2 in graphical mode.

```
hfss = pyaedt.Hfss(projectname=project_name, solution_type="Modal")
```

## Define variable

Define a variable for the dipole length.

```
hfss["l_dipole"] = "13.5cm"
```

## Get 3D component from system library

Get a 3D component from the syslib directory. For this example to run correctly, you must get all geometry parameters of the 3D component or, in case of an encrypted 3D component, create a dictionary of the parameters.

```
compfile = hfss.components3d["Dipole_Antenna_DM"]
geometriparams = hfss.get_components3d_vars("Dipole_Antenna_DM")
geometriparams["dipole_length"] = "l_dipole"
hfss.modeler.insert_3d_component(compfile, geometriparams)
```

```
<pyaedt.modeler.cad.components_3d.UserDefinedComponent object at 0x00000258EA1C8430>
```

## Create boundaries

Create boundaries. A region with openings is needed to run the analysis.

```
hfss.create_open_region(frequency="1GHz")
```

```
True
```

## Plot model

Plot the model.

```
my_plot = hfss.plot(show=False, plot_air_objects=False)
my_plot.show_axes = False
my_plot.show_grid = False
my_plot.isometric_view = False
my_plot.plot(
 os.path.join(hfss.working_directory, "Image.jpg"),
)
```



arm1  
arm1\_1  
port1

True

## Create setup

Create a setup with a sweep to run the simulation.

```
setup = hfss.create_setup("MySetup")
setup.props["Frequency"] = "1GHz"
setup.props["MaximumPasses"] = 1
hfss.create_linear_count_sweep(setup=setup.name, units="GHz", start_frequency=0.5, stop_
 ↪frequency=1.5,
 num_of_freq_points=251, name="sweep1", save_fields=False, ↪
 ↪sweep_type="Interpolating",
 interpolation_tol=3, interpolation_max_solutions=255)
```

<pyaedt.modules.SolveSweeps.SweepHFSS object at 0x00000258EA1C8730>

## Save and run simulation

Save and run the simulation.

```
hfss.analyze_setup("MySetup")
```

```
True
```

## Create scattering plot and far fields report

Create a scattering plot and a far fields report.

```
hfss.create_scattering("MyScattering")
variations = hfss.available_variations.nominal_w_values_dict
variations["Freq"] = ["1GHz"]
variations["Theta"] = ["All"]
variations["Phi"] = ["All"]
hfss.post.create_report("db(GainTotal)", hfss.nominal_adaptive, variations, primary_
 ↴sweep_variable="Theta",
 report_category="Far Fields", context="3D")
```

```
<pyaedt.modules.report_templates.FarField object at 0x00000258EA1C9570>
```

## Create far fields report using report objects

Create a far fields report using the `report_by_category.far_field` method, which gives you more freedom.

```
new_report = hfss.post.reports_by_category.far_field("db(RealizedGainTotal)", hfss.
 ↴nominal_adaptive, "3D")
new_report.variations = variations
new_report.primary_sweep = "Theta"
new_report.create("Realized2D")
```

```
True
```

## Generate multiple plots

Generate multiple plots using the object `new_report`. This code generates 2D and 3D polar plots.

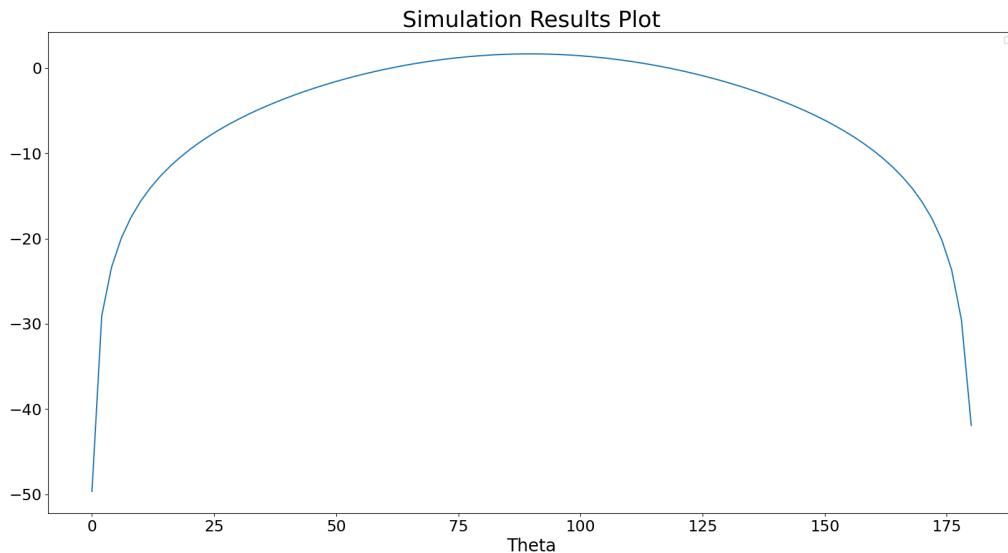
```
new_report.report_type = "3D Polar Plot"
new_report.secondary_sweep = "Phi"
new_report.create("Realized3D")
```

```
True
```

## Get solution data

Get solution data using the object `new_report`` and postprocess or plot the data outside AEDT.

```
solution_data = new_report.get_solution_data()
solution_data.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

`<Figure size 2000x1000 with 1 Axes>`

## Generate far field plot

Generate a far field plot by creating a postprocessing variable and assigning it to a new coordinate system. You can use the `post` prefix to create a postprocessing variable directly from a setter, or you can use the `set_variable` method with an arbitrary name.

```
hfss["post_x"] = 2
hfss.variable_manager.set_variable(variable_name="y_post", expression=1,
 postprocessing=True)
hfss.modeler.create_coordinate_system(origin=["post_x", "y_post", 0], name="CS_Post")
hfss.insert_infinite_sphere(custom_coordinate_system="CS_Post", name="Sphere_Custom")
```

`<pyaedt.modules.Boundary.FarFieldSetup object at 0x00000258F8AA2EF0>`

## Get solution data

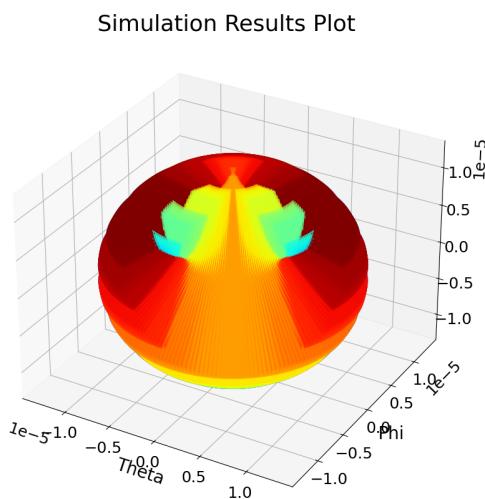
Get solution data. You can use this code to generate the same plot outside AEDT.

```
new_report = hfss.post.reports_by_category.far_field("GainTotal", hfss.nominal_adaptive,
→ "3D")
new_report.primary_sweep = "Theta"
new_report.far_field_sphere = "3D"
solutions = new_report.get_solution_data()
```

## Generate 3D plot using Matplotlib

Generate a 3D plot using Matplotlib.

```
solutions.plot_3d()
```



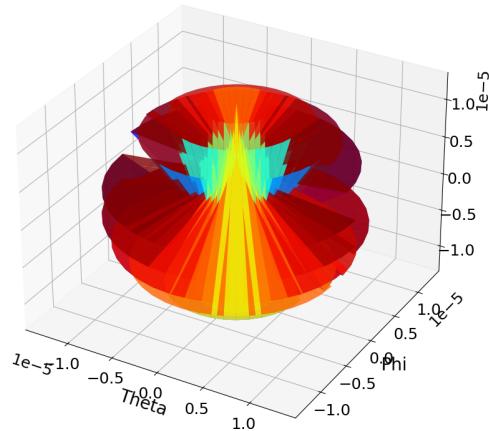
```
<Figure size 2000x1000 with 1 Axes>
```

## Generate 3D far fields plot using Matplotlib

Generate a far fields plot using Matplotlib.

```
new_report.far_field_sphere = "Sphere_Custom"
solutions_custom = new_report.get_solution_data()
solutions_custom.plot_3d()
```

Simulation Results Plot

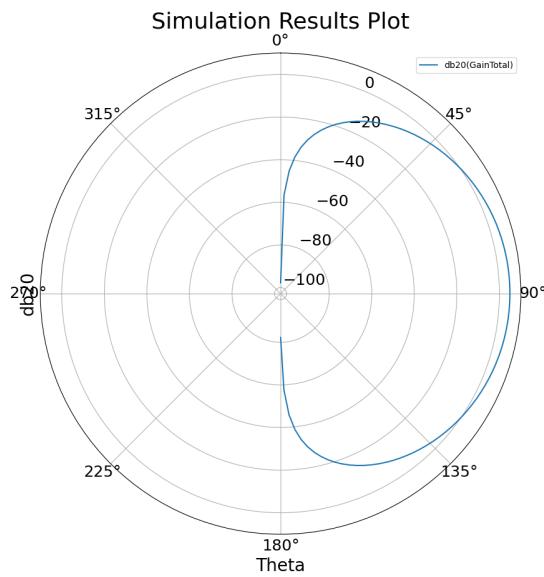


<Figure size 2000x1000 with 1 Axes>

### Generate 2D plot using Matplotlib

Generate a 2D plot using Matplotlib where you specify whether it is a polar plot or a rectangular plot.

```
solutions.plot(formula="db20", is_polar=True)
```



<Figure size 2000x1000 with 1 Axes>

## Get far field data

Get far field data. After the simulation completes, the far field data is generated port by port and stored in a data class, user can use this data once AEDT is released.

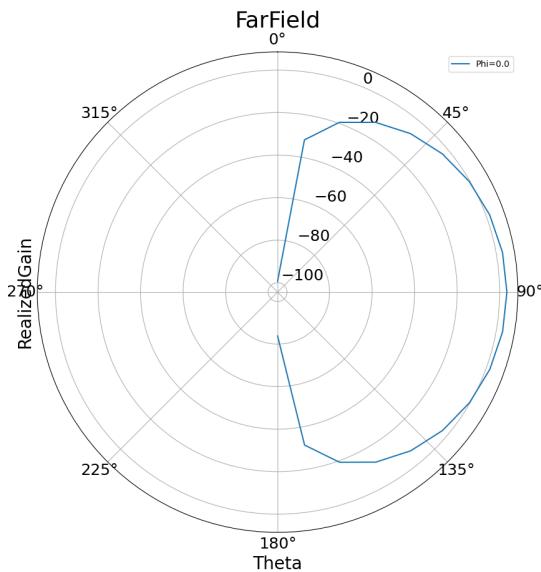
```
ffdata = hfss.get_antenna_ffd_solution_data(frequencies=["1000MHz"], setup=hfss.nominal_
 ↪adaptive,
 sphere="Sphere_Custom")
```



## Generate 2D cutout plot

Generate 2D cutout plot. You can define the Theta scan and Phi scan.

```
ffdata.plot_2d_cut(quantity='RealizedGain', primary_sweep="theta", secondary_sweep_
 ↪value=0, title='FarField',
 quantity_format="dB20", is_polar=True)
```



```
<Figure size 2000x1000 with 1 Axes>
```

## Close AEDT

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

```
d.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 34.274 seconds)

## HFSS: FSS Unitcell Simulation

This example shows how you can use PyAEDT to create a FSS unitcell simulations in HFSS and postprocess results.

### Perform required imports

Perform required imports.

```
import os
import pyaedt

project_name = pyaedt.generate_unique_project_name(project_name="FSS")
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

## Launch AEDT

Launch AEDT 2023 R2 in graphical mode.

```
d = pyaedt.launch_desktop(aedt_version, non_graphical=non_graphical, new_desktop_=
 ↪session=True)
```

## Launch HFSS

Launch HFSS 2023 R2 in graphical mode.

```
hfss = pyaedt.Hfss(projectname=project_name, solution_type="Modal")
```

## Define variable

Define a variable for the 3D-component.

```
hfss["patch_dim"] = "10mm"
```

## Insert 3D component from system library

Download the 3D component from the example data and insert the 3D Component.

```
unitcell_3d_component_path = pyaedt.downloads.download_FSS_3dcomponent()
unitcell_path = os.path.join(unitcell_3d_component_path, "FSS_unitcell_23R2.a3dcomp")

comp = hfss.modeler.insert_3d_component(unitcell_path)
```

## Assign design parameter to 3D Component parameter

Assign parameter.

```
component_name = hfss.modeler.user_defined_component_names
comp.parameters["a"] = "patch_dim"
```

## Create air region

Create an open region along +Z direction for unitcell analysis.

```
bounding_dimensions = hfss.modeler.get_bounding_dimension()

periodicity_x = bounding_dimensions[0]
periodicity_y = bounding_dimensions[1]

region = hfss.modeler.create_air_region(
 z_pos=10 * bounding_dimensions[2],
 is_percentage=False,
)
[x_min, y_min, z_min, x_max, y_max, z_max] = region.bounding_box
```

## Assign Lattice pair boundary

Assigning lattice pair boundary automatically detected.

```
hfss.auto_assign_lattice_pairs(assignment=region.name)

['LatticePair1', 'LatticePair2']
```

## Assign Floquet port excitation along +Z direction

Assign Floquet port.

```
id_z_pos = region.top_face_z
hfss.create_floquet_port(id_z_pos, [0, 0, z_max], [0, y_max, z_max], [x_max, 0, z_max], name='port_z_max',
 deembed_distance=10 * bounding_dimensions[2])

<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258A80968F0>
```

## Create setup

Create a setup with a sweep to run the simulation.

```
setup = hfss.create_setup("MySetup")
setup.props["Frequency"] = "10GHz"
setup.props["MaximumPasses"] = 10
hfss.create_linear_count_sweep(setup=setup.name, units="GHz", start_frequency=6, stop_
 ↪frequency=15,
 num_of_freq_points=51, name="sweep1", save_fields=False, ↪
 ↪sweep_type="Interpolating",
 interpolation_tol=6)
```

```
<pyaedt.modules.SolveSweeps.SweepHFSS object at 0x00000258A8095B40>
```

## Create S-parameter report using report objects

Create S-parameter reports using create report.

```
all_quantities = hfss.post.available_report_quantities()
str_mag = []
str_ang = []

variation = {"Freq": ["All"]}

for i in all_quantities:
 str_mag.append("mag(" + i + ")")
 str_ang.append("ang_deg(" + i + ")")

hfss.post.create_report(expressions=str_mag, variations=variation, plot_name="magnitude_"
 ↪plot")
hfss.post.create_report(expressions=str_ang, variations=variation, plot_name="phase_plot"
 ↪")
```

```
<pyaedt.modules.report_templates.Standard object at 0x00000258A80943A0>
```

## Save and run simulation

Save and run the simulation. Uncomment the line following line to run the analysis.

```
hfss.analyze()
```

## Close AEDT

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

```
hfss.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 51.973 seconds)

## HFSS: spiral inductor

This example shows how you can use PyAEDT to create a spiral inductor, solve it, and plot results.

### Perform required imports

Perform required imports.

```
import os
import pyaedt

project_name = pyaedt.generate_unique_project_name(project_name="spiral")
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Launch HFSS

Launch HFSS 2023 R2 in non-graphical mode and change the units to microns.

```
hfss = pyaedt.Hfss(specified_version=aedt_version, non_graphical=non_graphical,
 designname="A1",
 new_desktop_session=True)
hfss.solution_type = "Modal"
hfss.modeler.model_units = "um"
p = hfss.modeler
```

## Define variables

Define input variables. You can use the values that follow or edit them.

```
rin = 10
width = 2
spacing = 1
thickness = 1
Np = 8
Nr = 10
gap = 3
hfss["Tsub"] = "6" + hfss.modeler.model_units
```

## Standardize polyline

Standardize the polyline using the `create_line` method to fix the width, thickness, and material.

```
def create_line(pts):
 p.create_polyline(pts, material="copper", xsection_type="Rectangle", xsection_
 width=width,
 xsection_height=thickness)
```

## Create spiral inductor

Create the spiral inductor. This spiral inductor is not parametric, but you could parametrize it later.

```
ind = hfss.modeler.create_spiral(
 internal_radius=rin,
 width=width,
 spacing=spacing,
 turns=Nr,
 faces=Np,
 thickness=thickness,
 material="copper",
 name="Inductor1",
)
```

## Center return path

Center the return path.

```
x0, y0, z0 = ind.points[0]
x1, y1, z1 = ind.points[-1]
create_line([(x0 - width / 2, y0, -gap), (abs(x1) + 5, y0, -gap)])
p.create_box([x0 - width / 2, y0 - width / 2, -gap - thickness / 2], [width, width, gap ↴+ thickness], material="copper")
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258F2A188B0>
```

## Create port 1

Create port 1.

```
p.create_rectangle(orientation=pyaedt.constants.PLANE.YZ,
 origin=[abs(x1) + 5, y0 - width / 2, -gap - thickness / 2],
 sizes=[width, "-Tsub+{}{}".format(gap, hfss.modeler.model_units)],
 name="port1")
hfss.lumped_port(assignment="port1", integration_line=pyaedt.constants.AXIS.Z)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F2A1BE80>
```

## Create port 2

Create port 2.

```
create_line([(x1 + width / 2, y1, 0), (x1 - 5, y1, 0)])
p.create_rectangle(pyaedt.constants.PLANE.YZ, [x1 - 5, y1 - width / 2, -thickness / 2],
 [width, "-Tsub"],
 name="port2")
hfss.lumped_port(assignment="port2", integration_line=pyaedt.constants.AXIS.Z)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F2A187C0>
```

## Create silicon substrate and ground plane

Create the silicon substrate and the ground plane.

```
p.create_box([x1 - 20, x1 - 20, "-Tsub-{}{}/2".format(thickness, hfss.modeler.model_units)],
 [-2 * x1 + 40, -2 * x1 + 40, "Tsub"], material="silicon")

p.create_box([x1 - 20, x1 - 20, "-Tsub-{}{}/2".format(thickness, hfss.modeler.model_units)],
 [-2 * x1 + 40, -2 * x1 + 40, -0.1], material="PEC")
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258F2A1B310>
```

### Assign airbox and radiation

Assign the airbox and the radiation.

```
box = p.create_box(
 [x1 - 20, x1 - 20, "-Tsub-{}{}/2 - 0.1{}".format(thickness, hfss.modeler.model_units,
 ↪ hfss.modeler.model_units)],
 [-2 * x1 + 40, -2 * x1 + 40, 100], name="airbox", material="air")

hfss.assign_radiation_boundary_to_objects("airbox")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258F2A1A350>
```

### Assign material override

Assign a material override so that the validation check does not fail.

```
hfss.change_material_override()
```

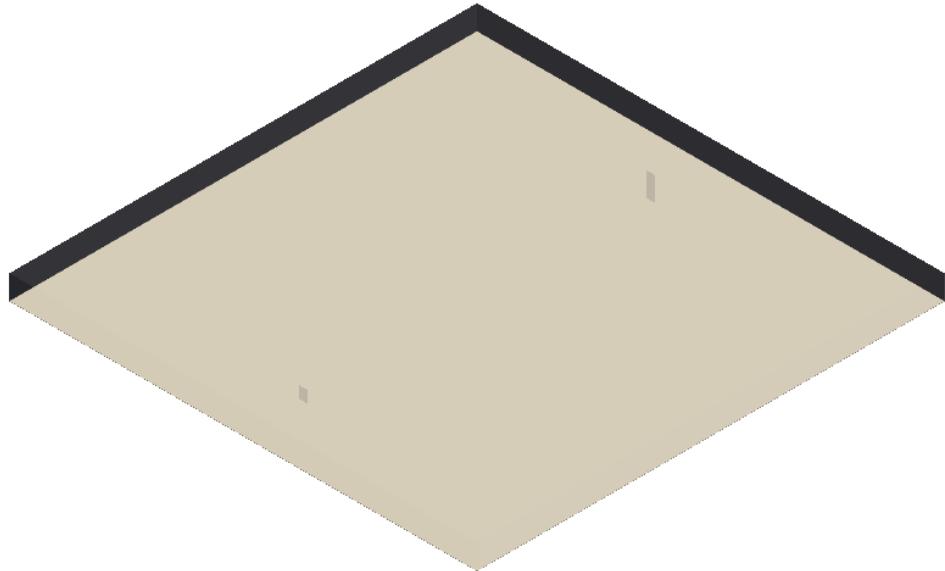
```
True
```

### Plot model

Plot the model.

```
hfss.plot(show=False, export_path=os.path.join(hfss.working_directory, "Image.jpg"),
 ↪ plot_air_objects=False)
```

	Inductor1
	NewObject_0BF2YQ
	NewObject_263HS4
	NewObject_6G3TRW
	NewObject_E53YFJ
	NewObject_FRH8X0
	port1
	port2



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258F2A1BD00>
```

## Create setup

Create the setup and define a frequency sweep to solve the project.

```
setup1 = hfss.create_setup(name="setup1")
setup1.props["Frequency"] = "10GHz"
hfss.create_linear_count_sweep(setup="setup1", units="GHz", start_frequency=1e-3, stop_
 ↪frequency=50,
 num_of_freq_points=451, sweep_type="Interpolating")
hfss.save_project()
hfss.analyze()
```

```
True
```

## Get report data

Get report data and use the following formulas to calculate the inductance and quality factor.

```
L_formula = "1e9*im(1/Y(1,1))/(2*pi*freq)"
Q_formula = "im(Y(1,1))/re(Y(1,1))"
```

## Create output variable

Create output variable

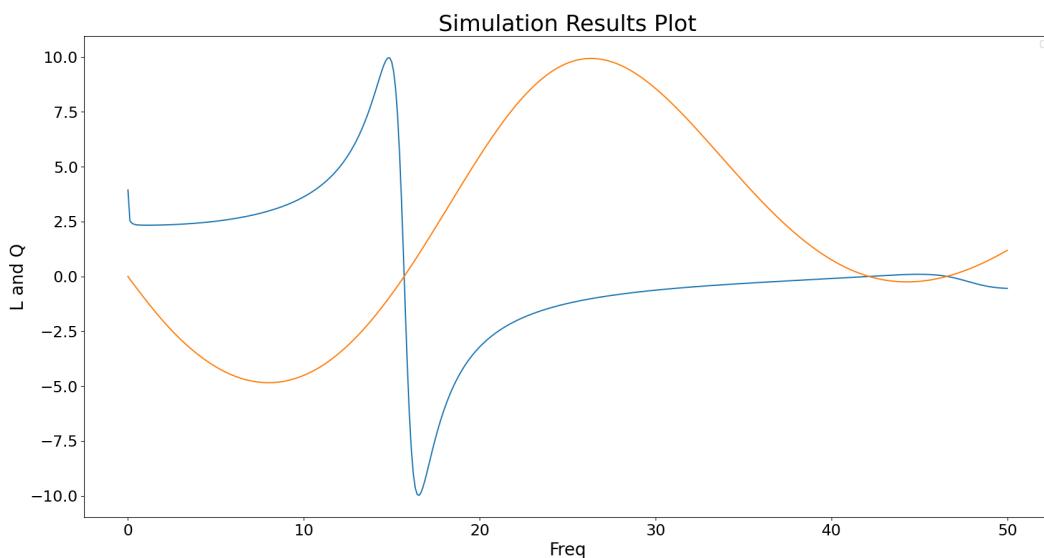
```
hfss.create_output_variable("L", L_formula, solution="setup1 : LastAdaptive")
```

```
True
```

## Plot calculated values in Matplotlib

Plot the calculated values in Matplotlib.

```
data = hfss.post.get_solution_data([L_formula, Q_formula])
data.plot(curves=[L_formula, Q_formula], formula="re", x_label="Freq", y_label="L and Q")
```



```
No artists with labels found to put in legend. Note that artists whose label start with _
an underscore are ignored when legend() is called with no argument.
```

```
<Figure size 2000x1000 with 1 Axes>
```

## Export results to csv file

Export results to csv file

```
data.export_data_to_csv(os.path.join(hfss.toolkit_directory, "output.csv"))
```

True

## Save project and close AEDT

Save the project and close AEDT.

```
hfss.save_project(project_name)
hfss.release_desktop()
```

True

**Total running time of the script:** (2 minutes 6.793 seconds)

## HFSS: Eigenmode filter

This example shows how you can use PyAEDT to automate the eigenmode solver in HFSS. Eigenmode analysis can be applied to open, radiating structures using an absorbing boundary condition. This type of analysis is useful for determining the resonant frequency of a geometry or an antenna and can be used to refine the mesh at the resonance, even when the resonant frequency of the antenna is not known.

The challenge posed by this method is to identify and filter the non-physical modes resulting from reflection from boundaries of the main domain. Because the Eigenmode solver sorts by frequency and does not filter on the quality factor, these virtual modes are present when the eigenmode approach is applied to nominally open structures. When looking for resonant modes over a wide frequency range for nominally enclosed structures, several iterations may be required because the minimum frequency is determined manually and simulations re-run until the complete frequency range is covered and all important physical modes are calculated.

The following script finds the physical modes of a model in a wide frequency range by automating the solution setup. During each simulation, a user-defined number of modes is simulated, and the modes with a Q higher than a user-defined value are filtered. The next simulation automatically continues to find modes having a frequency higher than the last mode of the previous analysis. This continues until the maximum frequency in the desired range is achieved.

## Perform required imports

Run through each cell. This cell imports the required packages.

```
import sys
import os
import pyaedt

Create a temporary folder to download the example to.

temp_folder = pyaedt.generate_unique_folder_name()
project_path = pyaedt.download_file("eigenmode", "emi_PCB_house.aedt", temp_
→folder)
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Launch AEDT

Launch AEDT 2023 R2 in graphical mode.

```
d = pyaedt.launch_desktop(aedt_version, non_graphical=non_graphical, new_desktop_=
 ↪session=True)
```

## Launch HFSS

Launch HFSS 2023 R2 in graphical mode.

```
hfss = pyaedt.Hfss(projectname=project_path, non_graphical=non_graphical)
```

## Input parameters for eigenmode solver

The geometry and material should be already set. The analyses are generated by the code. Number of modes during each analysis, max allowed number is 20. Entering a number higher than 10 might need long simulation time as the eigenmode solver needs to converge on modes. `fmin` is the lowest frequency of interest. `fmax` is the highest frequency of interest. `limit` is the parameter limit that determines which modes are ignored.

```
num_modes = 6
fmin = 1
fmax = 2
next_fmin = fmin
setup_nr = 1

limit = 10
resonance = {}
```

## Find the modes

The following cell is a function. If called, it creates an eigenmode setup and solves it. After the solve, each mode, along with its corresponding real frequency and quality factor, are saved for further processing.

```
def find_resonance():
 # setup creation
 next_min_freq = str(next_fmin) + " GHz"
 setup_name = "em_setup" + str(setup_nr)
 setup = hfss.create_setup(setup_name)
 setup.props['MinimumFrequency'] = next_min_freq
 setup.props['NumModes'] = num_modes
 setup.props['ConvergeOnRealFreq'] = True
 setup.props['MaximumPasses'] = 10
 setup.props['MinimumPasses'] = 3
 setup.props['MaxDeltaFreq'] = 5
 # analyzing the eigenmode setup
 hfss.analyze_setup(setup_name, cores=8, use_auto_settings=True)
 # getting the Q and real frequency of each mode
 eigen_q = hfss.post.available_report_quantities(quantities_category="Eigen Q")
 eigen_mode = hfss.post.available_report_quantities()
 data = {}
 cont = 0
 for i in eigen_mode:
 eigen_q_value = hfss.post.get_solution_data(expressions=eigen_q[cont],
 setup_sweep_name=setup_name + ' : '
 ↪LastAdaptive',
 report_category="Eigenmode")
 eigen_mode_value = hfss.post.get_solution_data(expressions=eigen_mode[cont],
 setup_sweep_name=setup_name + ' : '
 ↪LastAdaptive',
 report_category="Eigenmode")
 data[cont] = [eigen_q_value.data_real()[0], eigen_mode_value.data_real()[0]]
 cont += 1

 print(data)
 return data
```

## Automate eigenmode solution

Running the next cell calls the resonance function and saves only those modes with a Q higher than the defined limit. The `find_resonance` function is called until the complete frequency range is covered. When the automation ends, the physical modes in the whole frequency range are reported.

```
while next_fmin < fmax:
 output = find_resonance()
 next_fmin = output[len(output) - 1][1] / 1e9
 setup_nr += 1
 cont_res = len(resonance)
 for q in output:
 if output[q][0] > limit:
 resonance[cont_res] = output[q]
```

(continues on next page)

(continued from previous page)

```

cont_res += 1

resonance_frequencies = [f'{resonance[i][1] / 1e9:.5} GHz' for i in resonance]
print(str(resonance_frequencies))

{0: [110.84360086592784, 1369784273.07735], 1: [1.2787481882242286, 1488174006.10637], 2: [0.7526747858563642, 1746620660.63392], 3: [0.6749380619287106, 1748998178.13158], 4: [0.6652140915487272, 1762041793.53741], 5: [388.12028334033624, 1856703504.11958]}
{0: [433.3549638792473, 2314972023.45024], 1: [1.0974536975835376, 2350970945.929], 2: [1.605521322208046, 2375043714.13562], 3: [1.8426438873717188, 2519292658.75623], 4: [394.9903855203817, 2773232062.13332], 5: [695.9490363997055, 2786043715.62203]}
['1.3698 GHz', '1.8567 GHz', '2.315 GHz', '2.7732 GHz', '2.786 GHz']

```

## Save project

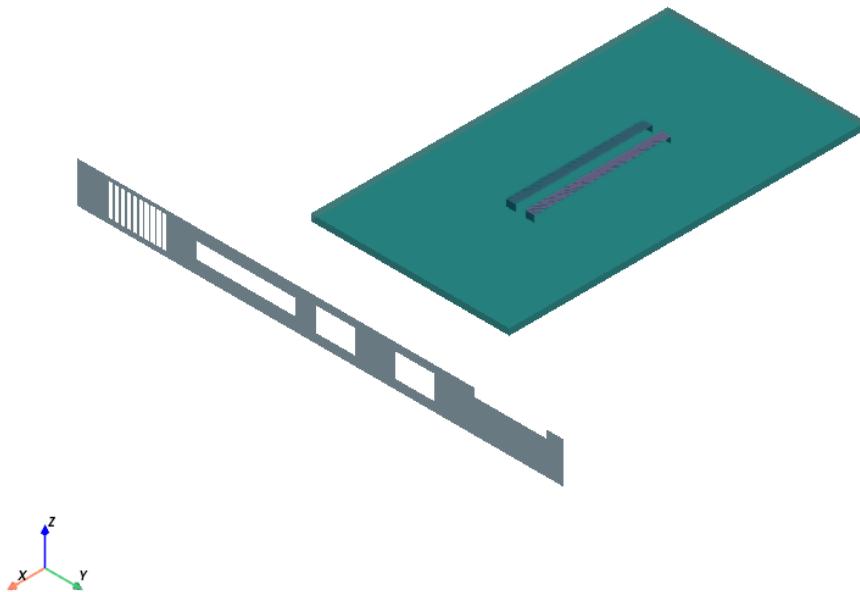
Save the project.

```

hfss.modeler.fit_all()
hfss.plot(show=False, export_path=os.path.join(hfss.working_directory, "Image.jpg"), plot_air_objects=False)

```

substrate
trace1
port1
term1
ground
port2
term2
trace2
front_face



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258F2A1A350>
```

### Save project and close AEDT

Save the project and close AEDT.

```
hfss.save_project()
hfss.release_desktop()
```

```
True
```

**Total running time of the script:** (2 minutes 53.735 seconds)

### HFSS: Probe-fed patch antenna

This example shows how to use the `Stackup3D` class to create and analyze a patch antenna in HFSS.

Note that the HFSS 3D Layout interface may offer advantages for laminate structures such as the patch antenna.

### Perform imports

```
import os

import pyaedt
import tempfile
from pyaedt.modeler.advanced_cad.stackup_3d import Stackup3D
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. "PYAEDT\_NON\_GRAPHICAL" is set to `False` to create this documentation.

You can set `non_graphical` to `True` to view HFSS while the notebook cells are executed.

```
non_graphical = False
length_units = "mm"
freq_units = "GHz"
```

## Create temporary working folder

Use tempfile to create a temporary working folder. Project data is deleted after this example is run.

To save the project data in another location, change the location of the project directory.

```
tmpdir.cleanup() at the end of this notebook removes all
project files and data.

tmpdir = tempfile.TemporaryDirectory(suffix="_aedt")
project_folder = tmpdir.name
proj_name = os.path.join(project_folder, "antenna")
```

## Launch HFSS

```
hfss = pyaedt.Hfss(projectname=proj_name,
 solution_type="Terminal",
 designname="patch",
 non_graphical=non_graphical,
 new_desktop_session=True,
 specified_version=aedt_version)

hfss.modeler.model_units = length_units
```

## Create patch

Create the patch.

```
stackup = Stackup3D(hfss)
ground = stackup.add_ground_layer("ground", material="copper", thickness=0.035, fill_
material="air")
dielectric = stackup.add_dielectric_layer("dielectric", thickness="0.5" + length_units,_
material="Duroid (tm)")
signal = stackup.add_signal_layer("signal", material="copper", thickness=0.035, fill_
material="air")
patch = signal.add_patch(patch_length=9.57, patch_width=9.25,
 patch_name="Patch", frequency=1E10)

stackup.resize_around_element(patch)
pad_length = [3, 3, 3, 3, 3, 3] # Air bounding box buffer in mm.
region = hfss.modeler.create_region(pad_length, is_percentage=False)
hfss.assign_radiation_boundary_to_objects(region.name)

patch.create_probe_port(ground, rel_x_offset=0.485)
setup = hfss.create_setup(name="Setup1",
 setup_type="HFSSDriven",
 Frequency="10GHz")

setup.create_frequency_sweep(unit="GHz",
 name="Sweep1",
 start_frequency=8,
```

(continues on next page)

(continued from previous page)

```

stop_frequency=12,
sweep_type="Interpolating")

hfss.save_project()
hfss.analyze()

```

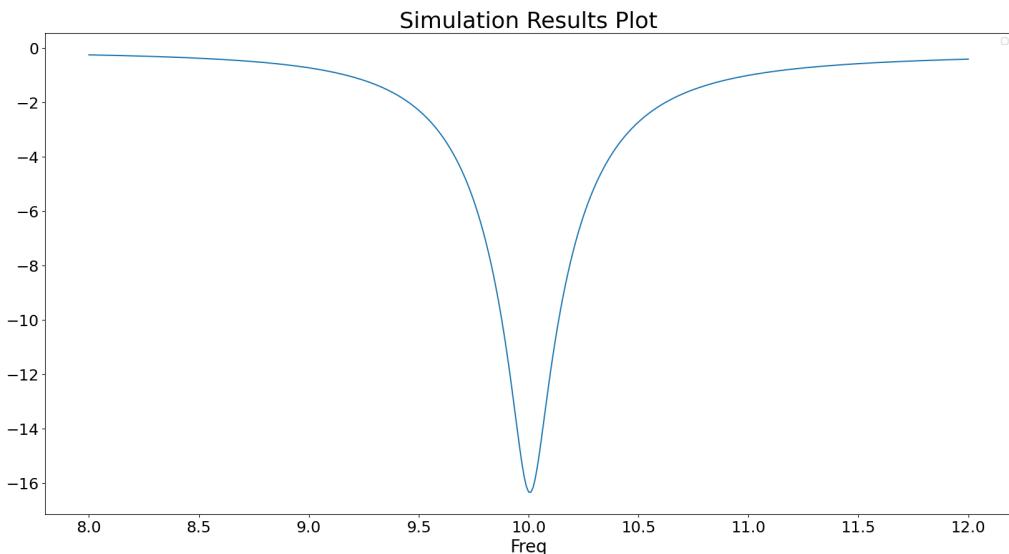
```
True
```

### Plot S11

```

plot_data = hfss.get_traces_for_plot()
report = hfss.post.create_report(plot_data)
solution = report.get_solution_data()
plt = solution.plot(solution.expressions)

```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

### Release AEDT

Release AEDT and clean up temporary folders and files.

```

hfss.release_desktop()
tmpdir.cleanup()

```

**Total running time of the script:** (2 minutes 10.862 seconds)

## HFSS: Inductive Iris waveguide filter

This example shows how to build and analyze a 4-pole X-Band waveguide filter using inductive irises.

```
sphinx_gallery_thumbnail_path = 'Resources/wgf.png'
```

### Perform required imports

Perform required imports.

```
import os
import tempfile
import pyaedt
from pyaedt import general_methods
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Launch Ansys Electronics Desktop (AEDT)

#### Define parameters and values for waveguide iris filter

**l:** Length of the cavity from the mid-point of one iris  
to the midpoint of the next iris.

w: Width of the iris opening. a: Long dimension of the waveguide cross-section (X-Band) b: Short dimension of the waveguide cross-section. t: Metal thickness of the iris insert.

```
wgparams = {'l': [0.7428, 0.82188],
 'w': [0.50013, 0.3642, 0.3458],
 'a': 0.4,
 'b': 0.9,
 't': 0.15,
 'units': 'in'}

non_graphical = False
new_thread = True
```

## Save the project and results in the TEMP folder

```
project_folder = os.path.join(tempfile.gettempdir(), "waveguide_example")
if not os.path.exists(project_folder):
 os.mkdir(project_folder)
project_name = os.path.join(project_folder, general_methods.generate_unique_name("wgf", n=2))

Instantiate the HFSS application
hfss = pyaedt.Hfss(projectname=project_name + '.aedt',
 specified_version=aedt_version,
 designname="filter",
 non_graphical=non_graphical,
 new_desktop_session=True,
 close_on_exit=True,
 solution_type="Modal")

hfss.settings.enable_debug_methods_argument_logger = False # Only for debugging.

var_mapping = dict() # Used by parse_expr to parse expressions.
```

## Initialize design parameters in HFSS.

```
hfss.modeler.model_units = "in" # Set to inches
for key in wgparams:
 if type(wgparams[key]) in [int, float]:
 hfss[key] = str(wgparams[key]) + wgparams['units']
 var_mapping[key] = wgparams[key] # Used for expression parsing
 elif type(wgparams[key]) == list:
 count = 1
 for v in wgparams[key]:
 this_key = key + str(count)
 hfss[this_key] = str(v) + wgparams['units']
 var_mapping[this_key] = v # Used to parse expressions and generate numerical values.
 count += 1

if len(wgparams['l']) % 2 == 0:
 zstart = "-t/2" # Even number of cavities, odd number of irises.
 is_even = True
else:
 zstart = "11/2 - t/2" # Odd number of cavities, even number of irises.
 is_even = False
```

## Draw parametric waveguide filter

Define a function to place each iris at the correct longitudinal (z) position. Loop from the largest index (interior of the filter) to 1, which is the first iris nearest the waveguide ports.

```
def place_iris(zpos, dz, n):
 w_str = "w" + str(n) # Iris width parameter as a string.
 this_name = "iris_a_" + str(n) # Iris object name in the HFSS project.
 iris = [] # Return a list of the two objects that make up the iris.
 if this_name in hfss.modeler.object_names:
 this_name = this_name.replace("a", "c")
 iris.append(hfss.modeler.create_box(origin=[-b/2, -a/2, zpos],
 sizes=[(b - w_str)/2, 'a', dz],
 name=this_name,
 material="silver"))
 iris.append(iris[0].mirror([0, 0, 0], [1, 0, 0], duplicate=True))
 return iris
```

## Place irises

Place the irises from inner (highest integer) to outer.

```
for count in reversed(range(1, len(wgparams['w']) + 1)):
 if count < len(wgparams['w']): # Update zpos
 zpos = zpos + "" .join([" + 1" + str(count) + " + "])[:-3]
 iris = place_iris(zpos, "t", count)
 iris = place_iris("-(" + zpos + ")", "-t", count)

 else: # Place first iris
 zpos = zstart
 iris = place_iris(zpos, "t", count)
 if not is_even:
 iris = place_iris("-(" + zpos + ")", "-t", count)
```

## Draw full waveguide with ports

Use `hfss.variable_manager` which acts like a `dict()` to return an instance of the `pyaedt.application.Variables.VariableManager` class for any variable. The `VariableManager` instance takes the HFSS variable name as a key. `VariableManager` properties enable access to update, modify and evaluate variables.

```
var_mapping['port_extension'] = 1.5 * wgparams['l'][0]
hfss['port_extension'] = str(var_mapping['port_extension']) + wgparams['units']
hfss["wg_z_start"] = "-(" + zpos + ") - port_extension"
hfss["wg_length"] = "2*(" + zpos + " + port_extension)"
wg_z_start = hfss.variable_manager["wg_z_start"]
wg_length = hfss.variable_manager["wg_length"]
hfss["u_start"] = "-a/2"
hfss["u_end"] = "a/2"
hfss.modeler.create_box(origin=[-b/2, -a/2, wg_z_start],
 sizes=[b, "a", "wg_length"],
```

(continues on next page)

(continued from previous page)

```
 name="waveguide",
 material="vacuum")
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258A897DEA0>
```

### Draw the whole waveguide.

`wg_z` is the total length of the waveguide, including port extension. Note that the `.evaluated_value` provides access to the numerical value of `wg_z_start` which is an expression in HFSS.

```
wg_z = [wg_z_start.evaluated_value, hfss.value_with_units(wg_z_start.numeric_value + wg_
↳length.numeric_value, "in")]
```

Assign wave ports to the end faces of the waveguid and define the calibration lines to ensure self-consistent polarization between wave ports. ~~~~~

```
count = 0
ports = []
for n, z in enumerate(wg_z):
 face_id = hfss.modeler.get_faceid_from_position(position=[0, 0, z], assignment=
↳"waveguide")
 u_start = [0, hfss.variable_manager["u_start"].evaluated_value, z]
 u_end = [0, hfss.variable_manager["u_end"].evaluated_value, z]

 ports.append(hfss.wave_port(face_id, integration_line=[u_start, u_end], name="P" +_
↳str(n + 1), renormalize=False))
```

### Insert the mesh adaptation setup using refinement at two frequencies.

This approach is useful for resonant structures as the coarse initial mesh impacts the resonant frequency and hence, the field propagation through the filter. Adaptation at multiple frequencies helps to ensure that energy propagates through the resonant structure while the mesh is refined.

```
setup = hfss.create_setup("Setup1", setup_type="HFSSDriven",
 MultipleAdaptiveFreqsSetup=['9.8GHz', '10.2GHz'],
 MaximumPasses=5)

setup.create_frequency_sweep(
 unit="GHz",
 name="Sweep1",
 start_frequency=9.5,
 stop_frequency=10.5,
 sweep_type="Interpolating",
)
```

```
<pyaedt.modules.SolveSweeps.SweepHFSS object at 0x00000258A897F4F0>
```

Solve the project with two tasks. Each frequency point is solved simultaneously.

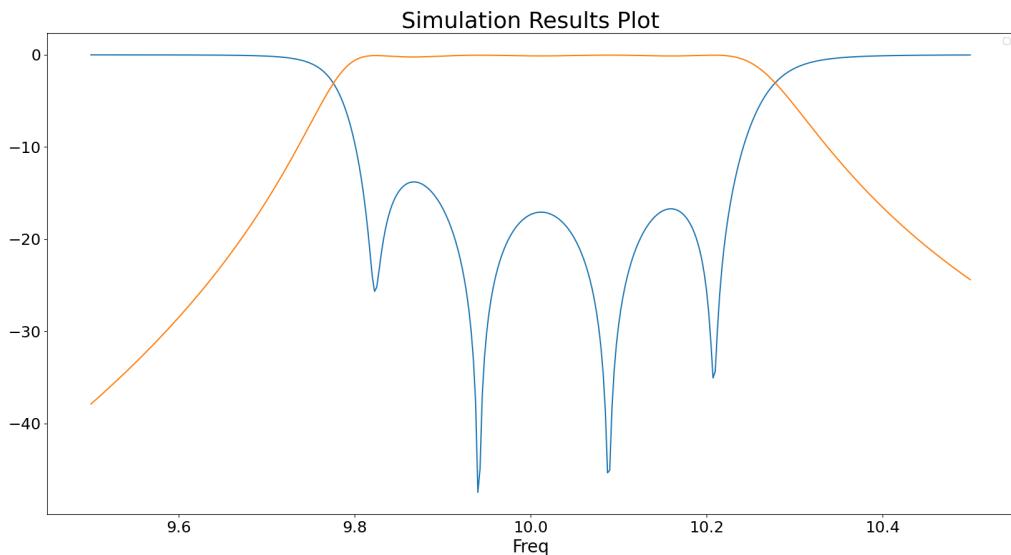
```
setup.analyze(tasks=2)
```

## Generate S-parameter plots

The following commands fetch solution data from HFSS for plotting directly from the Python interpreter.  
 Caution: The syntax for expressions must be identical to that used in HFSS.

```
traces_to_plot = hfss.get_traces_for_plot(second_element_filter="P1*")
report = hfss.post.create_report(traces_to_plot) # Creates a report in HFSS
solution = report.get_solution_data()

plt = solution.plot(solution.expressions) # Matplotlib axes object.
```

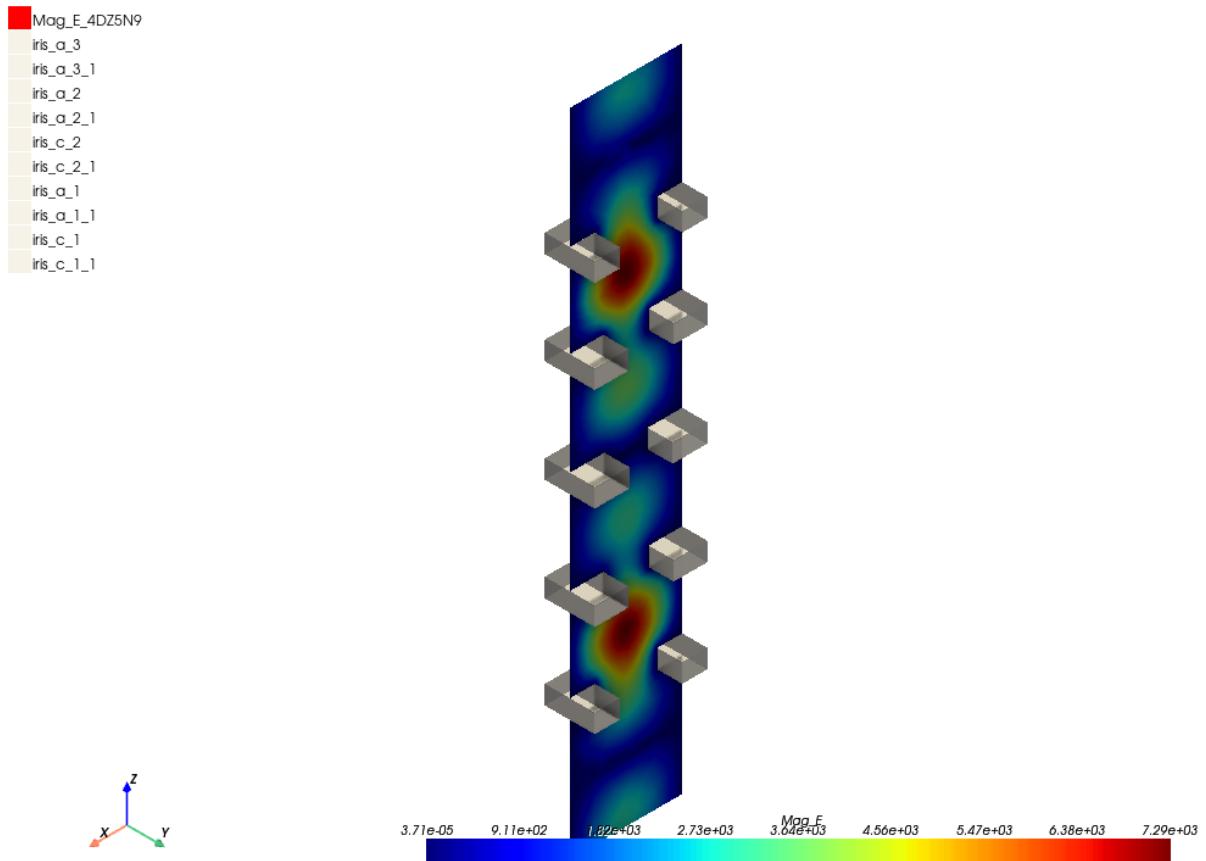


No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

## Generate E field plot

The following command generates a field plot in HFSS and uses PyVista to plot the field in Jupyter.

```
plot = hfss.post.plot_field(quantity="Mag_E", assignment=["Global:XZ"], plot_type=
 "CutPlane",
 setup=hfss.nominal_adaptive, intrinsics={"Freq": "9.8GHz",
 "Phase": "0deg"}, show=False,
 export_path=hfss.working_directory)
```



### Save and close the desktop

The following command saves the project to a file and closes the desktop.

```
hfss.save_project()
hfss.release_desktop()
```

```
True
```

**Total running time of the script:** (2 minutes 21.512 seconds)

### 3.12.5 SBR+ examples

These examples use PyAEDT to show some end-to-end workflows for HFSS SBR+. This includes model generation, setup, meshing, and postprocessing.

## SBR+: Import Geometry from Maps

This example shows how you can use PyAEDT to create an HFSS SBR+ project from an OpenStreetMaps.

### Perform required imports

Perform required imports and set up the local path to the PyAEDT directory path.

```
import os
from pyaedt import Hfss
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set non\_graphical either to True or False.

```
non_graphical = False
```

### Define designs

Define two designs, one source and one target. Each design is connected to a different object.

```
app = Hfss(
 designname="Ansys",
 solution_type="SBR+",
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical
)
```

### Define Location to import

Define latitude and longitude to import.

```
ansys_home = [40.273726, -80.168269]
```

## Generate map and import

Assign boundaries.

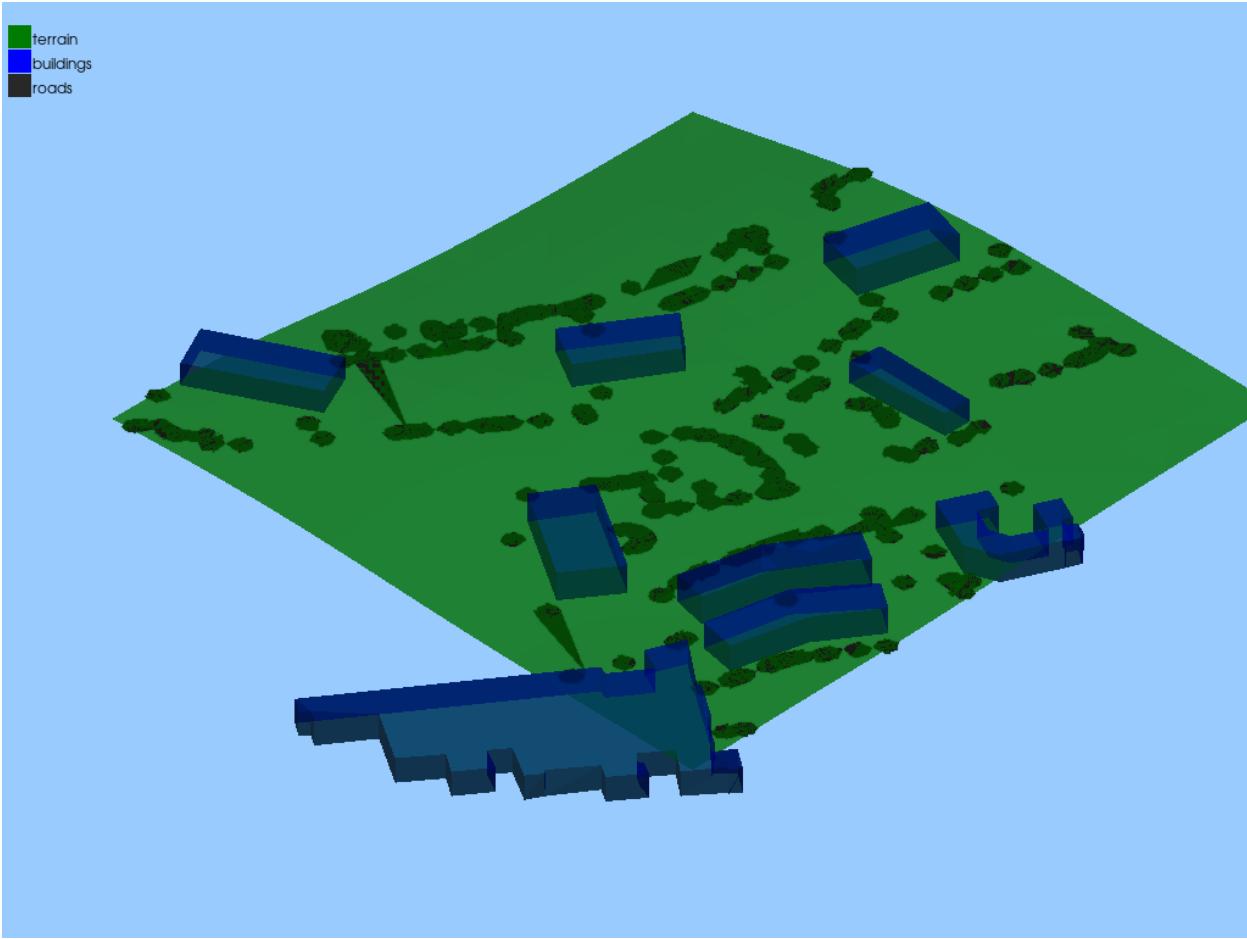
```
app.modeler.import_from_openstreet_map(ansys_home,
 terrain_radius=250,
 road_step=3,
 plot_before_importing=False,
 import_in_aedt=True)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pvista\core\filters\poly_
data.py:2942: PyVistaFutureWarning: The default value of the ``capping`` keyword_
argument will change in a future version to ``True`` to match the behavior of VTK. We_
recommend passing the keyword explicitly to prevent future surprises.
warnings.warn(
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\osmnx\graph.py:106:_
FutureWarning: The clean_periphery argument has been deprecated and will be removed in_
the v2.0.0 release. Future behavior will be as though clean_periphery=True. See the_
OSMnx v2 migration guide: https://github.com/gboeing/osmnx/issues/1123
G = graph_from_polygon(
{'name': 'default', 'version': 1, 'type': 'environment', 'center_lat_lon': [40.273726, -_
80.168269], 'radius': 250, 'include_buildings': True, 'include_roads': True, 'parts': {_
'terrain': {'file_name': 'C:/Users/ansys/Documents/Ansoft/Project839.pyaedt\\Ansyst_
terrain.stl', 'color': 'brown', 'material': 'earth'}, 'buildings': {'file_name': 'C:/_
Users/ansys/Documents/Ansoft/Project839.pyaedt\\Ansyst\\buildings.stl', 'color': 'grey',_
'material': 'concrete'}, 'roads': {'file_name': 'C:/Users/ansys/Documents/Ansoft/_'
Project839.pyaedt\\Ansyst\\roads.stl', 'color': 'black', 'material': 'asphalt'}}}
```

## Plot model

Plot the model.

```
plot_obj = app.plot(show=False, plot_air_objects=True)
plot_obj.background_color = [153, 203, 255]
plot_obj.zoom = 1.5
plot_obj.show_grid = False
plot_obj.show_axes = False
plot_obj.bounding_box = False
plot_obj.plot(os.path.join(app.working_directory, "Source.jpg"))
```



```
True
```

## Release AEDT

Release AEDT and close the example.

```
app.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 24.695 seconds)

## SBR+: doppler setup

This example shows how you can use PyAEDT to create a multipart scenario in HFSS SBR+ and set up a doppler analysis.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"

Launch AEDT
~~~~~
Launch AEDT.

projectname = "MicroDoppler_with_ADP"
designname = "doppler"
library_path = pyaedt.downloads.download_multiparts()
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

### Download and open project

Download and open the project.

```
project_name = pyaedt.generate_unique_project_name(project_name="doppler")

Instantiate the application.
app = pyaedt.Hfss(
 specified_version=aedt_version,
 solution_type="SBR+",
 new_desktop_session=True,
 projectname=project_name,
 close_on_exit=True,
 non_graphical=non_graphical
)

app.autosave_disable()
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
↳ subprocess 5676 is still running
 _warn("subprocess %s is still running" % self.pid,
```

```
True
```

## Save project and rename design

Save the project to the temporary folder and rename the design.

```
app.save_project()
app.rename_design(designname)
```

```
True
```

## Set up library paths

Set up library paths to 3D components.

```
actor_lib = os.path.join(library_path, "actor_library")
env_lib = os.path.join(library_path, "environment_library")
radar_lib = os.path.join(library_path, "radar_modules")
env_folder = os.path.join(env_lib, "road1")
person_folder = os.path.join(actor_lib, "person3")
car_folder = os.path.join(actor_lib, "vehicle1")
bike_folder = os.path.join(actor_lib, "bike1")
bird_folder = os.path.join(actor_lib, "bird1")
```

## Define environment

Define the background environment.

```
road1 = app.modeler.add_environment(input_dir=env_folder, name="Bari")
prim = app.modeler
```

## Place actors

Place actors in the environment. This code places persons, birds, bikes, and cars in the environment.

```
person1 = app.modeler.add_person(input_dir=person_folder, speed=1.0, global_offset=[25, -1.5, 0], yaw=180,
 name="Massimo")
person2 = app.modeler.add_person(input_dir=person_folder, speed=1.0, global_offset=[25, -2.5, 0], yaw=180, name="Devin")
car1 = app.modeler.add_vehicle(input_dir=car_folder, speed=8.7, global_offset=[3, -2.5, 0], name="LuxuryCar")
bike1 = app.modeler.add_vehicle(input_dir=bike_folder, speed=2.1, global_offset=[24, 3.6, 0])
(continues on next page)
```

(continued from previous page)

```

↪ 0], yaw=180,
 name="Alberto_in_bike")
bird1 = app.modeler.add_bird(input_dir=bird_folder, speed=1.0, global_offset=[19, 4, 3],
↪ yaw=120, pitch=-5,
 flapping_rate=30, name="Pigeon")
bird2 = app.modeler.add_bird(input_dir=bird_folder, speed=1.0, global_offset=[6, 2, 3],
↪ yaw=-60, pitch=10, name="Eagle")

```

## Place radar

Place radar on the car. The radar is created relative to the cars coordinate system.

```

radar1 = app.create_sbr_radar_from_json(radar_file=radar_lib, name="Example_1Tx_1Rx",
↪ offset=[2.57, 0, 0.54],
 use_relative_cs=True, relative_cs_name=car1.cs_
↪ name)

```

## Create setup

Create setup and validate it. The `create_sbr_pulse_doppler_setup` method creates a setup and a parametric sweep on the time variable with a duration of two seconds. The step is computed automatically from CPI.

```

setup, sweep = app.create_sbr_pulse_doppler_setup(sweep_time_duration=2)
app.set_sbr_current_sources_options()
app.validate_simple()

```

1

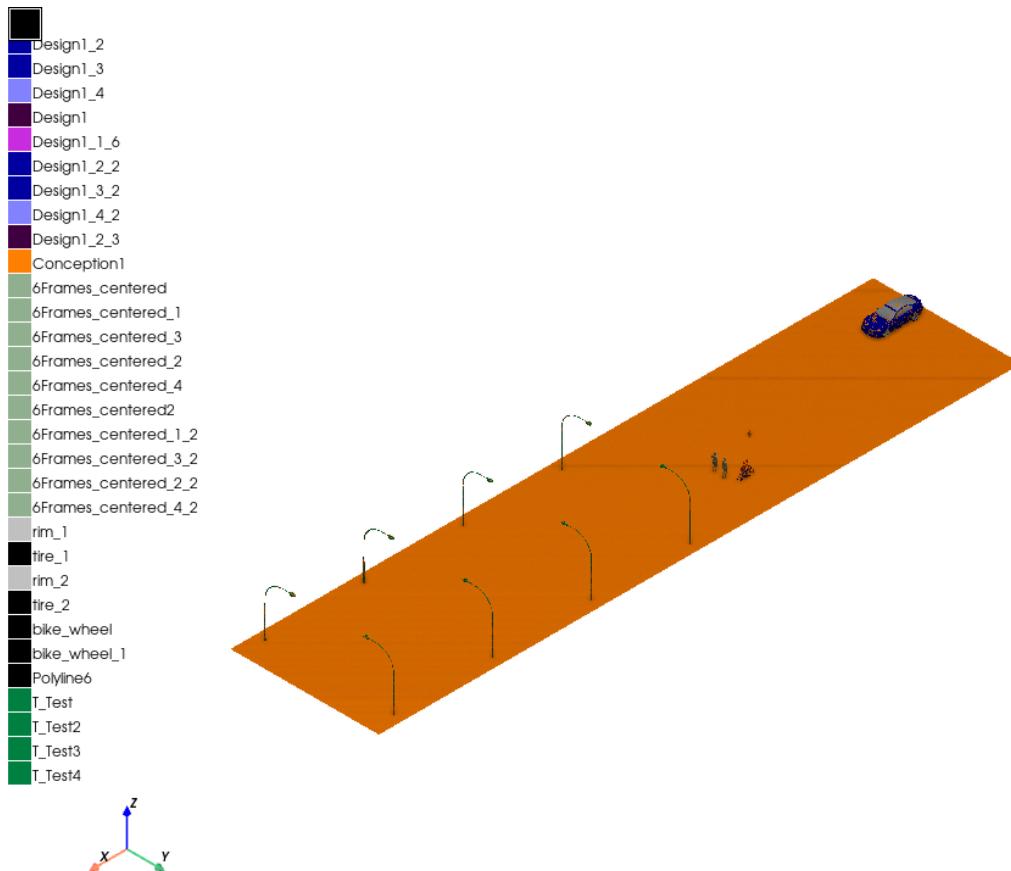
## Plot model

Plot the model.

```

app.plot(show=False, export_path=os.path.join(app.working_directory, "Image.jpg"), plot_
↪ air_objects=True)

```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258F248E8F0>
```

## Solve and release AEDT

Solve and release AEDT. To solve, uncomment the `app.analyze_setup` command to activate the simulation.

```
app.analyze_setup(sweep.name)
app.save_project()
app.release_desktop(close_projects=True, close_desktop=True)
```

```
True
```

**Total running time of the script:** (1 minutes 26.498 seconds)

## SBR+: HFSS to SBR+ coupling

This example shows how you can use PyAEDT to create an HFSS SBR+ project from an HFSS antenna and run a simulation.

### Perform required imports

Perform required imports and set up the local path to the path for the PyAEDT directory.

```
import os
import pyaedt

project_full_name = pyaedt.downloads.download_sbr(pyaedt.generate_unique_project_
 ↪name(project_name="sbr_freq"))
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

### Define designs

Define two designs, one source and one target, with each design connected to a different object.

```
target = pyaedt.Hfss(
 projectname=project_full_name,
 designname="Cassegrain_",
 solution_type="SBR+",
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical
)

source = pyaedt.Hfss(projectname=target.project_name,
 designname="feeder",
 specified_version=aedt_version,
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: ↴
 subprocess 7636 is still running
 _warn("subprocess %s is still running" % self.pid,
```

(continues on next page)

(continued from previous page)

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Define linked antenna

Define a linked antenna. This is HFSS far field applied to HFSS SBR+.

```
target.create_sbr_linked_antenna(source, target_cs="feederPosition", field_type="farfield")
<pyaedt.modules.Boundary.NativeComponentObject object at 0x00000258A9B085B0>
```

## Assign boundaries

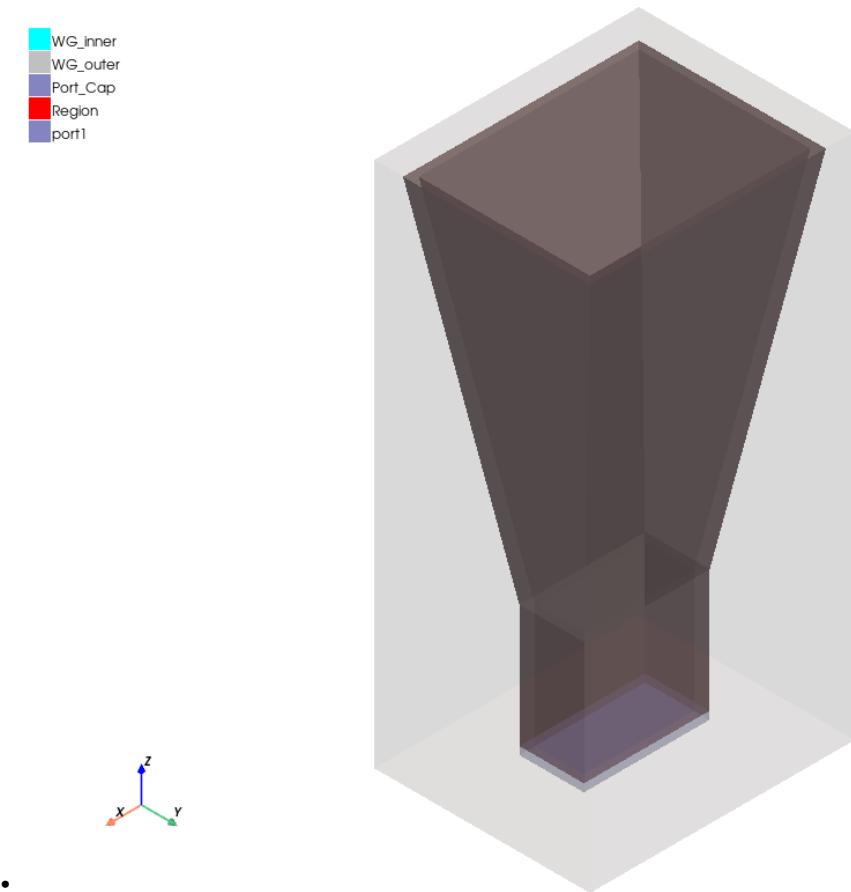
Assign boundaries.

```
target.assign_perfecte_to_sheets(["Reflector", "Subreflector"])
target.mesh.assign_curvilinear_elements(["Reflector", "Subreflector"])
<pyaedt.modules.Mesh.MeshOperation object at 0x00000258A9B0BD00>
```

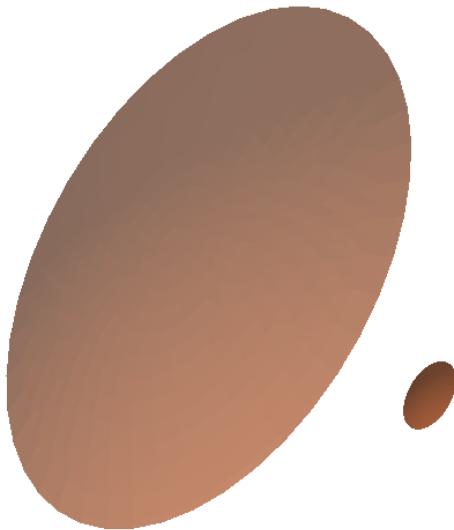
## Plot model

Plot the model

```
source.plot(show=False, export_path=os.path.join(target.working_directory, "Source.jpg"),
 plot_air_objects=True)
target.plot(show=False, export_path=os.path.join(target.working_directory, "Target.jpg"),
 plot_air_objects=False)
```



■ Reflector  
■ Subreflector



<pyaedt.generic.plot.ModelPlotter object at 0x00000258A9B0AFB0>

### Create setup and solve

Create a setup and solve it.

```
setup1 = target.create_setup()
setup1.props["RadiationSetup"] = "ATK_3D"
setup1.props["ComputeFarFields"] = True
setup1.props["RayDensityPerWavelength"] = 2
setup1.props["MaxNumberOfBounces"] = 3
setup1["RangeType"] = "SinglePoints"
setup1["RangeStart"] = "10GHz"
target.analyze()
```

True

## Plot results

Plot results.

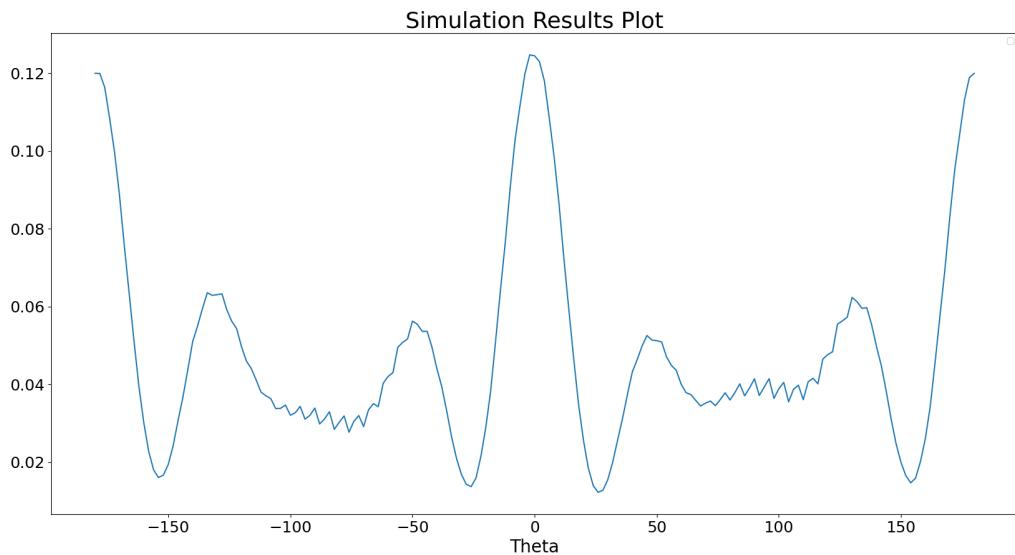
```
variations = target.available_variations.nominal_w_values_dict
variations["Freq"] = ["10GHz"]
variations["Theta"] = ["All"]
variations["Phi"] = ["All"]
target.post.create_report("db(GainTotal)", target.nominal_adaptive,
 variations=variations,
 primary_sweep_variable="Theta", report_category="Far Fields",
 context="ATK_3D")
```

```
<pyaedt.modules.report_templates.FarField object at 0x00000258A91C8D60>
```

## Plot results outside AEDT

Plot results using Matplotlib.

```
solution = target.post.get_solution_data(
 "GainTotal",
 target.nominal_adaptive,
 variations=variations,
 primary_sweep_variable="Theta",
 context="ATK_3D",
 report_category="Far Fields",
)
solution.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with  
an underscore are ignored when legend() is called with no argument.

(continues on next page)

(continued from previous page)

```
<Figure size 2000x1000 with 1 Axes>
```

## Release AEDT

Release AEDT and close the example.

```
target.release_desktop()
```

```
True
```

**Total running time of the script:** (2 minutes 22.777 seconds)

## 3.12.6 Maxwell examples

These examples use PyAEDT to show some end-to-end workflows for Maxwell 2D and Maxwell 3D. This includes model generation, setup, meshing, and postprocessing. Examples cover different Maxwell solution types (Eddy Current, Magnetostatic, and Transient).

### Maxwell 2D Electrostatic analysis

This example shows how you can use PyAEDT to create a Maxwell 2D electrostatic analysis. It shows how to create the geometry, load material properties from an Excel file and set up the mesh settings. Moreover, it focuses on post-processing operations, in particular how to plot field line traces, relevant for an electrostatic analysis.

#### Perform required imports

Perform required imports.

```
import pyaedt
```

#### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

#### Initialize Maxwell 2D

Initialize Maxwell 2D, providing the version, path to the project, and the design name and type.

```
setup_name = 'MySetupAuto'
solver = 'Electrostatic'
design_name = 'Design1'
project_name = pyaedt.generate_unique_project_name()
non_graphical = False
```

## Download .xlsx file

Set local temporary folder to export the .xlsx file to.

```
file_name_xlsx = pyaedt.downloads.download_file("field_line_traces", "my_copper.xlsx")
```

## Initialize dictionaries

Initialize dictionaries that contain all the definitions for the design variables.

```
geom_params_circle = {
 'circle_x0': '-10mm',
 'circle_y0': '0mm',
 'circle_z0': '0mm',
 'circle_axis': 'Z',
 'circle_radius': '1mm'
}

geom_params_rectangle = {
 'r_x0': '1mm',
 'r_y0': '5mm',
 'r_z0': '0mm',
 'r_axis': 'Z',
 'r_dx': '-1mm',
 'r_dy': '-10mm'
}
```

## Launch Maxwell 2D

Launch Maxwell 2D and save the project.

```
M2D = pyaedt.Maxwell2d(projectname=project_name,
 specified_version=aedt_version,
 designname=design_name,
 solution_type=solver,
 new_desktop_session=True,
 non_graphical=non_graphical
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 8912 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create object to access 2D modeler

Create the object mod2D to access the 2D modeler easily.

```
mod2D = M2D.modeler
mod2D.delete()
mod2D.model_units = "mm"
```

## Define variables from dictionaries

Define design variables from the created dictionaries.

```
for k, v in geom_params_circle.items():
 M2D[k] = v
for k, v in geom_params_rectangle.items():
 M2D[k] = v
```

## Read materials from .xlsx file

Read materials from .xlsx file into and set into design.

```
mats = M2D.materials.import_materials_from_excel(file_name_xlsx)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\modules\MaterialLib.
→py:907: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
→a future version, integer keys will always be treated as labels (consistent with
→DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
and val[keys.index(prop)]
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\modules\MaterialLib.
→py:908: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
→a future version, integer keys will always be treated as labels (consistent with
→DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
and not (isinstance(val[keys.index(prop)], float) and math.isnan(val[keys.
→index(prop)]))
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\modules\MaterialLib.
→py:910: FutureWarning: Series.__getitem__ treating keys as positions is deprecated. In
→a future version, integer keys will always be treated as labels (consistent with
→DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
props[prop] = float(val[keys.index(prop)])
```

## Create design geometries

Create rectangle and a circle and assign the material read from the .xlsx file. Create two new polylines and a region.

```
rect = mod2D.create_rectangle(origin=['r_x0', 'r_y0', 'r_z0'],
 sizes=['r_dx', 'r_dy', 0],
 name='Ground', matname=mats[0])
rect.color = (0, 0, 255) # rgb
rect.solve_inside = False
```

(continues on next page)

(continued from previous page)

```

circle = mod2D.create_circle(position=['circle_x0', 'circle_y0', 'circle_z0'], radius=
 ↪'circle_radius',
 num_sides='0', is_covered=True, name='Electrode', ↪
 ↪matname=mats[0])
circle.color = (0, 0, 255) # rgb
circle.solve_inside = False

poly1_points = [[-9, 2, 0], [-4, 2, 0], [2, -2, 0],[8, 2, 0]]
poly2_points = [[-9, 0, 0], [9, 0, 0]]
poly1_id = mod2D.create_polyline(points=poly1_points, segment_type='Spline', name='Poly1'
 ↪')
poly2_id = mod2D.create_polyline(points=poly2_points, name='Poly2')
mod2D.split([poly1_id, poly2_id], 'YZ', sides='NegativeOnly')
mod2D.create_region([20, 100, 20, 100])

```

<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BBA90130>

## Define excitations

Assign voltage excitations to rectangle and circle.

```

M2D.assign_voltage(rect.id, amplitude=0, name='Ground')
M2D.assign_voltage(circle.id, amplitude=50e6, name='50kV')

```

<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BBA93610>

## Create initial mesh settings

Assign a surface mesh to the rectangle.

```
M2D.mesh.assign_surface_mesh_manual(assignment=['Ground'], surface_deviation=0.001)
```

<pyaedt.modules.Mesh.MeshOperation object at 0x00000258BBA93250>

## Create, validate and analyze the setup

Create, update, validate and analyze the setup.

```

setup = M2D.create_setup(name=setup_name)
setup.props['PercentError'] = 0.5
setup.update()
M2D.validate_simple()
M2D.analyze_setup(setup_name)

```

True

## Evaluate the E Field tangential component

Evaluate the E Field tangential component along the given polylines. Add these operations to the Named Expression list in Field Calculator.

```
fields = M2D.ofieldsreporter
fields.CalcStack("clear")
fields.EnterQty("E")
fields.EnterEdge("Poly1")
fields.CalcOp("Tangent")
fields.CalcOp("Dot")
fields.AddNamedExpression("e_tan_poly1", "Fields")
fields.EnterQty("E")
fields.EnterEdge("Poly2")
fields.CalcOp("Tangent")
fields.CalcOp("Dot")
fields.AddNamedExpression("e_tan_poly2", "Fields")
```

## Create Field Line Traces Plot

Create Field Line Traces Plot specifying as seeding faces the ground, the electrode and the region and as In surface objects only the region.

```
plot = M2D.post.create_fieldplot_line_traces(seeding_faces=["Ground", "Electrode",
 ↪ "Region"],
 in_volume_tracing_objs="Region", plot_name=
 ↪ "LineTracesTest")
```

## Update Field Line Traces Plot

Update field line traces plot. Update seeding points number, line style and line width.

```
plot.SeedingPointsNumber = 20
plot.LineStyle = "Cylinder"
plot.LineWidth = 3
plot.update()
```

```
True
```

## Export field line traces plot

Export field line traces plot. For field lint traces plot, the export file format is .fldplt.

```
M2D.post.export_field_plot(plot_name="LineTracesTest", output_dir=M2D.toolkit_directory,
 ↪ file_format="fldplt")
```

```
'D:/Temp/pyaedt_prj_ITG/Project_551.pyaedt\\LineTracesTest.fldplt'
```

## Export the mesh field plot

Export the mesh in aedtplt format.

```
M2D.post.export_mesh_obj(setup=M2D.nominal_adaptive)
```

```
'D:/Temp/pyaedt_prj_ITG/Project_55J.pyaedt\\Design1\\Mesh_ZI95NS.aedtplt'
```

## Save project and close AEDT

Save the project and close AEDT.

```
M2D.save_project()
M2D.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 17.135 seconds)

## Maxwell 2D: PM synchronous motor transient analysis

This example shows how you can use PyAEDT to create a Maxwell 2D transient analysis for an interior permanent magnet electric motor.

### Perform required imports

Perform required imports.

```
from math import sqrt as mysqrt

import csv
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Initialize Maxwell 2D

Initialize Maxwell 2D, providing the version, path to the project, and the design name and type.

```
setup_name = "MySetupAuto"
solver = "TransientXY"

project_name = pyaedt.generate_unique_project_name()
design_name = "Sinusoidal"
```

## Initialize definitions for stator, rotor, and shaft

Initialize geometry parameter definitions for the stator, rotor, and shaft. The naming refers to RMxprt primitives.

```
geom_params = {
 "DiaGap": "132mm",
 "DiaStatorYoke": "198mm",
 "DiaStatorInner": "132mm",
 "DiaRotorLam": "130mm",
 "DiaShaft": "44.45mm",
 "DiaOuter": "198mm",
 "Airgap": "1mm",
 "SlotNumber": "48",
 "SlotType": "3"
}
```

## Initialize definitions for stator windings

Initialize geometry parameter definitions for the stator windings. The naming refers to RMxprt primitives.

```
wind_params = {
 "Layers": "1",
 "ParallelPaths": "2",
 "R_Phase": "7.5mOhm",
 "WdgExt_F": "5mm",
 "SpanExt": "30mm",
 "SegAngle": "0.25",
 "CoilPitch": "5", # coil pitch in slots
 "Coil_SetBack": "3.605732823mm",
 "SlotWidth": "2.814mm", # RMxprt Bs0
 "Coil_Edge_Short": "3.769235435mm",
 "Coil_Edge_Long": "15.37828521mm"
}
```

## Initialize definitions for model setup

Initialize geometry parameter definitions for the model setup.

```
mod_params = {
 "NumPoles": "8",
 "Model_Length": "80mm",
 "SymmetryFactor": "8",
 "Magnetic_Axial_Length": "150mm",
 "Stator_Lam_Length": "0mm",
 "StatorSkewAngle": "0deg",
 "NumTorquePointsPerCycle": "30",
 "mapping_angle": "0.125*4deg",
 "num_m": "16",
 "Section_Angle": "360deg/SymmetryFactor"
}
```

## Initialize definitions for operational machine

Initialize geometry parameter definitions for the operational machine. This identifies the operating point for the transient setup.

```
oper_params = {
 "InitialPositionMD": "180deg/4",
 "IPeak": "480A",
 "MachineRPM": "3000rpm",
 "ElectricFrequency": "MachineRPM/60rpm*NumPoles/2*1Hz",
 "ElectricPeriod": "1/ElectricFrequency",
 "BandTicksinModel": "360deg/NumPoles/mapping_angle",
 "TimeStep": "ElectricPeriod/(2*BandTicksinModel)",
 "StopTime": "ElectricPeriod",
 "Theta_i": "135deg"
}
```

## Set non-graphical mode

Set non-graphical mode. "PYAEDT\_NON\_GRAPHICAL" is needed to generate documentation only. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

## Launch Maxwell 2D

Launch Maxwell 2D and save the project.

```
M2D = pyaedt.Maxwell2d(projectname=project_name,
 specified_version=aedt_version,
 designname=design_name,
 solution_type=solver,
 new_desktop_session=True,
 non_graphical=non_graphical
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 7784 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create object to access 2D modeler

Create the object mod2D to access the 2D modeler easily.

```
mod2D = M2D.modeler
mod2D.delete()
mod2D.model_units = "mm"
```

## Define variables from dictionaries

Define design variables from the created dictionaries.

```
for k, v in geom_params.items():
 M2D[k] = v
for k, v in wind_params.items():
 M2D[k] = v
for k, v in mod_params.items():
 M2D[k] = v
for k, v in oper_params.items():
 M2D[k] = v
```

## Define path for non-linear material properties

Define the path for non-linear material properties. Materials are stored in text files.

```
filename_lam, filename_PM = pyaedt.downloads.download_leaf()
```

### Create first material

Create the material "Copper (Annealed)\_65C".

```
mat_coils = M2D.materials.add_material("Copper (Annealed)_65C")
mat_coils.update()
mat_coils.conductivity = "49288048.9198"
mat_coils.permeability = "1"
```

### Create second material

Create the material "Arnold\_Magnetics\_N30UH\_80C". The BH curve is read from a tabbed CSV file, and a list (BH\_List\_PM) is created. This list is passed to the mat\_PM.permeability.value method.

```
mat_PM = M2D.materials.add_material("Arnold_Magnetics_N30UH_80C_new")
mat_PM.update()
mat_PM.conductivity = "555555.5556"
mat_PM.set_magnetic_coercivity(value=-800146.66287534, x=1, y=0, z=0)
mat_PM.mass_density = "7500"
BH_List_PM = []
with open(filename_PM) as f:
 reader = csv.reader(f, delimiter='\t')
 next(reader)
 for row in reader:
 BH_List_PM.append([float(row[0]), float(row[1])])
mat_PM.permeability.value = BH_List_PM
```

### Create third material

Create the laminated material 30DH\_20C\_smooth. This material has a BH curve and a core loss model, which is set to electrical steel.

```
mat_lam = M2D.materials.add_material("30DH_20C_smooth")
mat_lam.update()
mat_lam.conductivity = "1694915.25424"
kh = 71.7180985413
kc = 0.25092214579
ke = 12.1625774023
kdc = 0.001
eq_depth = 0.001
mat_lam.set_electrical_steel_coreloss(kh, kc, ke, kdc, eq_depth)
mat_lam.mass_density = "7650"
BH_List_lam = []
```

(continues on next page)

(continued from previous page)

```

with open(filename_lam) as f:
 reader = csv.reader(f, delimiter='\t')
 next(reader)
 for row in reader:
 BH_List_lam.append([float(row[0]), float(row[1])])
mat_lam.permeability.value = BH_List_lam

```

## Create geometry for stator

Create the geometry for the stator. It is created via the RMxprt user-defined primitive. A list of lists is created with the proper UDP parameters.

```

udp_par_list_stator = [[["DiaGap", "DiaGap"], ["DiaYoke", "DiaStatorYoke"], ["Length",
→ "Stator_Lam_Length"], ["Skew", "StatorSkewAngle"], ["Slots", "SlotNumber"], ["SlotType",
→ "SlotType"], ["Hs0", "1.2mm"], ["Hs01", "0mm"], ["Hs1", "0.4834227384999mm"], ["Hs2",
"17.287669825502mm"], ["Bs0", "2.814mm"], ["Bs1", "4.71154109036mm"], ["Bs2", "6.
→ 9777285790998mm"], ["Rs", "2mm"], ["FilletType", "1"], ["HalfSlot", "0"], ["VentHoles", "0"], [
→ "HoleDiaIn", "0mm"], ["HoleDiaOut", "0mm"], ["HoleLocIn", "0mm"], ["HoleLocOut", "0mm"], ["VentDucts", "0"], [
→ "DuctWidth", "0mm"], ["DuctPitch", "0mm"], ["SegAngle", "0deg"], ["LenRegion", "Model_Length"], ["InfoCore",
→ "0"]]

stator_id = mod2D.create_udp(dll="RMxprt/VentSlotCore.dll", parameters=udp_par_list_
→ stator, library='syslib',
 name='my_stator') # name not taken

```

## Assign properties to stator

Assign properties to the stator. The following code assigns the material, name, color, and `solve_inside` properties.

```

M2D.assign_material(obj=stator_id, mat="30DH_20C_smooth")
stator_id.name = "Stator"
stator_id.color = (0, 0, 255) # rgb
stator_id.solve_inside = True # to be reassigned: M2D.assign material puts False if not_
→ dielectric

```

## Create geometry for PMs

Create the geometry for the PMs (permanent magnets). In Maxwell 2D, you assign magnetization via the coordinate system. Because each PM needs to have a coordinate system in the face center, auxiliary functions are created. Here, you use the auxiliary function `find_elements(lst1, lst2)` to find the elements in list `lst1` with indexes in list `lst2`.

```
def find_elements(lst1, lst2):
 return [lst1[i] for i in lst2]
```

## Find largest elements in list

Use the auxiliary function `find_n_largest(input_len_list, n_largest_edges)` to find the `n` largest elements in the list `input_len_list`.

```
def find_n_largest(input_len_list, n_largest_edges):
 tmp = list(input_len_list)
 copied = list(input_len_list)
 copied.sort() # sort list so that largest elements are on the far right
 index_list = []
 for n in range(1, n_largest_edges + 1): # get index of the nth largest element
 index_list.append(tmp.index(copied[-n]))
 tmp[tmp.index(copied[-n])] = 0 # index can only get the first occurrence that
 ↪solves the problem
 return index_list
```

## Create coordinate system for PMs

Create the coordinate system for the PMs. The inputs are the object name, coordinate system name, and inner or outer magnetization. Find the two longest edges of the magnets and get the midpoint of the outer edge. You must have this point to create the face coordinate systems in case of outer magnetization.

```
def create_cs_magnets(pm_id, cs_name, point_direction):
 pm_face_id = mod2D.get_object_faces(pm_id.name)[0] # works with name only
 pm_edges = mod2D.get_object_edges(pm_id.name) # gets the edges of the PM object
 edge_len_list = list(
 map(mod2D.get_edge_length, pm_edges)) # apply method get_edge_length to all
 ↪elements of list pm_edges
 index_2_longest = find_n_largest(edge_len_list, 2) # find the 2 longest edges of
 ↪the PM
 longest_edge_list = find_elements(pm_edges, index_2_longest)
 edge_center_list = list(map(mod2D.get_edge_midpoint,
 longest_edge_list)) # apply method get_edge_midpoint to
 ↪all elements of list longest_edge_list

 rad = lambda x: mysqrt(x[0] * x[0] + x[1] * x[1] + x[2] * x[2])
 index_largest_r = find_n_largest(list(map(rad, edge_center_list)), 2)
 longest_edge_list2 = [longest_edge_list[i] for i in index_largest_r] # reorder:
 ↪outer first element of the list
 if point_direction == 'outer':
```

(continues on next page)

(continued from previous page)

```

my_axis_pos = longest_edge_list2[0]
elif point_direction == 'inner':
 my_axis_pos = longest_edge_list2[1]

mod2D.create_face_coordinate_system(face=pm_face_id, origin=pm_face_id, axis_
position=my_axis_pos,
 axis="X", name=cs_name)
pm_id.part_coordinate_system = cs_name
mod2D.set_working_coordinate_system('Global')

```

## Create outer and inner PMs

Create the outer and inner PMs and assign color to them.

```

IM1_points = [[56.70957112, 3.104886585, 0], [40.25081875, 16.67243502, 0], [38.59701538,
← 14.66621111, 0],
 [55.05576774, 1.098662669, 0]]
OM1_points = [[54.37758185, 22.52393189, 0], [59.69688156, 9.68200639, 0], [63.26490432, ←
11.15992981, 0],
 [57.94560461, 24.00185531, 0]]
IPM1_id = mod2D.create_polyline(points=IM1_points, cover_surface=True, name="PM_I1",
 material="Arnold_Magnetics_N30UH_80C_new")
IPM1_id.color = (0, 128, 64)
OPM1_id = mod2D.create_polyline(points=OM1_points, cover_surface=True, name="PM_O1",
 material="Arnold_Magnetics_N30UH_80C_new")
OPM1_id.color = (0, 128, 64)

```

## Create coordinate system for PMs in face center

Create the coordinate system for PMs in the face center.

```

create_cs_magnets(IPM1_id, 'CS_' + IPM1_id.name, 'outer')
create_cs_magnets(OPM1_id, 'CS_' + OPM1_id.name, 'outer')

```

## Duplicate and mirror PMs

Duplicate and mirror the PMs along with the local coordinate system.

```

mod2D.duplicate_and_mirror([IPM1_id, OPM1_id], origin=[0, 0, 0],
 vector=["cos((360deg/SymmetryFactor/2)+90deg)", "sin((360deg/
 ←SymmetryFactor/2)+90deg)", 0])
id_PM = mod2D.get_objects_w_string("PM", case_sensitive=True)

```

## Create coils

Create the coils.

```
coil_id = mod2D.create_rectangle(origin=['DiaRotorLam/2+Airgap+Coil_SetBack', '-Coil_Edge_Short/2', 0],
 sizes=['Coil_Edge_Long', 'Coil_Edge_Short', 0],
 name='Coil', material='Copper (Annealed)_65C')
coil_id.color = (255, 128, 0)
M2D.modeler.rotate(assignment=coil_id, axis="Z", angle="360deg/SlotNumber/2")
coil_id.duplicate_around_axis(axis="Z", angle="360deg/SlotNumber", nclones='CoilPitch+1',
 create_new_objects=True)
id_coils = mod2D.get_objects_w_string("Coil", case_sensitive=True)
```

## Create shaft and region

Create the shaft and region.

```
region_id = mod2D.create_circle(position=[0, 0, 0], radius='DiaOuter/2',
 num_sides='SegAngle', is_covered=True, name='Region')
shaft_id = mod2D.create_circle(position=[0, 0, 0], radius='DiaShaft/2',
 num_sides='SegAngle', is_covered=True, name='Shaft')
```

## Create bands

Create the inner band, band, and outer band.

```
bandIN_id = mod2D.create_circle(position=[0, 0, 0], radius='(DiaGap - (1.5 * Airgap))/2',
 num_sides='mapping_angle', is_covered=True, name='Inner_Band')
bandMID_id = mod2D.create_circle(position=[0, 0, 0], radius='(DiaGap - (1.0 * Airgap))/2',
 num_sides='mapping_angle', is_covered=True, name='Band')
bandOUT_id = mod2D.create_circle(position=[0, 0, 0], radius='(DiaGap - (0.5 * Airgap))/2',
 num_sides='mapping_angle', is_covered=True, name='Outer_Band')
```

## Assign motion setup to object

Assign a motion setup to a Band object named RotatingBand\_mid.

```
M2D.assign_rotate_motion(assignment='Band', coordinate_system="Global", axis="Z",
 positive_movement=True,
 start_position="InitialPositionMD", angular_velocity="MachineRPM")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258AAEE9C90>
```

## Create list of vacuum objects

Create a list of vacuum objects and assign color.

```
vacuum_obj_id = [shaft_id, region_id, bandIN_id, bandMID_id, bandOUT_id] # put shaft
 ↪ first
for item in vacuum_obj_id:
 item.color = (128, 255, 255)
```

## Create rotor

Create the rotor. Holes are specific to the lamination. Allocated PMs are created.

```
rotor_id = mod2D.create_circle(position=[0, 0, 0], radius='DiaRotorLam/2',
 num_sides=0, name="Rotor", material="30DH_20C_smooth")
rotor_id.color = (0, 128, 255)
mod2D.subtract(rotor_id, shaft_id, keep_originals=True)
void_small_1_id = mod2D.create_circle(position=[62, 0, 0], radius="2.55mm",
 num_sides=0, name="void1", material="vacuum")
M2D.modeler.duplicate_around_axis(void_small_1_id, axis="Z", angle="360deg/SymmetryFactor
 ↪",
 clones=2, create_new_objects=False)
void_big_1_id = mod2D.create_circle(position=[29.5643, 12.234389332712, 0], radius='9.
 ↪88mm/2',
 num_sides=0, name="void_big", material="vacuum")
mod2D.subtract(rotor_id, [void_small_1_id, void_big_1_id], keep_originals=False)

slot_IM1_points = [[37.5302872, 15.54555396, 0], [55.05576774, 1.098662669, 0], [57.
 ↪33637589, 1.25, 0],
 [57.28982158, 2.626565019, 0], [40.25081875, 16.67243502, 0]]
slot_OM1_points = [[54.37758185, 22.52393189, 0], [59.69688156, 9.68200639, 0], [63.
 ↪53825619, 10.5, 0],
 [57.94560461, 24.00185531, 0]]
slot_IM_id = mod2D.create_polyline(points=slot_IM1_points, cover_surface=True, name=
 ↪"slot_IM1", material="vacuum")
slot_OM_id = mod2D.create_polyline(points=slot_OM1_points, cover_surface=True, name=
 ↪"slot_OM1", material="vacuum")

M2D.modeler.duplicate_and_mirror(assignment=[slot_IM_id, slot_OM_id], origin=[0, 0, 0],
 vector=["cos((360deg/SymmetryFactor/2)+90deg)",
 "sin((360deg/SymmetryFactor/2)+90deg)", 0])

id_holes = mod2D.get_objects_w_string("slot_", case_sensitive=True)
M2D.modeler.subtract(rotor_id, id_holes, keep_originals=True)
```

True

## Create section of machine

Create a section of the machine. This allows you to take advantage of symmetries.

```
object_list = [stator_id, rotor_id] + vacuum_obj_id
mod2D.create_coordinate_system(origin=[0, 0, 0],
 reference_cs="Global",
 name="Section",
 mode="axis",
 x_pointing=["cos(360deg/SymmetryFactor)", "sin(360deg/
→SymmetryFactor)", 0],
 y_pointing=["-sin(360deg/SymmetryFactor)", "cos(360deg/
→SymmetryFactor)", 0])

mod2D.set_working_coordinate_system("Section")
mod2D.split(object_list, "ZX", sides="NegativeOnly")
mod2D.set_working_coordinate_system("Global")
mod2D.split(object_list, "ZX", sides="PositiveOnly")
```

```
['Stator', 'Rotor', 'Shaft', 'Region', 'Inner_Band', 'Band', 'Outer_Band']
```

## Create boundary conditions

Create independent and dependent boundary conditions. Edges for assignment are picked by position. The points for edge picking are in the airgap.

```
pos_1 = "(DiaGap - (1.0 * Airgap))/4"
id_bc_1 = mod2D.get_edgeid_from_position(position=[pos_1, 0, 0], assignment='Region')
id_bc_2 = mod2D.get_edgeid_from_position(
 position=[pos_1 + "*cos((360deg/SymmetryFactor))", pos_1 + "*sin((360deg/
→SymmetryFactor))", 0], assignment='Region')
M2D.assign_master_slave(independent=id_bc_1, dependent=id_bc_2, reverse_master=False,
→reverse_slave=True,
 same_as_master=False, boundary="Matching")
```

```
(<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE2F3100>, <pyaedt.modules.
→Boundary.BoundaryObject object at 0x00000258BE2F35B0>)
```

## Assign vector potential

Assign a vector potential of 0 to the second position.

```
pos_2 = "(DiaOuter/2)"
id_bc_az = mod2D.get_edgeid_from_position(
 position=[pos_2 + "*cos((360deg/SymmetryFactor/2))", pos_2 + "*sin((360deg/
→SymmetryFactor)/2)", 0],
 assignment='Region')
M2D.assign_vector_potential(id_bc_az, vector_value=0, boundary="VectorPotentialZero")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x000000258BE2F32B0>
```

## Create excitations

Create excitations, defining phase currents for the windings.

```

PhA_current = "IPeak * cos(2*pi*ElectricFrequency*time+Theta_i)"
PhB_current = "IPeak * cos(2*pi * ElectricFrequency*time - 120deg+Theta_i)"
PhC_current = "IPeak * cos(2*pi * ElectricFrequency*time - 240deg+Theta_i)"

```

## Define windings in phase A

Define windings in phase A.

```
M2D.assign_coil(assignment=["Coil"], conductors_number=6, polarity="Positive", name="CT_Ph1_P2_C1_Go")
M2D.assign_coil(assignment=["Coil_5"], conductors_number=6, polarity="Negative", name="CT_Ph1_P2_C1_Ret")
M2D.assign_winding(assignment=None, winding_type="Current", is_solid=False, current=PhA_current, parallel_branches=1, name="Phase_A")
M2D.add_winding_coils(assignment="Phase_A", coils=["CT_Ph1_P2_C1_Go", "CT_Ph1_P2_C1_Ret"])
```

True

## Define windings in phase B

Define windings in phase B.

```

M2D.assign_coil(assignment="Coil_3", conductors_number=6, polarity="Positive", name="CT_Ph3_P1_C2_Go")
M2D.assign_coil(assignment="Coil_4", conductors_number=6, polarity="Positive", name="CT_Ph3_P1_C1_Go")
M2D.assign_winding(assignment=None, winding_type="Current", is_solid=False, current=PhB_current,
 parallel_branches=1,
 name="Phase_B")
M2D.add_winding_coils(assignment="Phase_B", coils=["CT_Ph3_P1_C2_Go", "CT_Ph3_P1_C1_Go"])

```

True

## Define windings in phase C

Define windings in phase C.

```
M2D.assign_coil(assignment="Coil_1", conductors_number=6, polarity="Negative", name="CT_
↪Ph2_P2_C2_Ret")
M2D.assign_coil(assignment="Coil_2", conductors_number=6, polarity="Negative", name="CT_
↪Ph2_P2_C1_Ret")
M2D.assign_winding(assignment=None, winding_type="Current", is_solid=False, current=PhC_
↪current, parallel_branches=1,
↪name="Phase_C")
M2D.add_winding_coils(assignment="Phase_C", coils=["CT_Ph2_P2_C2_Ret", "CT_Ph2_P2_C1_Ret
↪"])
```

True

## Assign total current on PMs

Assign a total current of 0 on the PMs.

```
PM_list = id_PMs
for item in PM_list:
 M2D.assign_current(item, amplitude=0, solid=True, name=item + "_I0")
```

## Create mesh operations

Create the mesh operations.

```
M2D.mesh.assign_length_mesh(id_coils, inside_selection=True, maximum_length=3, maximum_
↪elements=None, name="coils")
M2D.mesh.assign_length_mesh(stator_id, inside_selection=True, maximum_length=3, maximum_
↪elements=None, name="stator")
M2D.mesh.assign_length_mesh(rotor_id, inside_selection=True, maximum_length=3, maximum_
↪elements=None, name="rotor")
```

<pyaedt.modules.Mesh.MeshOperation object at 0x00000258BE27C550>

## Turn on eddy effects

Turn on eddy effects.

```
M2D.eddy_effects_on(eddy_effects_list, activate_eddy_effects=True, activate_
↪displacement_current=False)
```

## Turn on core loss

Turn on core loss.

```
core_loss_list = ['Rotor', 'Stator']
M2D.set_core_losses(core_loss_list)
```

```
True
```

## Compute transient inductance

Compute the transient inductance.

```
M2D.change_inductance_computation(compute_transient_inductance=True, incremental_
matrix=False)
```

```
True
```

## Set model depth

Set the model depth.

```
M2D.model_depth = "Magnetic_Axial_Length"
```

## Set symmetry factor

Set the symmetry factor.

```
M2D.change_symmetry_multiplier("SymmetryFactor")
```

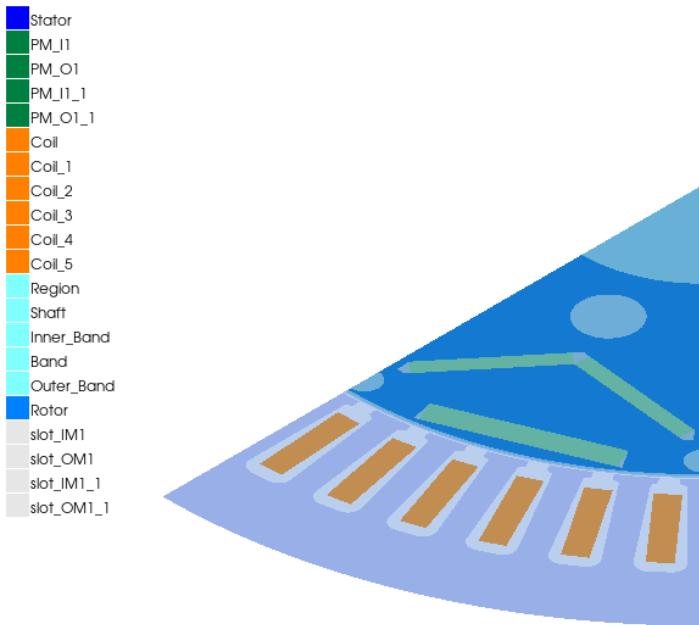
```
True
```

## Create setup and validate

Create the setup and validate it.

```
setup = M2D.create_setup(name=setup_name)
setup.props["StopTime"] = "StopTime"
setup.props["TimeStep"] = "TimeStep"
setup.props["SaveFieldsType"] = "None"
setup.props["OutputPerObjectCoreLoss"] = True
setup.props["OutputPerObjectSolidLoss"] = True
setup.props["OutputError"] = True
setup.update()
M2D.validate_simple()

model = M2D.plot(show=False)
model.plot(os.path.join(M2D.working_directory, "Image.jpg"))
```



True

## Initialize definitions for output variables

Initialize the definitions for the output variables. These will be used later to generate reports.

```
output_vars = {
 "Current_A": "InputCurrent(Phase_A)",
 "Current_B": "InputCurrent(Phase_B)",
 "Current_C": "InputCurrent(Phase_C)",
 "Flux_A": "FluxLinkage(Phase_A)",
 "Flux_B": "FluxLinkage(Phase_B)",
 "Flux_C": "FluxLinkage(Phase_C)",
 "pos": "(Moving1.Position -InitialPositionMD) *NumPoles/2",
 "cos0": "cos(pos)",
 "cos1": "cos(pos-2*PI/3)",
 "cos2": "cos(pos-4*PI/3)",
 "sin0": "sin(pos)",
 "sin1": "sin(pos-2*PI/3)",
 "sin2": "sin(pos-4*PI/3)",
 "Flux_d": "2/3*(Flux_A*cos0+Flux_B*cos1+Flux_C*cos2)",
 "Flux_q": "-2/3*(Flux_A*sin0+Flux_B*sin1+Flux_C*sin2)",
```

(continues on next page)

(continued from previous page)

```

 "I_d": "2/3*(Current_A*cos0 + Current_B*cos1 + Current_C*cos2)",
 "I_q": "-2/3*(Current_A*sin0 + Current_B*sin1 + Current_C*sin2)",
 "Irms": "sqrt(I_d^2+I_q^2)/sqrt(2)",
 "ArmatureOhmicLoss_DC": "Irms^2*R_phase",
 "Lad": "L(Phase_A,Phase_A)*cos0 + L(Phase_A,Phase_B)*cos1 + L(Phase_A,Phase_C)*cos2",
 "Laq": "L(Phase_A,Phase_A)*sin0 + L(Phase_A,Phase_B)*sin1 + L(Phase_A,Phase_C)*sin2",
 "Lbd": "L(Phase_B,Phase_A)*cos0 + L(Phase_B,Phase_B)*cos1 + L(Phase_B,Phase_C)*cos2",
 "Lbq": "L(Phase_B,Phase_A)*sin0 + L(Phase_B,Phase_B)*sin1 + L(Phase_B,Phase_C)*sin2",
 "Lcd": "L(Phase_C,Phase_A)*cos0 + L(Phase_C,Phase_B)*cos1 + L(Phase_C,Phase_C)*cos2",
 "Lcq": "L(Phase_C,Phase_A)*sin0 + L(Phase_C,Phase_B)*sin1 + L(Phase_C,Phase_C)*sin2",
 "L_d": "(Lad*cos0 + Lbd*cos1 + Lcd*cos2) * 2/3",
 "L_q": "(Laq*sin0 + Lbq*sin1 + Lcq*sin2) * 2/3",
 "OutputPower": "Moving1.Speed*Moving1.Torque",
 "Ui_A": "InducedVoltage(Phase_A)",
 "Ui_B": "InducedVoltage(Phase_B)",
 "Ui_C": "InducedVoltage(Phase_C)",
 "Ui_d": "2/3*(Ui_A*cos0 + Ui_B*cos1 + Ui_C*cos2)",
 "Ui_q": "-2/3*(Ui_A*sin0 + Ui_B*sin1 + Ui_C*sin2)",
 "U_A": "Ui_A+R_Phase*Current_A",
 "U_B": "Ui_B+R_Phase*Current_B",
 "U_C": "Ui_C+R_Phase*Current_C",
 "U_d": "2/3*(U_A*cos0 + U_B*cos1 + U_C*cos2)",
 "U_q": "-2/3*(U_A*sin0 + U_B*sin1 + U_C*sin2)"
}

```

## Create output variables for postprocessing

Create output variables for postprocessing.

```

for k, v in output_vars.items():
 M2D.create_output_variable(k, v)

```

## Initialize definition for postprocessing plots

Initialize the definition for postprocessing plots.

```

post_params = {
 "Moving1.Torque": "TorquePlots"
}

```

## Initialize definition for postprocessing multiplots

Initialize the definition for postprocessing multiplots.

```
post_params_multiplot = { # reports
 ("U_A", "U_B", "U_C", "Ui_A", "Ui_B", "Ui_C"): "PhaseVoltages",
 ("CoreLoss", "SolidLoss", "ArmatureOhmicLoss_DC"): "Losses",
 ("InputCurrent(Phase_A)", "InputCurrent(Phase_B)", "InputCurrent(Phase_C)": "PhaseCurrents",
 ("FluxLinkage(Phase_A)", "FluxLinkage(Phase_B)", "FluxLinkage(Phase_C)": "PhaseFluxes",
 ("I_d", "I_q"): "Currents_dq",
 ("Flux_d", "Flux_q"): "Fluxes_dq",
 ("Ui_d", "Ui_q"): "InducedVoltages_dq",
 ("U_d", "U_q"): "Voltages_dq",
 ("L(Phase_A,Phase_A)", "L(Phase_B,Phase_B)", "L(Phase_C,Phase_C)", "L(Phase_A,Phase_B)", "L(Phase_A,Phase_C)",
 "L(Phase_B,Phase_C)": "PhaseInductances",
 ("L_d", "L_q"): "Inductances_dq",
 ("CoreLoss", "CoreLoss(Stator)", "CoreLoss(Rotor)": "CoreLosses",
 ("EddyCurrentLoss", "EddyCurrentLoss(Stator)", "EddyCurrentLoss(Rotor)": "EddyCurrentLosses (Core)",
 ("ExcessLoss", "ExcessLoss(Stator)", "ExcessLoss(Rotor)": "ExcessLosses (Core)",
 ("HysteresisLoss", "HysteresisLoss(Stator)", "HysteresisLoss(Rotor)": "HysteresisLosses (Core)",
 ("SolidLoss", "SolidLoss(IPM1)", "SolidLoss(IPM1_1)", "SolidLoss(OPM1)",
 "SolidLoss(OPM1_1)": "SolidLoss"
}
```

## Create report

Create a report.

```
for k, v in post_params.items():
 M2D.post.create_report(expressions=k, setup_sweep_name="", domain="Sweep", variations=None,
 report_category=None, primary_sweep_variable="Time", secondary_sweep_variable=None,
 polyline_points=1001, plot_type="Rectangular Plot", context=None, subdesign_id=None,
 plot_name=v)
```

## Create multiplot report

Create a multiplot report.

```
for k, v in post_params_multiplot.items():
M2D.post.create_report(expressions=list(k), setup_sweep_name="", domain="Sweep",
↪variations=None,
primary_sweep_variable="Time", secondary_sweep_
↪variable=None,
report_category=None, plot_type="Rectangular Plot",
↪context=None, subdesign_id=None,
polyline_points=1001, plotname=v)
```

## Analyze and save project

Analyze and save the project.

```
M2D.save_project()
M2D.analyze_setup(setup_name, use_auto_settings=False)
```

```
True
```

## Create flux lines plot on region

Create a flux lines plot on a region. The object\_list is formerly created when the section is applied.

```
faces_reg = mod2D.get_object_faces(object_list[1].name) # Region
plot1 = M2D.post.create_fieldplot_surface(assignment=faces_reg, quantity='Flux_Lines',
 ↪intrinsics={
 "Time": M2D.variable_manager.variables["StopTime"].evaluated_value}, plot_name="Flux_
 ↪Lines")
```

## Export a field plot to an image file

Export the flux lines plot to an image file using Python PyVista.

```
M2D.post.plot_field_from_fieldplot(plot1.name, show=False)
```

```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BBA92020>
```

## Get solution data

Get a simulation result from a solved setup and cast it in a `SolutionData` object. Plot the desired expression by using `Matplotlib plot()`.

```
solutions = M2D.post.get_solution_data(expressions="Moving1.Torque",
 primary_sweep_variable="Time")
#solutions.plot()
```

## Retrieve the data magnitude of an expression

List of shaft torque points and compute average.

```
mag = solutions.data_magnitude()
avg = sum(mag) / len(mag)
```

## Export a report to a file

Export a 2D Plot data to a .csv file.

```
M2D.post.export_report_to_file(output_dir=M2D.toolkit_directory,
 plot_name="TorquePlots",
 extension=".csv")
```

```
'D:/Temp/pyaedt_prj_NKY/Project_ZGZ.pyaedt\\TorquePlots.csv'
```

## Close AEDT

Close AEDT.

```
M2D.release_desktop()
```

```
True
```

**Total running time of the script:** (2 minutes 51.270 seconds)

## Transformer leakage inductance calculation in Maxwell 2D Magnetostatic

This example shows how you can use pyAEDT to create a Maxwell 2D magnetostatic analysis to calculate transformer leakage inductance and reactance. The analysis based on this document form page 8 on: <https://www.ee.iitb.ac.in/~fclab/FEM/FEM1.pdf>

## Perform required imports

```
import tempfile
from pyaedt import Maxwell2d

temp_dir = tempfile.TemporaryDirectory(suffix=".ansys")
```

## Initialize and launch Maxwell 2D

Initialize and launch Maxwell 2D, providing the version, path to the project, and the design name and type.

```
non_graphical = False

project_name = "Transformer_leakage_inductance"
design_name = "1_Magnetostatic"
solver = "MagnetostaticXY"
desktop_version = "2024.1"

m2d = Maxwell2d(specified_version=desktop_version,
 new_desktop_session=False,
 designname=design_name,
 projectname=project_name,
 solution_type=solver,
 non_graphical=non_graphical)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 4480 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 _py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 _log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Initialize dictionaries

Initialize dictionaries that contain all the definitions for the design variables.

```
mod = m2d.modeler
mod.model_units = "mm"

dimensions = {
 "core_width": "1097mm",
 "core_height": "2880mm",
 "core_opening_x1": "270mm",
 "core_opening_x2": "557mm",
 "core_opening_y1": "540mm",
 "core_opening_y2": "2340mm",
 "core_opening_width": "core_opening_x2-core_opening_x1",
 "core_opening_height": "core_opening_y2-core_opening_y1",
 "LV_x1": "293mm",
 "LV_x2": "345mm",
 "LV_width": "LV_x2-LV_x1",
```

(continues on next page)

(continued from previous page)

```

"LV_mean_radius": "LV_x1+LV_width/2",
"LV_mean_turn_length": "pi*2*LV_mean_radius",
"LV_y1": "620mm",
"LV_y2": "2140mm",
"LV_height": "LV_y2-LV_y1",
"HV_x1": "394mm",
"HV_x2": "459mm",
"HV_width": "HV_x2-HV_x1",
"HV_mean_radius": "HV_x1+HV_width/2",
"HV_mean_turn_length": "pi*2*HV_mean_radius",
"HV_y1": "620mm",
"HV_y2": "2140mm",
"HV_height": "HV_y2-HV_y1",
"HV_LV_gap_radius": "(LV_x2 + HV_x1)/2",
"HV_LV_gap_length": "pi*2*HV_LV_gap_radius",
}

specifications = {
 "Amp_turns": "135024A",
 "Frequency": "50Hz",
 "HV_turns": "980",
 "HV_current": "Amp_turns/HV_turns",
}

```

## Define variables from dictionaries

Define design variables from the created dictionaries.

```

m2d.variable_manager.set_variable(variable_name="Dimensions")

for k, v in dimensions.items():
 m2d[k] = v

m2d.variable_manager.set_variable(variable_name="Windings")

for k, v in specifications.items():
 m2d[k] = v

```

## Create design geometries

Create transformer core, HV and LV windings, and the region.

```

core_id = mod.create_rectangle(
 position=[0, 0, 0],
 dimension_list=["core_width", "core_height", 0],
 name="core",
 matname="steel_1008",
)

```

(continues on next page)

(continued from previous page)

```

core_hole_id = mod.create_rectangle(
 position=["core_opening_x1", "core_opening_y1", 0],
 dimension_list=["core_opening_width", "core_opening_height", 0],
 name="core_hole",
)

mod.subtract(blank_list=[core_id], tool_list=[core_hole_id], keep_originals=False)

lv_id = mod.create_rectangle(
 position["LV_x1", "LV_y1", 0],
 dimension_list=["LV_width", "LV_height", 0],
 name="LV",
 matname="copper",
)

hv_id = mod.create_rectangle(
 position["HV_x1", "HV_y1", 0],
 dimension_list=["HV_width", "HV_height", 0],
 name="HV",
 matname="copper",
)

Very small region is enough, because all the flux is concentrated in the core
region_id = mod.create_region(
 pad_percent=[20, 10, 0, 10]
)

```

## Assign boundary condition

Assign vector potential to zero on all region boundaries. This makes x=0 edge a symmetry boundary.

```

region_edges = region_id.edges

m2d.assign_vector_potential(
 input_edge=region_edges,
 bound_name="VectorPotential1"
)

```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258AAEEA350>
```

## Create initial mesh settings

Assign a relatively dense mesh to all objects to ensure that the energy is calculated accurately.

```

m2d.mesh.assign_length_mesh(
 names=["core", "Region", "LV", "HV"],
 maxlen=50,
 maxel=None,
 meshop_name="all_objects"
)

```

```
<pyaedt.modules.Mesh.MeshOperation object at 0x00000258AAEE8700>
```

## Define excitations

Assign the same current in amp-turns but in opposite directions to HV and LV windings.

```
m2d.assign_current(
 object_list=lv_id,
 amplitude="Amp_turns",
 name="LV"
)
m2d.assign_current(
 object_list=hv_id,
 amplitude="Amp_turns",
 name="HV",
 swap_direction=True
)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258AAEB1C0>
```

## Create and analyze the setup

Create and analyze the setup. Setu no. of minimum passes to 3 to ensure accuracy.

```
m2d.create_setup(
 setupname="Setup1",
 MinimumPasses=3
)
m2d.analyze_setup()
```

```
True
```

## Calculate transformer leakage inductance and reactance

Calculate transformer leakage inductance from the magnetic energy.

```
field_calculator = m2d.ofieldsreporter

field_calculator.EnterQty("Energy")
field_calculator.EnterSurf("HV")
field_calculator.CalcOp("Integrate")
field_calculator.EnterScalarFunc("HV_mean_turn_length")
field_calculator.CalcOp("*")

field_calculator.EnterQty("Energy")
field_calculator.EnterSurf("LV")
field_calculator.CalcOp("Integrate")
field_calculator.EnterScalarFunc("LV_mean_turn_length")
```

(continues on next page)

(continued from previous page)

```

field_calculator.CalcOp("*")
field_calculator.EnterQty("Energy")
field_calculator.EnterSurf("Region")
field_calculator.CalcOp("Integrate")
field_calculator.EnterScalarFunc("HV_LV_gap_length")
field_calculator.CalcOp("*")
field_calculator.CalcOp("+")
field_calculator.CalcOp("+")

field_calculator.EnterScalar(2)
field_calculator.CalcOp("*")
field_calculator.EnterScalarFunc("HV_current")
field_calculator.EnterScalarFunc("HV_current")
field_calculator.CalcOp("*")
field_calculator.CalcOp("/")
field_calculator.AddNamedExpression("Leakage_inductance", "Fields")

field_calculator.CopyNamedExprToStack("Leakage_inductance")
field_calculator.EnterScalar(2)
field_calculator.EnterScalar(3.14159265358979)
field_calculator.EnterScalarFunc("Frequency")
field_calculator.CalcOp("*")
field_calculator.CalcOp("*")
field_calculator.CalcOp("*")
field_calculator.AddNamedExpression("Leakage_reactance", "Fields")

m2d.post.create_report(
 expressions=["Leakage_inductance", "Leakage_reactance"],
 report_category="Fields",
 primary_sweep_variable="core_width",
 plot_type="Data Table",
 plotname="Transformer Leakage Inductance",
)

```

```
<pyaedt.modules.report_templates.Fields object at 0x00000258BBA75510>
```

### Print leakage inductance and reactance values in the Message Manager

Print leakage inductance and reactance values in the Message Manager

```

m2d.logger.clear_messages()
m2d.logger.info(
 "Leakage_inductance = {:.4f}H".format(m2d.post.get_scalar_field_value(quantity_name=
 "Leakage_inductance")))
)
m2d.logger.info(
 "Leakage_reactance = {:.2f}Ohm".format(m2d.post.get_scalar_field_value(quantity_
 name="Leakage_reactance")))
)

```

## Plot energy in the simulation domain

Most of the energy is confined in the air between the HV and LV windings.

```
object_faces = []
for name in mod.object_names:
 object_faces.extend(m2d.modeler.get_object_faces(name))

energy_field_overlay = m2d.post.create_fieldplot_surface(
 objlist=object_faces,
 quantityName="energy",
 plot_name="Energy",
)

m2d.save_project()
m2d.release_desktop()
temp_dir.cleanup()
```

**Total running time of the script:** (1 minutes 32.561 seconds)

## Maxwell 3D: asymmetric conductor analysis

This example uses PyAEDT to set up the TEAM 7 problem for an asymmetric conductor with a hole and solve it using the Maxwell 3D Eddy Current solver. <https://www.compumag.org/wp/wp-content/uploads/2018/06/problem7.pdf>

### Perform required imports

Perform required imports.

```
import numpy as np
import os
import tempfile

from pyaedt import Maxwell3d
from pyaedt.generic.general_methods import write_csv
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Create temporary directory

Create temporary directory.

```
temp_dir = tempfile.TemporaryDirectory(suffix=".ansys")
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Launch AEDT and Maxwell 3D

Launch AEDT and Maxwell 3D. The following code sets up the project and design names, the solver, and the version. It also creates an instance of the `Maxwell3d` class named `m3d`.

```
project_name = "COMPUMAG"
design_name = "TEAM 7 Asymmetric Conductor"
solver = "EddyCurrent"

m3d = Maxwell3d(
 projectname=project_name,
 designname=design_name,
 solution_type=solver,
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True
)
m3d.modeler.model_units = "mm"
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 1396 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Add Maxwell 3D setup

Add a Maxwell 3D setup with frequency points at DC, 50 Hz, and 200Hz. Otherwise, the default PyAEDT setup values are used. To approximate a DC field in the Eddy Current solver, use a low frequency value. Second-order shape functions improve the smoothness of the induced currents in the plate.

```
dc_freq = 0.1
stop_freq = 50

setup = m3d.create_setup(name="Setup1")
```

(continues on next page)

(continued from previous page)

```
setup.props["Frequency"] = "200Hz"
setup.props["HasSweepSetup"] = True
setup.add_eddy_current_sweep("LinearStep", dc_freq, stop_freq, stop_freq - dc_freq, ↵
 clear=True)
setup.props["UseHighOrderShapeFunc"] = True
setup.props["PercentError"] = 0.4
setup.update()
```

True

## Define coil dimensions

Define coil dimensions as shown on the TEAM7 drawing of the coil.

```
coil_external = 150 + 25 + 25
coil_internal = 150
coil_r1 = 25
coil_r2 = 50
coil_thk = coil_r2 - coil_r1
coil_height = 100
coil_centre = [294 - 25 - 150 / 2, 25 + 150 / 2, 19 + 30 + 100 / 2]

Use expressions to construct the three dimensions needed to describe the midpoints of
the coil.

dim1 = coil_internal / 2 + (coil_external - coil_internal) / 4
dim2 = coil_internal / 2 - coil_r1
dim3 = dim2 + np.sqrt(((coil_r1 + (coil_r2 - coil_r1) / 2) ** 2) / 2)

Use coordinates to draw a polyline along which to sweep the coil cross sections.
P1 = [dim1, -dim2, 0]
P2 = [dim1, dim2, 0]
P3 = [dim3, dim3, 0]
P4 = [dim2, dim1, 0]
```

## Create coordinate system for positioning coil

Create a coordinate system for positioning the coil.

```
m3d.modeler.create_coordinate_system(origin=coil_centre, mode="view", view="XY", name=
 "Coil_CS")
```

```
<pyaedt.modeler.cad.Modeler.CoordinateSystem object at 0x00000258A9B1EE60>
```

## Create polyline

Create a polyline. One quarter of the coil is modeled by sweeping a 2D sheet along a polyline.

```
test = m3d.modeler.create_polyline(points=[P1, P2, P3, P4], segment_type=["Line", "Arc"],
 name="Coil")
test.set_crosssection_properties(type="Rectangle", width=coil_thk, height=coil_height)

<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258A9B1FC10>
```

## Duplicate and unite polyline to create full coil

Duplicate and unit the polyline to create a full coil.

```
m3d.modeler.duplicate_around_axis(
 "Coil", axis="Global", angle=90, clones=4, create_new_objects=True, is_3d_comp=False)
)
m3d.modeler.unite("Coil", "Coil_1", "Coil_2")
m3d.modeler.unite("Coil", "Coil_3")
m3d.modeler.fit_all()
```

## Assign material and if solution is allowed inside coil

Assign the material Cooper from the Maxwell internal library to the coil and allow a solution inside the coil.

```
m3d.assign_material("Coil", "Copper")
m3d.solve_inside("Coil")
```

```
True
```

## Create terminal

Create a terminal for the coil from a cross-section that is split and one half deleted.

```
m3d.modeler.section("Coil", "YZ")
m3d.modeler.separate_bodies("Coil_Section1")
m3d.modeler.delete("Coil_Section1_Separate1")

Add variable for coil excitation
~~~~~
Add a design variable for coil excitation. The NB units here are AmpereTurns.

Coil_Excitation = 2742
m3d["Coil_Excitation"] = str(Coil_Excitation) + "A"
m3d.assign_current(assignment="Coil_Section1", amplitude="Coil_Excitation", solid=False)
m3d.modeler.set_working_coordinate_system("Global")
```

```
True
```

## Add a material

Add a material named team3\_aluminium.

```
mat = m3d.materials.add_material("team7_aluminium")
mat.conductivity = 3.526e7
```

## Model aluminium plate with a hole

Model the aluminium plate with a hole by subtracting two rectangular cuboids.

```
plate = m3d.modeler.create_box(origin=[0, 0, 0], sizes=[294, 294, 19], name="Plate",
 material="team7_aluminium")
m3d.modeler.fit_all()
m3d.modeler.create_box(origin=[18, 18, 0], sizes=[108, 108, 19], name="Hole")
m3d.modeler.subtract(blank_list="Plate", tool_list=["Hole"], keep_originals=False)
```

True

## Draw a background region

Draw a background region that uses the default properties for an air region.

```
m3d.modeler.create_air_region(x_pos=100, y_pos=100, z_pos=100, x_neg=100, y_neg=100, z_
 _neg=100)
```

<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BBA93550>

## Adjust eddy effects for plate and coil

Adjust the eddy effects for the plate and coil by turning off displacement currents for all parts. The setting for eddy effect is ignored for the stranded conductor type used in the coil.

```
m3d.eddy_effects_on(assignment="Plate")
m3d.eddy_effects_on(assignment=["Coil", "Region", "Line_A1_B1mesh", "Line_A2_B2mesh"],
 enable_eddy_effects=False,
 enable_displacement_current=False)
```

True

## Create expression for Z component of B in Gauss

Create an expression for the Z component of B in Gauss using the fields calculator.

```
Fields = m3d.ofieldsreporter
Fields.CalcStack("clear")
Fields.EnterQty("B")
Fields.CalcOp("ScalarZ")
Fields.EnterScalarFunc("Phase")
Fields.CalcOp("AtPhase")
Fields.EnterScalar(10000)
Fields.CalcOp("*")
Fields.CalcOp("Smooth")
Fields.AddNamedExpression("Bz", "Fields")
```

## Draw two lines along which to plot Bz

Draw two lines along which to plot Bz. The following code also adds a small cylinder to refine the mesh locally around each line.

```
lines = ["Line_A1_B1", "Line_A2_B2"]
mesh_diameter = "2mm"

line_points_1 = [[["0mm", "72mm", "34mm"], ["288mm", "72mm", "34mm"]]]
polyline = m3d.modeler.create_polyline(points=line_points_1, name=lines[0])
l1_mesh = m3d.modeler.create_polyline(points=line_points_1, name=lines[0] + "mesh")
l1_mesh.set_crosssection_properties(type="Circle", width=mesh_diameter)

line_points_2 = [[["0mm", "144mm", "34mm"], ["288mm", "144mm", "34mm"]]]
polyline2 = m3d.modeler.create_polyline(points=line_points_2, name=lines[1])
l2_mesh = m3d.modeler.create_polyline(points=line_points_2, name=lines[1] + "mesh")
l2_mesh.set_crosssection_properties(type="Circle", width=mesh_diameter)
```

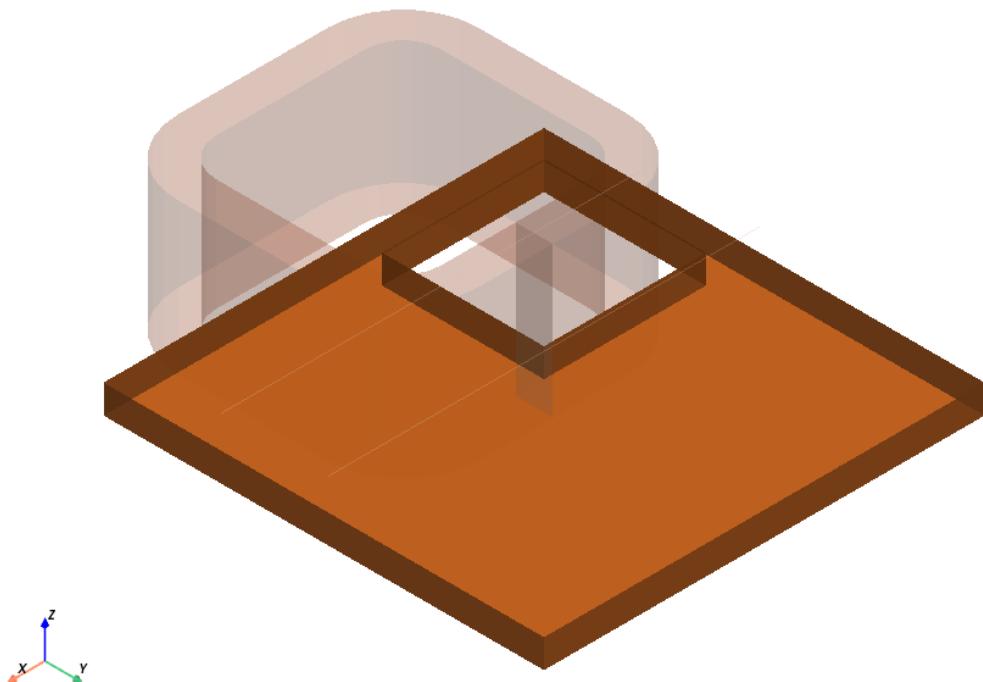
```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BE27EA10>
```

## Plot model

Plot the model.

```
m3d.plot(show=False, export_path=os.path.join(temp_dir.name, "model.jpg"), plot_air_
 ~objects=False)
```

■	Coil
■	Plate
■	Coil_Section1
■	Line_A1_B1
■	Line_A2_B2



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BE27F9A0>
```

Published measurement results are included with this script via the list below. Test results are used to create text files for import into a rectangular plot and to overlay simulation results.

```
dataset = [
 "Bz A1_B1 000 0",
 "Bz A1_B1 050 0",
 "Bz A1_B1 050 90",
 "Bz A1_B1 200 0",
 "Bz A1_B1 200 90",
 "Bz A2_B2 050 0",
 "Bz A2_B2 050 90",
 "Bz A2_B2 200 0",
 "Bz A2_B2 200 90",
]
```

(continues on next page)

(continued from previous page)

```
header = ["Distance [mm]", "Bz [Tesla]"]

line_length = [0, 18, 36, 54, 72, 90, 108, 126, 144, 162, 180, 198, 216, 234, 252, 270, 288]
data = [
 [
 -6.667,
 -7.764,
 -8.707,
 -8.812,
 -5.870,
 8.713,
 50.40,
 88.47,
 100.9,
 104.0,
 104.8,
 104.9,
 104.6,
 103.1,
 97.32,
 75.19,
 29.04,
],
 [
 4.90,
 -17.88,
 -22.13,
 -20.19,
 -15.67,
 0.36,
 43.64,
 78.11,
 71.55,
 60.44,
 53.91,
 52.62,
 53.81,
 56.91,
 59.24,
 52.78,
 27.61,
],
 [-1.16, 2.84, 4.15, 4.00, 3.07, 2.31, 1.89, 4.97, 12.61, 14.15, 13.04, 12.40, 12.05, 12.27, 12.66, 9.96, 2.36],
 [
 -3.63,
 -18.46,
 -23.62,
 -21.59,
 -16.09,
 0.23,
]
]
```

(continues on next page)

(continued from previous page)

```

44.35,
75.53,
63.42,
53.20,
48.66,
47.31,
48.31,
51.26,
53.61,
46.11,
24.96,
],
[-1.38, 1.20, 2.15, 1.63, 1.10, 0.27, -2.28, -1.40, 4.17, 3.94, 4.86, 4.09, 3.69, 4.
-60, 3.48, 4.10, 0.98],
[
-1.83,
-8.50,
-13.60,
-15.21,
-14.48,
-5.62,
28.77,
60.34,
61.84,
56.64,
53.40,
52.36,
53.93,
56.82,
59.48,
52.08,
26.56,
],
[-1.63, -0.60, -0.43, 0.11, 1.26, 3.40, 6.53, 10.25, 11.83, 11.83, 11.01, 10.58, 10.
-80, 10.54, 10.62, 9.03, 1.79],
[
-0.86,
-7.00,
-11.58,
-13.36,
-13.77,
-6.74,
24.63,
53.19,
54.89,
50.72,
48.03,
47.13,
48.25,
51.35,
53.35,
45.37,
]

```

(continues on next page)

(continued from previous page)

```

 24.01,
],
[-1.35, -0.71, -0.81, -0.67, 0.15, 1.39, 2.67, 3.00, 4.01, 3.80, 4.00, 3.02, 2.20, 2.
-78, 1.58, 1.37, 0.93],
]

```

## Write dataset values in a CSV file

Dataset details are used to encode test parameters in the text files. For example, Bz A1\_B1 050 0 is the Z component of flux density B. along line A1\_B1 at 50 Hz and 0 deg.

```

line_length.insert(0, header[0])
for i in range(len(dataset)):
 data[i].insert(0, header[1])
 ziplist = zip(line_length, data[i])
 file_path = os.path.join(temp_dir.name, str(dataset[i]) + ".csv")
 write_csv(output=file_path, list_data=ziplist)

```

## Create rectangular plots and import test data into report

Create rectangular plots, using text file encoding to control their formatting. Import test data into correct plot and overlay with simulation results. Variations for a DC plot must have different frequency and phase than the other plots.

```

for item in range(len(dataset)):
 if item % 2 == 0:
 t = dataset[item]
 plot_name = t[0:3] + "Along the Line" + t[2:9] + ", " + t[9:12] + "Hz"
 if t[9:12] == "000":
 variations = {
 "Distance": ["All"],
 "Freq": [str(dc_freq) + "Hz"],
 "Phase": ["0deg"],
 "Coil_Excitation": ["All"],
 }
 else:
 variations = {
 "Distance": ["All"],
 "Freq": [t[9:12] + "Hz"],
 "Phase": ["0deg", "90deg"],
 "Coil_Excitation": ["All"],
 }
 report = m3d.post.create_report(expressions=t[0:2], variations=variations,
- primary_sweep_variable="Distance",
 report_category="Fields", context="Line_" +
- t[3:8], plot_name=plot_name)
 file_path = os.path.join(temp_dir.name, str(dataset[i]) + ".csv")
 report.import_traces(file_path, plot_name)

```

## Analyze project

Analyze the project.

```
m3d.analyze()
```

```
True
```

## Create plots of induced current and flux density on surface of plate

Create two plots of the induced current (Mag\_J) and the flux density (Mag\_B) on the surface of the plate.

```
surf_list = m3d.modeler.get_object_faces("Plate")
intrinsic_dict = {"Freq": "200Hz", "Phase": "0deg"}
m3d.post.create_fieldplot_surface(surf_list, "Mag_J", intrinsics=intrinsic_dict, plot_
 ↴name="Mag_J")
m3d.post.create_fieldplot_surface(surf_list, "Mag_B", intrinsics=intrinsic_dict, plot_
 ↴name="Mag_B")
m3d.post.create_fieldplot_surface(surf_list, "Mesh", intrinsics=intrinsic_dict, plot_
 ↴name="Mesh")
```

```
<pyaedt.modules.solutions.FieldPlot object at 0x00000258BE29BAF0>
```

## Release AEDT and clean up temporary directory

Release AEDT and remove both the project and temporary directories.

```
m3d.release_desktop(True, True)
temp_dir.cleanup()
```

**Total running time of the script:** (6 minutes 31.259 seconds)

## Maxwell 3D: choke setup

This example shows how you can use PyAEDT to create a choke setup in Maxwell 3D.

### Perform required imports

Perform required imports.

```
import json
import os
import pyaedt
import tempfile
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Create temporary directory

Create temporary directory.

```
temp_dir = tempfile.TemporaryDirectory(suffix=".ansys")
```

## Set non-graphical mode

Set non-graphical mode. You can define `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Launch Maxwell3D

Launch Maxwell 3D 2023 R2 in graphical mode.

```
m3d = pyaedt.Maxwell3d(projectname=pyaedt.generate_unique_project_name(),
 solution_type="EddyCurrent",
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 5988 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 _py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 _log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Rules and information of use

The dictionary values containing the different parameters of the core and the windings that compose the choke. You must not change the main structure of the dictionary. The dictionary has many primary keys, including "Number of Windings", "Layer", and "Layer Type", that have dictionaries as values. The keys of these dictionaries are secondary keys of the dictionary values, such as "1", "2", "3", "4", and "Simple".

You must not modify the primary or secondary keys. You can modify only their values. You must not change the data types for these keys. For the dictionaries from "Number of Windings" through "Wire Section", values must be Boolean. Only one value per dictionary can be "True". If all values are True, only the first one remains set to True. If all values are False, the first value is chosen as the correct one by default. For the dictionaries from "Core" through "Inner Winding", values must be strings, floats, or integers.

Descriptions follow for primary keys:

- "Number of Windings": Number of windings around the core
- "Layer": Number of layers of all windings
- "Layer Type": Whether layers of a winding are linked to each other
- "Similar Layer": Whether layers of a winding have the same number of turns and same spacing between turns
- "Mode": When there are only two windows, whether they are in common or differential mode
- "Wire Section": Type of wire section and number of segments
- "Core": Design of the core
- "Outer Winding": Design of the first layer or outer layer of a winding and the common parameters for all layers
- "Mid Winding": Turns and turns spacing (Coil Pit) for the second or mid layer if it is necessary
- "Inner Winding": Turns and turns spacing (Coil Pit) for the third or inner layer if it is necessary
- "Occupation(%)": An informative parameter that is useless to modify

The following parameter values work. You can modify them if you want.

```
values = {
 "Number of Windings": {"1": False, "2": False, "3": True, "4": False},
 "Layer": {"Simple": False, "Double": False, "Triple": True},
 "Layer Type": {"Separate": False, "Linked": True},
 "Similar Layer": {"Similar": False, "Different": True},
 "Mode": {"Differential": True, "Common": False},
 "Wire Section": {"None": False, "Hexagon": False, "Octagon": True, "Circle": False},
 "Core": {
 "Name": "Core",
 "Material": "ferrite",
 "Inner Radius": 100,
 "Outer Radius": 143,
 "Height": 25,
 "Chamfer": 0.8,
 },
 "Outer Winding": {
 "Name": "Winding",
 "Material": "copper",
 "Inner Radius": 100,
 "Outer Radius": 143,
 "Height": 25,
 "Wire Diameter": 5,
 "Turns": 2,
 "Coil Pit(deg)": 4,
 "Occupation)": 0,
 },
 "Mid Winding": {"Turns": 7, "Coil Pit(deg)": 4, "Occupation(%)"": 0},
 "Inner Winding": {"Turns": 10, "Coil Pit(deg)": 4, "Occupation(%)"": 0},
}
```

## Convert dictionary to JSON file

Covert a dictionary to a JSON file. PyAEDT methods ask for the path of the JSON file as an argument. You can convert a dictionary to a JSON file.

```
json_path = os.path.join(temp_dir.name, "choke_example.json")

with open(json_path, "w") as outfile:
 json.dump(values, outfile)
```

## Verify parameters of JSON file

Verify parameters of the JSON file. The `check_choke_values` method takes the JSON file path as an argument and does the following:

- Checks if the JSON file is correctly written (as explained in the rules)
- Checks inequations on windings parameters to avoid having unintended intersections

```
dictionary_values = m3d.modeler.check_choke_values(json_path, create_another_file=False)
print(dictionary_values)
```

```
[True, {'Number of Windings': {'1': False, '2': False, '3': True, '4': False}, 'Layer': {
 'Simple': False, 'Double': False, 'Triple': True}, 'Layer Type': {'Separate': False,
 'Linked': True}, 'Similar Layer': {'Similar': False, 'Different': True}, 'Mode': {
 'Differential': True, 'Common': False}, 'Wire Section': {'None': False, 'Hexagon': False,
 'Octagon': True, 'Circle': False}, 'Core': {'Name': 'Core', 'Material': 'ferrite',
 'Inner Radius': 100, 'Outer Radius': 143, 'Height': 25, 'Chamfer': 0.8}, 'Outer Winding': {
 'Name': 'Winding', 'Material': 'copper', 'Inner Radius': 86.25, 'Outer Radius': 156.75,
 'Height': 52.5, 'Wire Diameter': 5, 'Turns': 2, 'Coil Pit(deg)': 4,
 'Occupation(%)': 13.33333333333334}, 'Mid Winding': {'Turns': 7, 'Coil Pit(deg)': 4.0,
 'Occupation(%)': 46.66666666666664}, 'Inner Winding': {'Turns': 10, 'Coil Pit(deg)': 4.0,
 'Occupation(%)': 66.6666666666667}}]
```

## Create choke

Create the choke. The `create_choke` method takes the JSON file path as an argument.

```
list_object = m3d.modeler.create_choke(json_path)
print(list_object)
core = list_object[1]
first_winding_list = list_object[2]
second_winding_list = list_object[3]
third_winding_list = list_object[4]
```

```
[<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BBA92560>, <pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BBA933A0>, [<pyaedt.modeler.cad.polylines.Polyline object at 0x00000258BBA93880>, [[-74.497822093321, 62.511095042016, -52.5], [-74.497822093321, 62.511095042016, 12.839087296526], [-76.344688372662, 64.060799855845, 15.25], [-115.963757581084, 84.252601589961, 15.25], [-117.914226930149, 85.669700521628, -12.839087296526], [-117.914226930149, 85.669700521628, -12.839087296526]], (continues on next page)]
```

(continued from previous page)

↵ [-115.963757581084, 84.252601589961, -15.25], [-84.517247279393, 52.812237532635, -15.  
 ↵ 25], [-82.472677351212, 51.534648446679, -12.839087296526], [-82.472677351212, 51.  
 ↵ 534648446679, 12.839087296526], [-84.517247279393, 52.812237532635, 15.25], [-126.  
 ↵ 560902091063, 67.293625321403, 15.25], [-128.689611659189, 68.425480276044, 12.  
 ↵ 839087296526], [-128.689611659189, 68.425480276044, -12.839087296526], [-126.  
 ↵ 560902091063, 67.293625321403, -15.25], [-91.044774104774, 40.535745078881, -15.25], [-  
 ↵ 88.842295755743, 39.555138539122, -12.839087296526], [-88.842295755743, 39.  
 ↵ 555138539122, 12.839087296526], [-91.044774104774, 40.535745078881, 15.25], [-134.  
 ↵ 694682602733, 49.024855181328, 15.25], [-136.960199479546, 49.849435889716, 12.  
 ↵ 839087296526], [-136.960199479546, 49.849435889716, -12.839087296526], [-134.  
 ↵ 694682602733, 49.024855181328, -15.25], [-95.800217964102, 27.470270455894, -15.25], [-  
 ↵ 93.482699930001, 26.805732853203, -12.839087296526], [-93.482699930001, 26.  
 ↵ 805732853203, 12.839087296526], [-95.800217964102, 27.470270455894, 15.25], [-140.  
 ↵ 20678433047, 29.801872000095, 15.25], [-142.565012806952, 30.303128936688, 12.  
 ↵ 839087296526], [-142.565012806952, 30.303128936688, -12.839087296526], [-140.  
 ↵ 20678433047, 29.801872000095, -15.25], [-98.691019551891, 13.870118265453, -15.25], [-  
 ↵ 96.303569685118, 13.534584068366, -12.839087296526], [-96.303569685118, 13.  
 ↵ 534584068366, 12.839087296526], [-98.691019551891, 13.870118265453, 15.25], [-142.  
 ↵ 989920484069, 9.998829279507, 15.25], [-145.394960325369, 10.167006048206, 12.  
 ↵ 839087296526], [-145.394960325369, 10.167006048206, -12.839087296526], [-142.  
 ↵ 989920484069, 9.998829279507, -15.25], [-99.660912703474, 0.0, -15.25], [-97.25, 0.0, -  
 ↵ 12.839087296526], [-97.25, 0.0, 12.839087296526], [-99.660912703474, 0.0, 15.25], [-  
 ↵ 142.989920484069, -9.998829279507, 15.25], [-145.394960325369, -10.167006048206, 12.  
 ↵ 839087296526], [-145.394960325369, -10.167006048206, -12.839087296526], [-142.  
 ↵ 989920484069, -9.998829279507, -15.25], [-98.691019551891, -13.870118265453, -15.25], [-  
 ↵ 96.303569685118, -13.534584068366, -12.839087296526], [-96.303569685118, -13.  
 ↵ 534584068366, 12.839087296526], [-98.691019551891, -13.870118265453, 15.25], [-140.  
 ↵ 20678433047, -29.801872000095, 15.25], [-142.565012806952, -30.303128936688, 12.  
 ↵ 839087296526], [-142.565012806952, -30.303128936688, -12.839087296526], [-140.  
 ↵ 20678433047, -29.801872000095, -15.25], [-95.800217964102, -27.470270455894, -15.25], [-  
 ↵ 93.482699930001, -26.805732853203, -12.839087296526], [-93.482699930001, -26.  
 ↵ 805732853203, 12.839087296526], [-95.800217964102, -27.470270455894, 15.25], [-134.  
 ↵ 694682602733, -49.024855181328, 15.25], [-136.960199479546, -49.849435889716, 12.  
 ↵ 839087296526], [-136.960199479546, -49.849435889716, -12.839087296526], [-134.  
 ↵ 694682602733, -49.024855181328, -15.25], [-91.044774104774, -40.535745078881, -15.25], [-  
 ↵ 88.842295755743, -39.555138539122, -12.839087296526], [-88.842295755743, -39.  
 ↵ 555138539122, 12.839087296526], [-91.044774104774, -40.535745078881, 15.25], [-126.  
 ↵ 560902091063, -67.293625321403, 15.25], [-128.689611659189, -68.425480276044, 12.  
 ↵ 839087296526], [-128.689611659189, -68.425480276044, -12.839087296526], [-126.  
 ↵ 560902091063, -67.293625321403, -15.25], [-84.517247279393, -52.812237532635, -15.25], [-  
 ↵ 82.472677351212, -51.534648446679, -12.839087296526], [-82.472677351212, -51.  
 ↵ 534648446679, 12.839087296526], [-84.517247279393, -52.812237532635, 15.25], [-115.  
 ↵ 963757581084, -84.252601589961, 15.25], [-117.914226930149, -85.669700521628, 12.  
 ↵ 839087296526], [-117.914226930149, -85.669700521628, -12.839087296526], [-115.  
 ↵ 963757581084, -84.252601589961, -15.25], [-76.344688372662, -64.060799855845, -15.25], [-  
 ↵ 74.497822093321, -62.511095042016, -12.839087296526], [-74.497822093321, -62.  
 ↵ 511095042016, 15.117261889578], [-76.344688372662, -64.060799855845, 20.75], [-117.  
 ↵ 806839543016, -85.591679017905, 20.75], [-122.363820399211, -88.902519409237, 15.  
 ↵ 117261889578], [-122.363820399211, -88.902519409237, -15.117261889578], [-117.  
 ↵ 806839543016, -85.591679017905, -20.75], [-82.585245653043, -51.60498892849, -20.75], [-  
 ↵ 77.808412822352, -48.620092493397, -15.117261889578], [-77.808412822352, -48.  
 ↵ 620092493397, 15.117261889578], [-82.585245653043, -51.60498892849, 20.75], [-128.

(continues on next page)

(continued from previous page)

```
→ 572410864111, -68.363163507903, 20.75], [-133.545823419913, -71.007573871366, 15.
→ 117261889578], [-133.545823419913, -71.007573871366, -15.117261889578], [-128.
→ 572410864111, -68.363163507903, -20.75], [-88.963558053575, -39.609127992563, -20.75],
→ [-83.817795738709, -37.318087002205, -15.117261889578], [-83.817795738709, -37.
→ 318087002205, 15.117261889578], [-88.963558053575, -39.609127992563, 20.75], [-136.
→ 835466456686, -49.804036782165, 20.75], [-142.128508893869, -51.730546678007, 15.
→ 117261889578], [-142.128508893869, -51.730546678007, -15.117261889578], [-136.
→ 835466456686, -49.804036782165, -20.75], [-93.610295991141, -26.842320434976, -20.75],
→ [-88.195760602341, -25.28972739621, -15.117261889578], [-88.195760602341, -25.
→ 28972739621, 15.117261889578], [-93.610295991141, -26.842320434976, 20.75], [-142.
→ 435175342717, -30.275531131715, 20.75], [-147.944824610988, -31.446643236186, 15.
→ 117261889578], [-147.944824610988, -31.446643236186, -15.117261889578], [-142.
→ 435175342717, -30.275531131715, -20.75], [-96.435015997374, -13.553057642809, -20.75],
→ [-90.857095307039, -12.769132013086, -15.117261889578], [-90.857095307039, -12.
→ 769132013086, 15.117261889578], [-96.435015997374, -13.553057642809, 20.75], [-145.
→ 262545558313, -10.157746705692, 20.75], [-150.881562601798, -10.550666653799, 15.
→ 117261889578], [-150.881562601798, -10.550666653799, -15.117261889578], [-145.
→ 262545558313, -10.157746705692, -20.75], [-97.382738110422, 0.0, -20.75], [-91.75, 0.0,
→ -15.117261889578], [-91.75, 0.0, 15.117261889578], [-97.382738110422, 0.0, 20.75], [-
→ 145.262545558313, 10.157746705692, 20.75], [-150.881562601798, 10.550666653799, 15.
→ 117261889578], [-150.881562601798, 10.550666653799, -15.117261889578], [-145.
→ 262545558313, 10.157746705692, -20.75], [-96.435015997374, 13.553057642809, -20.75], [-
→ 90.857095307039, 12.769132013086, -15.117261889578], [-90.857095307039, 12.
→ 769132013086, 15.117261889578], [-96.435015997374, 13.553057642809, 20.75], [-142.
→ 435175342717, 30.275531131715, 20.75], [-147.944824610988, 31.446643236186, 15.
→ 117261889578], [-147.944824610988, 31.446643236186, -15.117261889578], [-147.
→ 944824610988, 31.446643236186, -17.39543648263], [-142.435175342717, 30.275531131715, -
→ 26.25], [-94.179012442856, 13.235997020166, -26.25], [-85.41062092896, 12.003679957806,
→ -17.39543648263], [-85.41062092896, 12.003679957806, 17.39543648263], [-94.
→ 179012442856, 13.235997020166, 26.25], [-147.535170632557, 10.316664131877, 26.25], [-
→ 156.368164878227, 10.934327259392, 17.39543648263], [-156.368164878227, 10.
→ 934327259392, -17.39543648263], [-147.535170632557, 10.316664131877, -26.25], [-95.
→ 10456351737, 0.0, -26.25], [-86.25, 0.0, -17.39543648263], [-86.25, 0.0, 17.
→ 39543648263], [-95.10456351737, 0.0, 26.25], [-147.535170632557, -10.316664131877, 26.
→ 25], [-156.368164878227, -10.934327259392, 17.39543648263], [-156.368164878227, -10.
→ 934327259392, -17.39543648263], [-156.368164878227, -10.934327259392, -52.5]], [
→ <pyaedt.modeler.cad.polylines.Polyline object at 0x00000258BE29AC50>, [-16.
→ 887285278109, -95.772553980438, -52.5], [-16.887285278109, -95.772553980438, 12.
→ 839087296526], [-17.305935875581, -98.146839502654, 15.25], [-14.983014521293, -142.
→ 553860778499, 15.25], [-15.23502352126, -144.951566249926, 12.839087296526], [-15.
→ 23502352126, -144.951566249926, -12.839087296526], [-14.983014521293, -142.
→ 553860778499, -15.25], [-3.478115694263, -99.600201968203, -15.25], [-3.393976054318, -
→ 97.190757927607, -12.839087296526], [-3.393976054318, -97.190757927607, 12.
→ 839087296526], [-3.478115694263, -99.600201968203, 15.25], [5.002462004445, -143.
→ 251768997437, 15.25], [5.086601644389, -145.661213038034, 12.839087296526], [5.
→ 086601644389, -145.661213038034, -12.839087296526], [5.002462004445, -143.251768997437,
→ -15.25], [10.417402052746, -99.11495979599, -15.25], [10.165393052779, -96.
→ 717254324565, -12.839087296526], [10.165393052779, -96.717254324565, 12.839087296526],
→ [10.417402052746, -99.11495979599, 15.25], [24.890571297483, -141.161444479313, 15.25],
→ [25.309221894955, -143.535730001529, 12.839087296526], [25.309221894955, -143.
→ 535730001529, -12.839087296526], [24.890571297483, -141.161444479313, -15.25], [24.
→ 110156918418, -96.700557672946, -15.25], [23.526904347068, -94.36125938034, -12.
```

(continues on next page)

(continued from previous page)

↵ 839087296526], [23.526904347068, -94.36125938034, 12.839087296526], [24.110156918418, -  
 ↵ 96.700557672946, 15.25], [44.294213932821, -136.32357301316, 15.25], [45.039226930149, -  
 ↵ -138.616487250018, 12.839087296526], [45.039226930149, -138.616487250018, -12.  
 ↵ 839087296526], [44.294213932821, -136.32357301316, -15.25], [37.333635004569, -92.  
 ↵ 403989190051, -15.25], [36.430491209698, -90.16862985662, -12.839087296526], [36.  
 ↵ 430491209698, -90.16862985662, 12.839087296526], [37.333635004569, -92.403989190051, -  
 ↵ 15.25], [62.835720077878, -128.832318264074, 15.25], [63.892594644508, -130.  
 ↵ 999232248103, 12.839087296526], [63.892594644508, -130.999232248103, -12.839087296526],  
 ↵ [62.835720077878, -128.832318264074, -15.25], [49.830456351737, -86.308882165552, -15.  
 ↵ 25], [48.625, -84.220970518037, -12.839087296526], [48.625, -84.220970518037, 12.  
 ↵ 839087296526], [49.830456351737, -86.308882165552, 15.25], [80.154200406191, -118.  
 ↵ 833488984567, 15.25], [81.502365680861, -120.832226199897, 12.839087296526], [81.  
 ↵ 502365680861, -120.832226199897, -12.839087296526], [80.154200406191, -118.  
 ↵ 833488984567, -15.25], [61.357384547322, -78.533870924598, -15.25], [59.87307847542, -  
 ↵ 76.634045788254, -12.839087296526], [59.87307847542, -76.634045788254, 12.  
 ↵ 839087296526], [61.357384547322, -78.533870924598, 15.25], [95.912570397649, -106.  
 ↵ 521701013065, 15.25], [97.525785876803, -108.31335831333, 12.839087296526], [97.  
 ↵ 525785876803, -108.31335831333, -12.839087296526], [95.912570397649, -106.521701013065,  
 ↵ -15.25], [71.690061045684, -69.230287217052, -15.25], [69.955795582933, -67.555526527137, 12.839087296526],  
 ↵ [71.690061045684, -69.230287217052, 15.25], [109.80411130525, -92.136589297985, 15.25],  
 ↵ [111.650977584591, -93.686294111813, 12.839087296526], [111.650977584591, -93.  
 ↵ 686294111813, -12.839087296526], [109.80411130525, -92.136589297985, -15.25], [80.  
 ↵ 627372052028, -58.579214717109, -15.25], [78.676902702964, -57.162115785443, -12.  
 ↵ 839087296526], [78.676902702964, -57.162115785443, 12.839087296526], [80.627372052028,  
 ↵ -58.579214717109, 15.25], [121.558440086618, -75.958143676034, 15.25], [123.6030100148,  
 ↵ -77.23573276199, 12.839087296526], [123.6030100148, -77.23573276199, -12.  
 ↵ 839087296526], [121.558440086618, -75.958143676034, -15.25], [87.995362973656, -46.  
 ↵ 787964435568, -15.25], [85.86665340553, -45.656109480928, -12.839087296526], [85.  
 ↵ 86665340553, -45.656109480928, 12.839087296526], [87.995362973656, -46.787964435568,  
 ↵ 15.25], [130.946772102377, -58.301259188539, 15.25], [133.149250451409, -59.  
 ↵ 281865728298, 12.839087296526], [133.149250451409, -59.281865728298, -12.839087296526],  
 ↵ [130.946772102377, -58.301259188539, -15.25], [93.650624248243, -34.086039646809, -15.  
 ↵ 25], [91.38510737143, -33.261458938422, -12.839087296526], [91.38510737143, -33.  
 ↵ 261458938422, 15.117261889578], [93.650624248243, -34.086039646809, 20.75], [133.  
 ↵ 027988153577, -59.227876274857, 20.75], [138.173750468444, -61.518917265215, 15.  
 ↵ 117261889578], [138.173750468444, -61.518917265215, -15.117261889578], [133.  
 ↵ 027988153577, -59.227876274857, -20.75], [85.983854200609, -45.718426249069, -20.75],  
 ↵ [81.010441644807, -43.074015885605, -15.117261889578], [81.010441644807, -43.  
 ↵ 074015885605, 15.117261889578], [85.983854200609, -45.718426249069, 20.75], [123.  
 ↵ 490441712969, -77.165392280179, 20.75], [128.26727454366, -80.150288715273, 15.  
 ↵ 117261889578], [128.26727454366, -80.150288715273, -15.117261889578], [123.  
 ↵ 490441712969, -77.165392280179, -20.75], [78.784290090096, -57.240137289166, -20.75],  
 ↵ [74.227309233902, -53.929296897835, -15.117261889578], [74.227309233902, -53.  
 ↵ 929296897835, 15.117261889578], [78.784290090096, -57.240137289166, 20.75], [111.  
 ↵ 549294292712, -93.600971699101, 20.75], [115.864222021745, -97.22162596509, 15.  
 ↵ 117261889578], [115.864222021745, -97.22162596509, -15.117261889578], [111.  
 ↵ 549294292712, -93.600971699101, -20.75], [70.051279388782, -67.647734166621, -20.75],  
 ↵ [65.999426681072, -63.734905489613, -15.117261889578], [65.999426681072, -63.  
 ↵ 734905489613, 15.117261889578], [70.051279388782, -67.647734166621, 20.75], [97.  
 ↵ 43696674449, -108.214714673426, 20.75], [101.206004211777, -112.400654853456, 15.  
 ↵ 117261889578], [101.206004211777, -112.400654853456, -15.117261889578], [97.

(continues on next page)

(continued from previous page)

```
↳ 43696674449, -108.214714673426, -20.75], [59.954800216314, -76.73864484668, -20.75], ↳
↳ [56.486940361129, -72.299986643417, -15.117261889578], [56.486940361129, -72.
↳ 299986643417, 15.117261889578], [59.954800216314, -76.73864484668, 20.75], [81.
↳ 428139471493, -120.722181319047, 20.75], [84.57792664995, -125.39193284895, 15.
↳ 117261889578], [84.57792664995, -125.39193284895, -15.117261889578], [81.428139471493, ↳
↳ -120.722181319047, -20.75], [48.691369055211, -84.335925093712, -20.75], [45.875, -79.
↳ 457830797222, -15.117261889578], [45.875, -79.457830797222, 15.117261889578], [48.
↳ 691369055211, -84.335925093712, 20.75], [63.83440608682, -130.879928024739, 20.75], ↳
↳ [66.303635951848, -135.942599502749, 15.117261889578], [66.303635951848, -135.
↳ 942599502749, -15.117261889578], [63.83440608682, -130.879928024739, -20.75], [36.
↳ 48021578106, -90.291702489489, -20.75], [34.37015494591, -85.069118656503, -15.
↳ 117261889578], [34.37015494591, -85.069118656503, 15.117261889578], [36.48021578106, -
↳ 90.291702489489, 20.75], [44.998208598227, -138.490245805141, 20.75], [46.738820399211,
↳ -143.847298089642, 15.117261889578], [46.738820399211, -143.847298089642, -15.
↳ 117261889578], [46.738820399211, -143.847298089642, -17.39543648263], [44.998208598227,
↳ -138.490245805141, -26.25], [35.626796557549, -88.179415788927, -26.25], [32.
↳ 309818682122, -79.969607456385, -17.39543648263], [32.309818682122, -79.969607456385, ↳
↳ 17.39543648263], [35.626796557549, -88.179415788927, 26.25], [64.833092095761, -132.
↳ 927537785405, 26.25], [68.714677259187, -140.885966757394, 17.39543648263], [68.
↳ 714677259187, -140.885966757394, -17.39543648263], [64.833092095761, -132.927537785405,
↳ -26.25], [47.552281758685, -82.362968021873, -26.25], [43.125, -74.694691076408, -17.
↳ 39543648263], [43.125, -74.694691076408, 17.39543648263], [47.552281758685, -82.
↳ 362968021873, 26.25], [82.702078536796, -122.610873653528, 26.25], [87.65348761904, -
↳ 129.951639498002, 17.39543648263], [87.65348761904, -129.951639498002, -17.
↳ 39543648263], [87.65348761904, -129.951639498002, -52.5]], [<pyaedt.modeler.cad.
↳ polylines.Polyline object at 0x00000258BE27E410>, [[91.38510737143, 33.261458938422, -52.5],
↳ [91.38510737143, 33.261458938422, 12.839087296526], [93.650624248243, 34.
↳ 086039646809, 15.25], [130.946772102377, 58.301259188538, 15.25], [133.149250451409, ↳
↳ 59.281865728298, 12.839087296526], [133.149250451409, 59.281865728298, -12.
↳ 839087296526], [130.946772102377, 58.301259188538, -15.25], [87.995362973656, 46.
↳ 787964435568, -15.25], [85.86665340553, 45.656109480928, -12.839087296526], [85.
↳ 86665340553, 45.656109480928, 12.839087296526], [87.995362973656, 46.787964435568, 15.
↳ 25], [121.558440086618, 75.958143676034, 15.25], [123.6030100148, 77.23573276199, 12.
↳ 839087296526], [123.6030100148, 77.23573276199, -12.839087296526], [121.558440086618, ↳
↳ 75.958143676034, -15.25], [80.627372052028, 58.579214717109, -15.25], [78.676902702964,
↳ 57.162115785443, -12.839087296526], [78.676902702964, 57.162115785443, 12.
↳ 839087296526], [80.627372052028, 58.579214717109, 15.25], [109.80411130525, 92.
↳ 136589297985, 15.25], [111.650977584591, 93.686294111813, 12.839087296526], [111.
↳ 650977584591, 93.686294111813, -12.839087296526], [109.80411130525, 92.136589297985, -
↳ 15.25], [71.690061045684, 69.230287217052, -15.25], [69.955795582933, 67.555526527137, ↳
↳ -12.839087296526], [69.955795582933, 67.555526527137, 12.839087296526], [71.
↳ 690061045684, 69.230287217052, 15.25], [95.912570397649, 106.521701013065, 15.25], [97.
↳ 525785876803, 108.31335831333, 12.839087296526], [97.525785876803, 108.31335831333, -12.839087296526], [95.912570397649, 106.521701013065, -15.25], [61.357384547322, 78.
↳ 533870924598, -15.25], [59.87307847542, 76.634045788254, -12.839087296526], [59.
↳ 87307847542, 76.634045788254, 12.839087296526], [61.357384547322, 78.533870924598, 15.
↳ 25], [80.154200406191, 118.833488984567, 15.25], [81.502365680861, 120.832226199897, ↳
↳ 12.839087296526], [81.502365680861, 120.832226199897, -12.839087296526], [80.
↳ 154200406191, 118.833488984567, -15.25], [49.830456351737, 86.308882165552, -15.25], ↳
↳ [48.625, 84.220970518037, -12.839087296526], [48.625, 84.220970518037, 12.
↳ 839087296526], [49.830456351737, 86.308882165552, 15.25], [62.835720077878, 128.
↳ 832318264074, 15.25], [63.892594644508, 130.999232248103, 12.839087296526], [63.
```

(continues on next page)

(continued from previous page)

↵ 892594644508, 130.999232248103, -12.839087296526], [62.835720077878, 128.832318264074, ↵ -15.25], [37.333635004569, 92.403989190051, -15.25], [36.430491209698, 90.16862985662, ↵ -12.839087296526], [36.430491209698, 90.16862985662, 12.839087296526], [37. ↵ 333635004569, 92.403989190051, 15.25], [44.294213932821, 136.32357301316, 15.25], [45. ↵ 039226930149, 138.616487250018, 12.839087296526], [45.039226930149, 138.616487250018, - ↵ 12.839087296526], [44.294213932821, 136.32357301316, -15.25], [24.110156918418, 96. ↵ 700557672946, -15.25], [23.526904347068, 94.36125938034, -12.839087296526], [23. ↵ 526904347068, 94.36125938034, 12.839087296526], [24.110156918418, 96.700557672946, 15. ↵ 25], [24.890571297483, 141.161444479313, 15.25], [25.309221894955, 143.535730001529, ↵ -12.839087296526], [25.309221894955, 143.535730001529, -12.839087296526], [24. ↵ 890571297483, 141.161444479313, -15.25], [10.417402052746, 99.11495979599, -15.25], ↵ [10.165393052779, 96.717254324565, -12.839087296526], [10.165393052779, 96. ↵ 717254324565, 12.839087296526], [10.417402052746, 99.11495979599, 15.25], [5. ↵ 002462004445, 143.251768997437, 15.25], [5.086601644389, 145.661213038034, 12. ↵ 839087296526], [5.086601644389, 145.661213038034, -12.839087296526], [5.002462004445, ↵ 143.251768997437, -15.25], [-3.478115694263, 99.600201968203, -15.25], [-3. ↵ 393976054318, 97.190757927607, -12.839087296526], [-3.393976054318, 97.190757927607, ↵ -12.839087296526], [-3.478115694263, 99.600201968203, 15.25], [-14.983014521293, 142. ↵ 5538607785, 15.25], [-15.23502352126, 144.951566249926, 12.839087296526], [-15.983014521293, 142.5538607785, - ↵ 15.25], [-17.305935875581, 98.146839502654, -15.25], [-16.887285278109, 95.772553980438, 15.117261889578], ↵ [-17.305935875581, 98.146839502654, 20.75], [-15.221148610561, 144.819555292762, 20. ↵ 75], [-15.809930069233, 150.421436674452, 15.117261889578], [-15.809930069233, 150. ↵ 421436674452, -15.117261889578], [-15.221148610561, 144.819555292762, -20.75], [-3. ↵ 398608547565, 97.323415177559, -20.75], [-3.202028822455, 91.694108379002, -15. ↵ 117261889578], [-3.202028822455, 91.694108379002, 15.117261889578], [-3.398608547565, ↵ 97.323415177559, 20.75], [5.081969151142, 145.528555788082, 20.75], [5.278548876253, ↵ 151.157862586639, 15.117261889578], [5.278548876253, 151.157862586639, -15. ↵ 117261889578], [5.081969151142, 145.528555788082, -20.75], [10.179267963479, 96. ↵ 849265281729, -20.75], [9.590486504807, 91.24738390004, -15.117261889578], [9. ↵ 590486504807, 91.24738390004, 15.117261889578], [10.179267963479, 96.849265281729, 20. ↵ 75], [25.286172163974, 143.405008481266, 20.75], [26.264286872124, 148.952172643097, ↵ 15.117261889578], [26.264286872124, 148.952172643097, -15.117261889578], [25. ↵ 286172163974, 143.405008481266, -20.75], [23.559016602359, 94.490054601597, -20.75], ↵ [22.196333921269, 89.024632885823, -15.117261889578], [22.196333921269, 89. ↵ 024632885823, 15.117261889578], [23.559016602359, 94.490054601597, 20.75], [44. ↵ 998208598227, 138.490245805141, 20.75], [46.738820399211, 143.847298089642, 15. ↵ 117261889578], [46.738820399211, 143.847298089642, -15.117261889578], [44.998208598227, ↵ 138.490245805141, -20.75], [36.48021578106, 90.291702489489, -20.75], [34.37015494591, ↵ 85.069118656503, -15.117261889578], [34.37015494591, 85.069118656503, 15. ↵ 117261889578], [36.48021578106, 90.291702489489, 20.75], [63.83440608682, 130. ↵ 879928024739, 20.75], [66.303635951848, 135.942599502749, 15.117261889578], [66. ↵ 303635951848, 135.942599502749, -15.117261889578], [63.83440608682, 130.879928024739, - ↵ 20.75], [48.691369055211, 84.335925093712, -20.75], [45.875, 79.457830797222, -15. ↵ 117261889578], [45.875, 79.457830797222, 15.117261889578], [48.691369055211, 84. ↵ 335925093712, 20.75], [81.428139471494, 120.722181319047, 20.75], [84.57792664995, 125.39193284895, -15.117261889578], ↵ [84.57792664995, 125.39193284895, -15.117261889578], [81.428139471494, 120.722181319047, -20.75], [59.954800216314, 76.73864484668, -20.75], ↵ [56.486940361129, 72.299986643417, -15.117261889578], [56.486940361129, 72. ↵ 299986643417, 15.117261889578], [59.954800216314, 76.73864484668, 20.75], [97. ↵ 43696674449, 108.214714673426, 20.75], [101.206004211777, 112.400654853456, 15.]

(continues on next page)

(continued from previous page)

```

→ [117261889578], [101.206004211777, 112.400654853456, -15.117261889578], [101.
→ 206004211777, 112.400654853456, -17.39543648263], [97.43696674449, 108.214714673426, -
→ 26.25], [58.552215885307, 74.943418768761, -26.25], [53.100802246838, 67.965927498579, -
→ -17.39543648263], [53.100802246838, 67.965927498579, 17.39543648263], [58.552215885307,
→ 74.943418768761, 26.25], [82.702078536796, 122.610873653528, 26.25], [87.65348761904, -
→ 129.951639498002, 17.39543648263], [87.65348761904, 129.951639498002, -17.39543648263],
→ [82.702078536796, 122.610873653528, -26.25], [47.552281758685, 82.362968021873, -26.
→ 25], [43.125, 74.694691076408, -17.39543648263], [43.125, 74.694691076408, 17.
→ 39543648263], [47.552281758685, 82.362968021873, 26.25], [64.833092095761, 132.
→ 927537785405, 26.25], [68.714677259187, 140.885966757394, 17.39543648263], [68.
→ 714677259187, 140.885966757394, -17.39543648263], [68.714677259187, 140.885966757394, -
→ 52.5]]]

```

## Assign excitations

Assign excitations.

```

first_winding_faces = m3d.modeler.get_object_faces(first_winding_list[0].name)
second_winding_faces = m3d.modeler.get_object_faces(second_winding_list[0].name)
third_winding_faces = m3d.modeler.get_object_faces(third_winding_list[0].name)
m3d.assign_current([first_winding_faces[-1]], amplitude=1000, phase="0deg", swap_
→ direction=False, name="phase_1_in")
m3d.assign_current([first_winding_faces[-2]], amplitude=1000, phase="0deg", swap_
→ direction=True, name="phase_1_out")
m3d.assign_current([second_winding_faces[-1]], amplitude=1000, phase="120deg", swap_
→ direction=False, name="phase_2_in")
m3d.assign_current([second_winding_faces[-2]], amplitude=1000, phase="120deg", swap_
→ direction=True, name="phase_2_out")
m3d.assign_current([third_winding_faces[-1]], amplitude=1000, phase="240deg", swap_
→ direction=False, name="phase_3_in")
m3d.assign_current([third_winding_faces[-2]], amplitude=1000, phase="240deg", swap_
→ direction=True, name="phase_3_out")

```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BBA92E90>
```

## Assign matrix

Assign the matrix.

```

m3d.assign_matrix(["phase_1_in", "phase_2_in", "phase_3_in"], matrix_name="current_matrix
→ ")

```

```
<pyaedt.modules.Boundary.MaxwellParameters object at 0x00000258BBA92EF0>
```

## Create mesh operation

Create the mesh operation.

```
mesh = m3d.mesh
mesh.assign_skin_depth(assignment=[first_winding_list[0], second_winding_list[0], third_
 ↪winding_list[0]], skin_depth=0.20, triangulation_max_length="10mm", name="skin_depth"
 ↪")
mesh.assign_surface_mesh_manual([first_winding_list[0], second_winding_list[0], third_
 ↪winding_list[0]], surface_deviation=None, normal_dev="30deg", name=
 ↪"surface_approx")
```

```
<pyaedt.modules.Mesh.MeshOperation object at 0x00000258BBA90BE0>
```

## Create boundaries

Create the boundaries. A region with openings is needed to run the analysis.

```
region = m3d.modeler.create_air_region(x_pos=100, y_pos=100, z_pos=100, x_neg=100, y_
 ↪neg=100, z_neg=0)
```

## Create setup

Create a setup with a sweep to run the simulation. Depending on your machines computing power, the simulation can take some time to run.

```
setup = m3d.create_setup("MySetup")
print(setup.props)
setup.props["Frequency"] = "100kHz"
setup.props["PercentRefinement"] = 15
setup.props["MaximumPasses"] = 10
setup.props["HasSweepSetup"] = True
setup.add_eddy_current_sweep(range_type="LinearCount", start=100, end=1000, count=12, u
 ↪nits="kHz", clear=True)
```

```
SetupProps([('Enabled', True), ('MeshLink', SetupProps([('ImportMesh', False)])), (
 ↪'MaximumPasses', 6), ('MinimumPasses', 1), ('MinimumConvergedPasses', 1), (
 ↪'PercentRefinement', 30), ('SolveFieldOnly', False), ('PercentError', 1), (
 ↪'SolveMatrixAtLast', True), ('UseIterativeSolver', False), ('RelativeResidual', 1e-05),
 ↪('NonLinearResidual', 0.0001), ('SmoothBHCurve', False), ('Frequency', '60Hz'), (
 ↪'HasSweepSetup', False), ('SweepRanges', SetupProps([('Subrange', SetupProps([
 ↪'SweepSetupType', 'LinearStep'), ('StartValue', '1e-08GHz'), ('StopValue', '1e-06GHz'),
 ↪('StepSize', '1e-08GHz')]))]), ('UseHighOrderShapeFunc', False), ('UseMuLink', u
 ↪False)])]
```

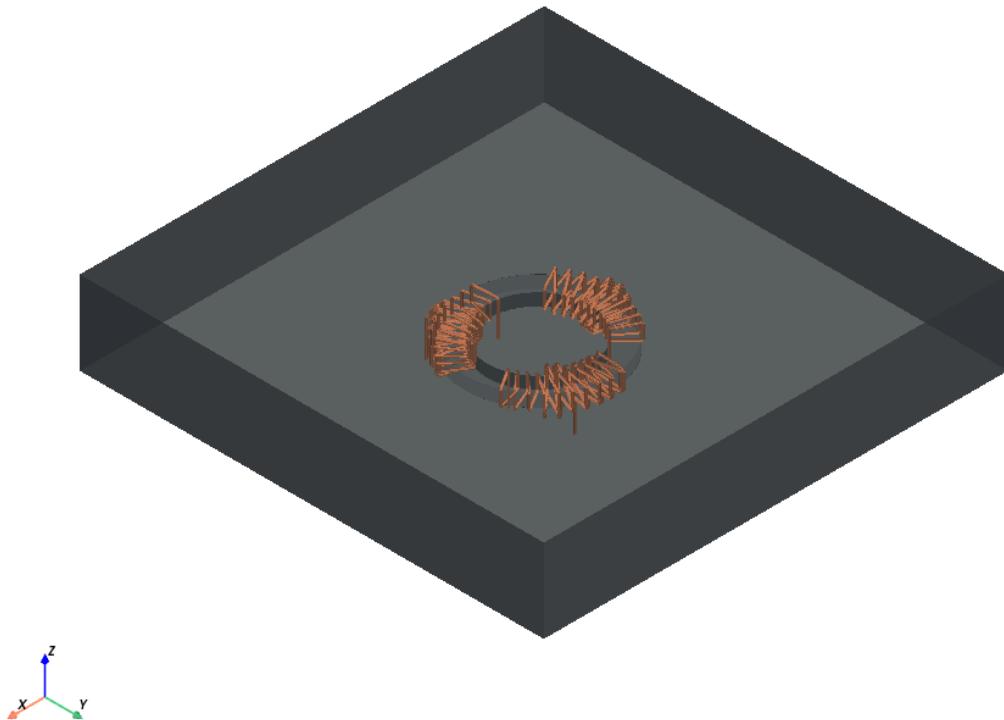
```
True
```

## Save project

Save the project.

```
m3d.save_project()
m3d.modeler.fit_all()
m3d.plot(show=False, export_path=os.path.join(temp_dir.name, "Image.jpg"), plot_air_
objects=True)
```

Core  
Winding  
Winding\_1  
Winding\_2  
Region



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BBA92380>
```

## Close AEDT

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

```
m3d.release_desktop()
temp_dir.cleanup()
```

**Total running time of the script:** (1 minutes 16.546 seconds)

## Maxwell 3D: magnet segmentation

This example shows how you can use PyAEDT to segment magnets of an electric motor. The method is valid and usable for any object the user would like to segment.

### Perform required imports

Perform required imports.

```
from pyaedt import downloads
from pyaedt import Maxwell3d

import tempfile
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Create temporary directory

Create temporary directory.

```
temp_dir = tempfile.TemporaryDirectory(suffix=".ansys")
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

### Download .aedt file example

Set local temporary folder to export the .aedt file to.

```
aedt_file = downloads.download_file("object_segmentation", "Motor3D_obj_segments.aedt",
 temp_dir.name)
```

## Launch Maxwell 3D

Launch Maxwell 3D.

```
m3d = Maxwell3d(projectname=aedt_file,
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 10000 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create object to access 3D modeler

Create the object mod3D to access the 3D modeler easily.

```
modeler = m3d.modeler
```

## Segment first magnet by specifying the number of segments

Select first magnet to segment by specifying the number of segments. The method accepts in input either the list of magnets names to segment or magnets ids or the magnet object *pyaedt.modeler.cad.object3d.Object3d*.*apply\_mesh\_sheets* is enabled which means that the mesh sheets will be applied in the geometry too. In this specific case we give as input the name of the magnet.

```
segments_number = 5
object_name = "PM_I1"
sheets_1 = modeler.objects_segmentation(object_name, segments_number=segments_number,
 apply_mesh_sheets=True)
```

## Segment second magnet by specifying the number of segments

Select second magnet to segment by specifying the number of segments. In this specific case we give as input the id of the magnet.

```
segments_number = 4
object_name = "PM_I1_1"
magnet_id = [obj.id for obj in modeler.object_list if obj.name == object_name][0]
sheets_2 = modeler.objects_segmentation(magnet_id, segments_number=segments_number,
 apply_mesh_sheets=True)
```

## Segment third magnet by specifying the segmentation thickness

Select third magnet to segment by specifying the segmentation thickness. In this specific case we give as input the magnet object of type `pyaedt.modeler.cad.object3d.Object3d`.

```
segmentation_thickness = 1
object_name = "PM_01"
magnet = [obj for obj in modeler.object_list if obj.name == object_name][0]
sheets_3 = modeler.objects_segmentation(magnet, segmentation_thickness=segmentation_
thickness, apply_mesh_sheets=True)
```

## Segment fourth magnet by specifying the number of segments

Select fourth magnet to segment by specifying the number of segments. In this specific case we give as input the name of the magnet and we disable the mesh sheets.

```
object_name = "PM_01_1"
segments_number = 10
sheets_4 = modeler.objects_segmentation(object_name, segments_number=segments_number)
```

## Save project and close AEDT

Save the project and close AEDT.

```
m3d.save_project()
m3d.release_desktop()
temp_dir.cleanup()
```

**Total running time of the script:** (1 minutes 19.789 seconds)

## Maxwell 3D: bath plate analysis

This example uses PyAEDT to set up the TEAM 3 bath plate problem and solve it using the Maxwell 3D Eddy Current solver. <https://www.compumag.org/wp/wp-content/uploads/2018/06/problem3.pdf>

## Perform required imports

Perform required imports.

```
import os
import pyaedt
import tempfile
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Create temporary directory

Create temporary directory.

```
temp_dir = tempfile.TemporaryDirectory(suffix=".ansys")
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Launch AEDT and Maxwell 3D

Launch AEDT and Maxwell 3D after first setting up the project and design names, the solver, and the version. The following code also creates an instance of the `Maxwell3d` class named `M3D`.

```
project_name = "COMPUMAG"
design_name = "TEAM 3 Bath Plate"
solver = "EddyCurrent"

m3d = pyaedt.Maxwell3d(
 projectname=project_name,
 designname=design_name,
 solution_type=solver,
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True,
)
m3d.modeler.model_units = "mm"
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 7444 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Add variable

Add a design variable named Coil\_Position that you use later to adjust the position of the coil.

```
Coil_Position = -20
m3d["Coil_Position"] = str(Coil_Position) + m3d.modeler.model_units
```

## Add material

Add a material named team3\_aluminium for the ladder plate.

```
mat = m3d.materials.add_material("team3_aluminium")
mat.conductivity = 32780000
```

## Draw background region

Draw a background region that uses the default properties for an air region.

```
m3d.modeler.create_air_region(x_pos=100, y_pos=100, z_pos=100, x_neg=100, y_neg=100, z_-
neg=100)
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258A9B1F0A0>
```

## Draw ladder plate and assign material

Draw a ladder plate and assign it the newly created material team3\_aluminium.

```
m3d.modeler.create_box(origin=[-30, -55, 0], sizes=[60, 110, -6.35], name="LadderPlate", u_
material="team3_aluminium")
m3d.modeler.create_box(origin=[-20, -35, 0], sizes=[40, 30, -6.35], name="CutoutTool1")
m3d.modeler.create_box(origin=[-20, 5, 0], sizes=[40, 30, -6.35], name="CutoutTool2")
m3d.modeler.subtract("LadderPlate", ["CutoutTool1", "CutoutTool2"], keep_originals=False)
```

```
True
```

## Add mesh refinement to ladder plate

Add a mesh refinement to the ladder plate.

```
m3d.mesh.assign_length_mesh("LadderPlate", maximum_length=3, maximum_elements=None, name=
"")
Ladder_Mesh")
```

```
<pyaedt.modules.Mesh.MeshOperation object at 0x00000258A9B1F550>
```

## Draw search coil and assign excitation

Draw a search coil and assign it a stranded current excitation. The stranded type forces the current density to be constant in the coil.

```
m3d.modeler.create_cylinder(orientation="Z", origin=[0, "Coil_Position", 15], radius=40, height=20, name="SearchCoil", material="copper")
m3d.modeler.create_cylinder(orientation="Z", origin=[0, "Coil_Position", 15], radius=20, height=20, name="Bore", material="copper")
m3d.modeler.subtract("SearchCoil", "Bore", keep_originals=False)
m3d.modeler.section("SearchCoil", "YZ")
m3d.modeler.separate_bodies("SearchCoil_Section1")
m3d.modeler.delete("SearchCoil_Section1_Separate1")
m3d.assign_current(assignment=["SearchCoil_Section1"], amplitude=1260, solid=False, name="SearchCoil_Excitation")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE2FFB50>
```

## Draw a line for plotting Bz

Draw a line for plotting Bz later. Bz is the Z component of the flux density. The following code also adds a small diameter cylinder to refine the mesh locally around the line.

```
line_points = [[["0mm", "-55mm", "0.5mm"], ["0mm", "55mm", "0.5mm"]]]
m3d.modeler.create_polyline(points=line_points, name="Line_AB")
poly = m3d.modeler.create_polyline(points=line_points, name="Line_AB_MeshRefinement")
poly.set_crosssection_properties(type="Circle", width="0.5mm")
```

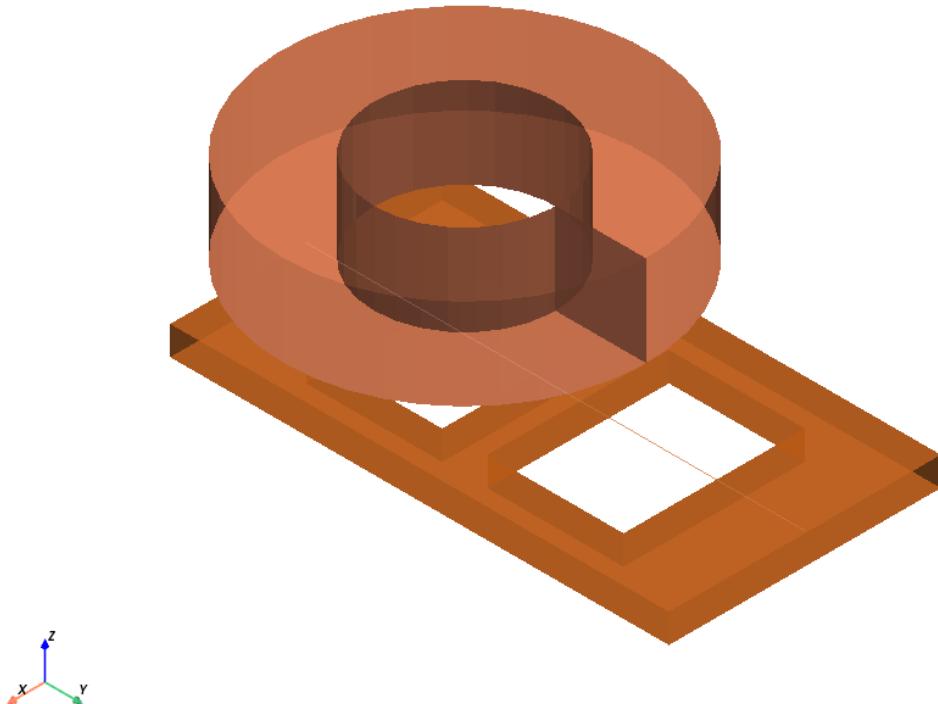
```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BE2FF940>
```

## Plot model

Plot the model.

```
m3d.plot(show=False, export_path=os.path.join(temp_dir.name, "Image.jpg"), plot_air_objects=False)
```

	LadderPlate
	SearchCoil
	SearchCoil_Section1
	Line_AB



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BE2FDA20>
```

### Add Maxwell 3D setup

Add a Maxwell 3D setup with frequency points at 50 Hz and 200 Hz.

```
setup = m3d.create_setup(name="Setup1")
setup.props["Frequency"] = "200Hz"
setup.props["HasSweepSetup"] = True
setup.add_eddy_current_sweep(range_type="LinearStep", start=50, end=200, count=150,
 clear=True)
```

```
True
```

## Adjust eddy effects

Adjust eddy effects for the ladder plate and the search coil. The setting for eddy effect is ignored for the stranded conductor type used in the search coil.

```
m3d.eddy_effects_on(assignment=["LadderPlate"], enable_eddy_effects=True, enable_
 ↴displacement_current=True)
m3d.eddy_effects_on(assignment=["SearchCoil"], enable_eddy_effects=False, enable_
 ↴displacement_current=True)
```

True

## Add linear parametric sweep

Add a linear parametric sweep for the two coil positions.

```
sweep_name = "CoilSweep"
param = m3d.parametrics.add("Coil_Position", -20, 0, 20, "LinearStep", True
 ↵parametricname=sweep_name)
param["SaveFields"] = True
param["CopyMesh"] = False
param["SolveWithCopiedMeshOnly"] = True
```

## Solve parametric sweep

Solve the parametric sweep directly so that results of all variations are available.

```
m3d.analyze_setup(sweep_name)
```

True

## Create expression for Bz

Create an expression for Bz using the fields calculator.

```
Fields = m3d.ofieldsreporter
Fields.EnterQty("B")
Fields.CalcOp("ScalarZ")
Fields.EnterScalar(1000)
Fields.CalcOp("*")
Fields.CalcOp("Smooth")
Fields.AddNamedExpression("Bz", "Fields")
```

## Plot mag(Bz) as a function of frequency

Plot mag(Bz) as a function of frequency for both coil positions.

```
variations = {"Distance": ["All"], "Freq": ["All"], "Phase": ["0deg"], "Coil_Position": [All]}
m3d.post.create_report(expressions="mag(Bz)", variations=variations, primary_sweep_variable="Distance",
report_category="Fields", context="Line_AB", plot_name="mag(Bz) Along 'Line_AB' Coil")
```

```
<pyaedt.modules.report_templates.Fields object at 0x00000258BE2FDBD0>
```

## Get simulation results from a solved setup

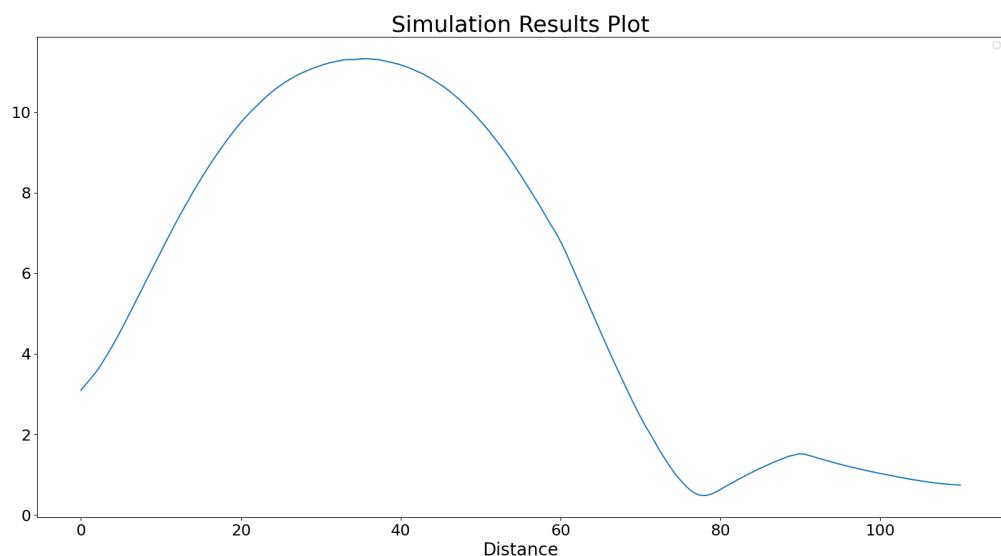
Get simulation results from a solved setup as a SolutionData object.

```
solutions = m3d.post.get_solution_data(
 expressions="mag(Bz)",
 report_category="Fields",
 context="Line_AB",
 variations=variations,
 primary_sweep_variable="Distance",
)
```

## Set up sweep value and plot solution

Set up a sweep value and plot the solution.

```
solutions.active_variation["Coil_Position"] = -0.02
solutions.plot()
```



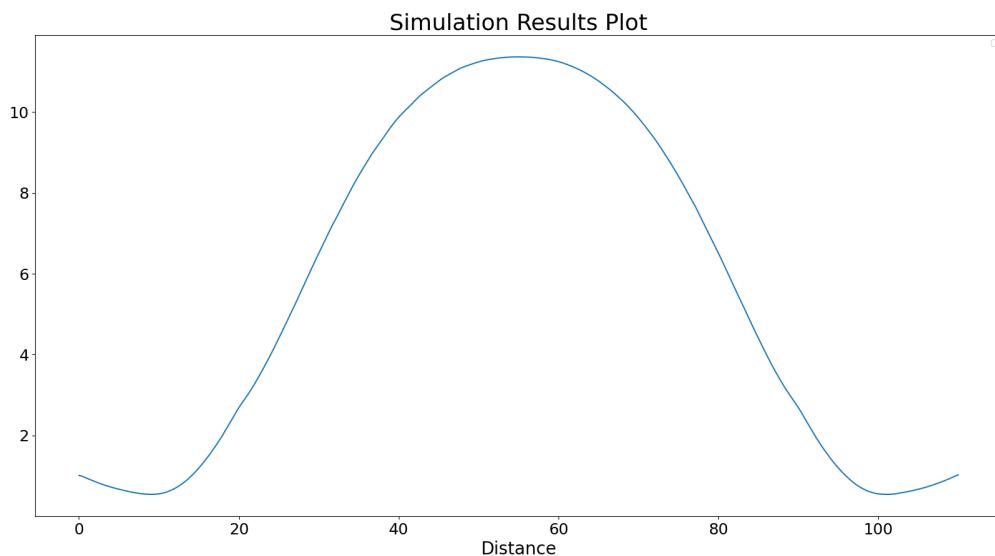
No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

### Change sweep value and plot solution

Change the sweep value and plot the solution again.

```
solutions.active_variation["Coil_Position"] = 0
solutions.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

### Plot induced current density on surface of ladder plate

Plot the induced current density, "Mag\_J", on the surface of the ladder plate.

```
ladder_plate = m3d.modeler.objects_by_name["LadderPlate"]
intrinsic_dict = {"Freq": "50Hz", "Phase": "0deg"}
m3d.post.create_fieldplot_surface(ladder_plate.faces, "Mag_J", intrinsics=intrinsic_dict,
plot_name="Mag_J")
```

<`pyaedt.modules.solutions.FieldPlot` object at 0x00000258BE29B400>

## Release AEDT

Release AEDT from the script engine, leaving both AEDT and the project open.

```
m3d.release_desktop(False, False)
temp_dir.cleanup()
```

**Total running time of the script:** (14 minutes 37.229 seconds)

## Enabling Control Program in a Maxwell 2D Project

This example shows how you can use PyAEDT to enable control program in a Maxwell 2D project. It shows how to create the geometry, load material properties from an Excel file and set up the mesh settings. Moreover, it focuses on post-processing operations, in particular how to plot field line traces, relevant for an electrostatic analysis.

### Perform required imports

Perform required imports.

```
from pyaedt import downloads
from pyaedt import generate_unique_folder_name
from pyaedt import Maxwell2d
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

### Download .aedt file example

Set local temporary folder to export the .aedt file to.

```
temp_folder = generate_unique_folder_name()
aedt_file = downloads.download_file("maxwell_ctrl_prg", "ControlProgramDemo.aedt", temp_
 ↴folder)
ctrl_prg_file = downloads.download_file("maxwell_ctrl_prg", "timestep_only.py", temp_
 ↴folder)
```

## Launch Maxwell 2D

Launch Maxwell 2D.

```
m2d = Maxwell2d(projectname=aedt_file,
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 4976 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Set active design

Set active design.

```
m2d.set_active_design("1 time step control")
```

```
True
```

## Get design setup

Get design setup to enable the control program to.

```
setup = m2d.setups[0]
```

## Enable control program

Enable control program by giving the path to the file.

```
setup.enable_control_program(control_program_path=ctrl_prg_file)
```

```
True
```

## Analyze setup

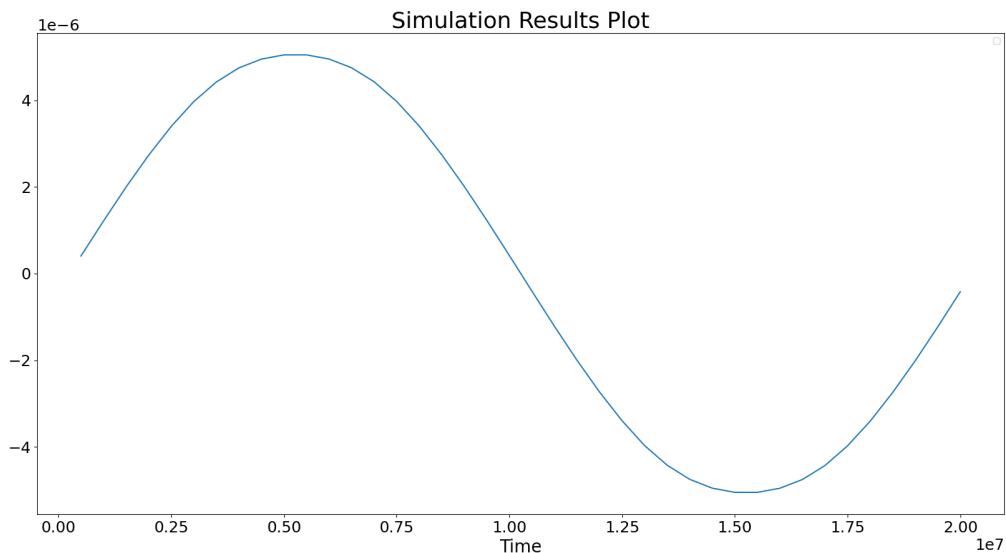
Analyze setup.

```
setup.analyze()
```

## Plot results

Plot Solved Results.

```
sols = m2d.post.get_solution_data("FluxLinkage(Winding1)", variations={"Time": ["All"]},
 primary_sweep_variable="Time")
sols.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Save project and close AEDT

Save the project and close AEDT.

```
m2d.save_project()
m2d.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 9.347 seconds)

## Maxwell 3D: magnet DC analysis

This example shows how you can use PyAEDT to create a Maxwell DC analysis, compute mass center, and move coordinate systems.

### Perform required imports

Perform required imports.

```
from pyaedt import Maxwell3d
from pyaedt import generate_unique_project_name
import os
import tempfile
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Create temporary directory

Create temporary directory.

```
temp_dir = tempfile.TemporaryDirectory(suffix=".ansys")
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

### Launch AEDT

Launch AEDT in graphical mode.

```
m3d = Maxwell3d(projectname=generate_unique_project_name(),
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 2072 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Set up Maxwell solution

Set up the Maxwell solution to DC.

```
m3d.solution_type = m3d.SOLUTIONS.Maxwell3d.ElectroDCConduction
```

## Create magnet

Create a magnet.

```
magnet = m3d.modeler.create_box(origin=[7, 4, 22], sizes=[10, 5, 30], name="Magnet",
material="copper")
```

## Create setup and assign voltage

Create the setup and assign a voltage.

```
m3d.assign_voltage(magnet.faces, 0)
m3d.create_setup()
```

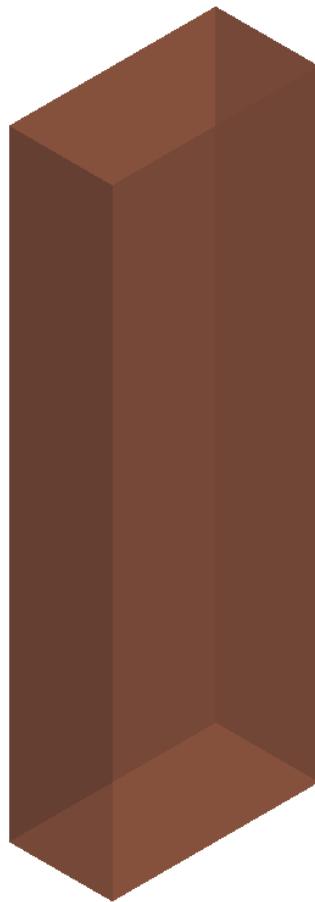
```
SetupName MySetupAuto with 0 Sweeps
```

## Plot model

Plot the model.

```
m3d.plot(show=False, export_path=os.path.join(temp_dir.name, "Image.jpg"), plot_air_
objects=True)
```

■ Magnet



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258AB1CFA90>
```

## Solve setup

Solve the setup.

```
m3d.analyze()
```

```
True
```

## Compute mass center

Compute mass center using the fields calculator.

```
m3d.post.ofieldsreporter.EnterScalarFunc("X")
m3d.post.ofieldsreporter.EnterVol(magnet.name)
m3d.post.ofieldsreporter.CalcOp("Mean")
m3d.post.ofieldsreporter.AddNamedExpression("CM_X", "Fields")
m3d.post.ofieldsreporter.EnterScalarFunc("Y")
m3d.post.ofieldsreporter.EnterVol(magnet.name)
```

(continues on next page)

(continued from previous page)

```
m3d.post.ofieldsreporter.CalcOp("Mean")
m3d.post.ofieldsreporter.AddNamedExpression("CM_Y", "Fields")
m3d.post.ofieldsreporter.EnterScalarFunc("Z")
m3d.post.ofieldsreporter.EnterVol(magnet.name)
m3d.post.ofieldsreporter.CalcOp("Mean")
m3d.post.ofieldsreporter.AddNamedExpression("CM_Z", "Fields")
m3d.post.ofieldsreporter.CalcStack("clear")
```

## Get mass center

Get mass center using the fields calculator.

```
xval = m3d.post.get_scalar_field_value("CM_X")
yval = m3d.post.get_scalar_field_value("CM_Y")
zval = m3d.post.get_scalar_field_value("CM_Z")
```

## Create variables

Create variables with mass center values.

```
m3d[magnet.name + "x"] = str(xval * 1e3) + "mm"
m3d[magnet.name + "y"] = str(yval * 1e3) + "mm"
m3d[magnet.name + "z"] = str(zval * 1e3) + "mm"
```

## Create coordinate system

Create a parametric coordinate system.

```
cs1 = m3d.modeler.create_coordinate_system(
 [magnet.name + "x", magnet.name + "y", magnet.name + "z"], reference_cs="Global",
 name=magnet.name + "CS"
)
```

## Save and close

Save the project and close AEDT.

```
m3d.save_project()
m3d.release_desktop(close_projects=True, close_desktop=True)
temp_dir.cleanup()
```

**Total running time of the script:** (1 minutes 17.098 seconds)

## Maxwell 3D: Transformer

This example shows how you can use PyAEDT to set core loss given a set of Power-Volume [kw/m<sup>3</sup>] curves at different frequencies.

### Perform required imports

Perform required imports.

```
from pyaedt import downloads
from pyaedt import generate_unique_folder_name
from pyaedt import Maxwell3d
from pyaedt.generic.constants import unit_converter
from pyaedt.generic.general_methods import read_csv_pandas
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Download .aedt file example

Set local temporary folder to export the .aedt file to.

```
temp_folder = generate_unique_folder_name()
aedt_file = downloads.download_file("core_loss_transformer", "Ex2-PlanarTransformer_"
 ↪ 2023R2.aedtz", temp_folder)
freq_curve_csv_25kHz = downloads.download_file("core_loss_transformer", "mf3_25kHz.csv", ↪
 ↪ temp_folder)
freq_curve_csv_100kHz = downloads.download_file("core_loss_transformer", "mf3_100kHz.csv",
 ↪", temp_folder)
freq_curve_csv_200kHz = downloads.download_file("core_loss_transformer", "mf3_200kHz.csv",
 ↪", temp_folder)
freq_curve_csv_400kHz = downloads.download_file("core_loss_transformer", "mf3_400kHz.csv",
 ↪", temp_folder)
freq_curve_csv_700kHz = downloads.download_file("core_loss_transformer", "mf3_700kHz.csv",
 ↪", temp_folder)
freq_curve_csv_1MHz = downloads.download_file("core_loss_transformer", "mf3_1MHz.csv", ↪
 ↪ temp_folder)

data = read_csv_pandas(filename=freq_curve_csv_25kHz)
curves_csv_25kHz = list(zip(data[data.columns[0]],
 data[data.columns[1]]))
data = read_csv_pandas(filename=freq_curve_csv_100kHz)
curves_csv_100kHz = list(zip(data[data.columns[0]],
 data[data.columns[1]]))
data = read_csv_pandas(filename=freq_curve_csv_200kHz)
curves_csv_200kHz = list(zip(data[data.columns[0]],
 data[data.columns[1]]))
```

(continues on next page)

(continued from previous page)

```
data = read_csv_pandas(filename=freq_curve_csv_400kHz)
curves_csv_400kHz = list(zip(data[data.columns[0]],
 data[data.columns[1]]))
data = read_csv_pandas(filename=freq_curve_csv_700kHz)
curves_csv_700kHz = list(zip(data[data.columns[0]],
 data[data.columns[1]]))
data = read_csv_pandas(filename=freq_curve_csv_1MHz)
curves_csv_1MHz = list(zip(data[data.columns[0]],
 data[data.columns[1]]))
```

## Launch AEDT

Launch AEDT in graphical mode.

```
m3d = Maxwell3d(projectname=aedt_file,
 designname="02_3D_eddycurrent_CmXY_for_thermal",
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=False)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 8024 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Set core loss at frequencies

Create a new material, create a dictionary of Power-Volume [kw/m<sup>3</sup>] points for a set of frequencies retrieved from datasheet provided by supplier and finally set Power-Ferrite core loss model.

```
mat = m3d.materials.add_material("newmat")
freq_25kHz = unit_converter(25, unit_system="Freq", input_units="kHz", output_units="Hz")
freq_100kHz = unit_converter(100, unit_system="Freq", input_units="kHz", output_units="Hz")
freq_200kHz = unit_converter(200, unit_system="Freq", input_units="kHz", output_units="Hz")
freq_400kHz = unit_converter(400, unit_system="Freq", input_units="kHz", output_units="Hz")
freq_700kHz = unit_converter(700, unit_system="Freq", input_units="kHz", output_units="Hz")
pv = {freq_25kHz: curves_csv_25kHz,
 freq_100kHz: curves_csv_100kHz,
 freq_200kHz: curves_csv_200kHz,
 freq_400kHz: curves_csv_400kHz,
 freq_700kHz: curves_csv_700kHz}
m3d.materials[mat.name].set_coreloss_at_frequency(points_list_at_freq=pv,
```

(continues on next page)

(continued from previous page)

```
coefficient_setup="kw_per_cubic_meter",
core_loss_model_type="Power Ferrite")
coefficients = m3d.materials[mat.name].get_core_loss_coefficients(points_at_frequency=pv,
 coefficient_setup="kw_
 →per_cubic_meter")
```

## Save project and close AEDT

Save the project and close AEDT.

```
m3d.save_project()
m3d.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 47.847 seconds)

## 3.12.7 Icepak examples

These examples use PyAEDT to show end-to-end workflows for Icepak. This includes schematic generation, setup, and thermal postprocessing.

### Icepak: thermal analysis with 3D components

This example shows how to create a thermal analysis of an electronic package by taking advantage of 3D components and features added by PyAEDT.

#### Import PyAEDT and download files

Perform import of required classes from the pyaedt package and import the os package.

```
from pyaedt import Icepak, generate_unique_folder_name, downloads, settings
import os

Download needed files

temp_folder = generate_unique_folder_name()
package_temp_name, qfp_temp_name = downloads.download_icepak_3d_component(temp_folder)
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Create heatsink

Open a new project in non-graphical mode.

```
ipk = Icepak(projectname=os.path.join(temp_folder, "Heatsink.aedt"),
 specified_version=aedt_version,
 non_graphical=non_graphical,
 close_on_exit=True,
 new_desktop_session=True)

Remove air region created by default because it is not needed as the heatsink will be
→exported as a 3DComponent.

ipk.modeler.get_object_from_name("Region").delete()

Definition of heatsink with boxes
hs_base = ipk.modeler.create_box(origin=[0, 0, 0], sizes=[37.5, 37.5, 2], name="HS_Base")
hs_base.material_name = "Al-Extruded"
hs_fin = ipk.modeler.create_box(origin=[0, 0, 2], sizes=[37.5, 1, 18], name="HS_Fin1")
hs_fin.material_name = "Al-Extruded"
hs_fin.duplicate_along_line([0, 3.65, 0], nclones=11)

ipk.plot(show=False, export_path=os.path.join(temp_folder, "Heatsink.jpg"))

Definition of a mesh region. First a non-model box is created, then the mesh region is
→assigned
mesh_box = ipk.modeler.create_box(origin=[-2, -2, -3], sizes=[41.5, 41.5, 24])
mesh_box.model = False
mesh_region = ipk.mesh.assign_mesh_region([mesh_box.name])
mesh_region.UserSpecifiedSettings = True
mesh_region.MaxElementSizeX = "5mm"
mesh_region.MaxElementSizeY = "5mm"
mesh_region.MaxElementSizeZ = "1mm"
mesh_region.MinElementsInGap = "4"
mesh_region.MaxLevels = "2"
mesh_region.BufferLayers = "1"
mesh_region.update()
```

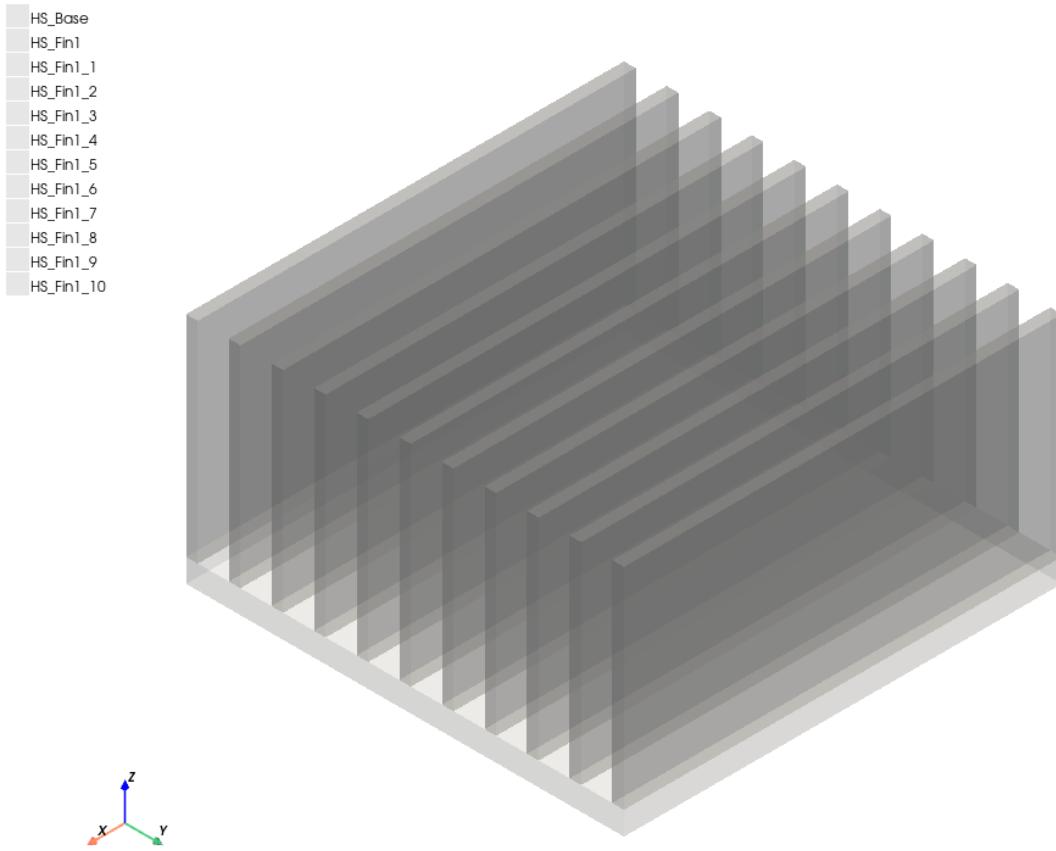
(continues on next page)

(continued from previous page)

```
Assignment of monitor objects.
hs_fin5_object = ipk.modeler.get_object_from_name("HS_Fin1_5")
point_monitor_position = [0.5 * (hs_base.bounding_box[i] + hs_base.bounding_box[i + 3])_
 ↪for i in range(2)] + [
 hs_fin5_object.bounding_box[-1]] # average x,y, top z

ipk.monitor.assign_point_monitor(point_monitor_position, monitor_quantity=["Temperature",
 ↪ "HeatFlux"],
 monitor_name="TopPoint")
ipk.monitor.assign_face_monitor(hs_base.bottom_face_z.id, monitor_quantity="Temperature",
 ↪ monitor_name="Bottom")
ipk.monitor.assign_point_monitor_in_object("HS_Fin1_5", monitor_quantity="Temperature",
 ↪ monitor_name="Fin5Center")

Export the heatsink 3D component and close project. auxiliary_dict is set to true in
order to export the
monitor objects along with the .a3dcomp file.
os.mkdir(os.path.join(temp_folder, "componentLibrary"))
ipk.modeler.create_3dcomponent(
 os.path.join(temp_folder, "componentLibrary", "Heatsink.a3dcomp"),
 component_name="Heatsink",
 auxiliary_dict=True
)
ipk.close_project(save_project=False)
```



```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 11040 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

True

## Create QFP

Download and open a project containing a QPF.

```
ipk = Icepak(projectname=qfp_temp_name)
ipk.plot(show=False, export_path=os.path.join(temp_folder, "QFP2.jpg"))

Create dataset for power dissipation.
x_datalist = [45, 53, 60, 70]
y_datalist = [0.5, 3, 6, 9]
ipk.create_dataset(
 "PowerDissipationDataset",
```

(continues on next page)

(continued from previous page)

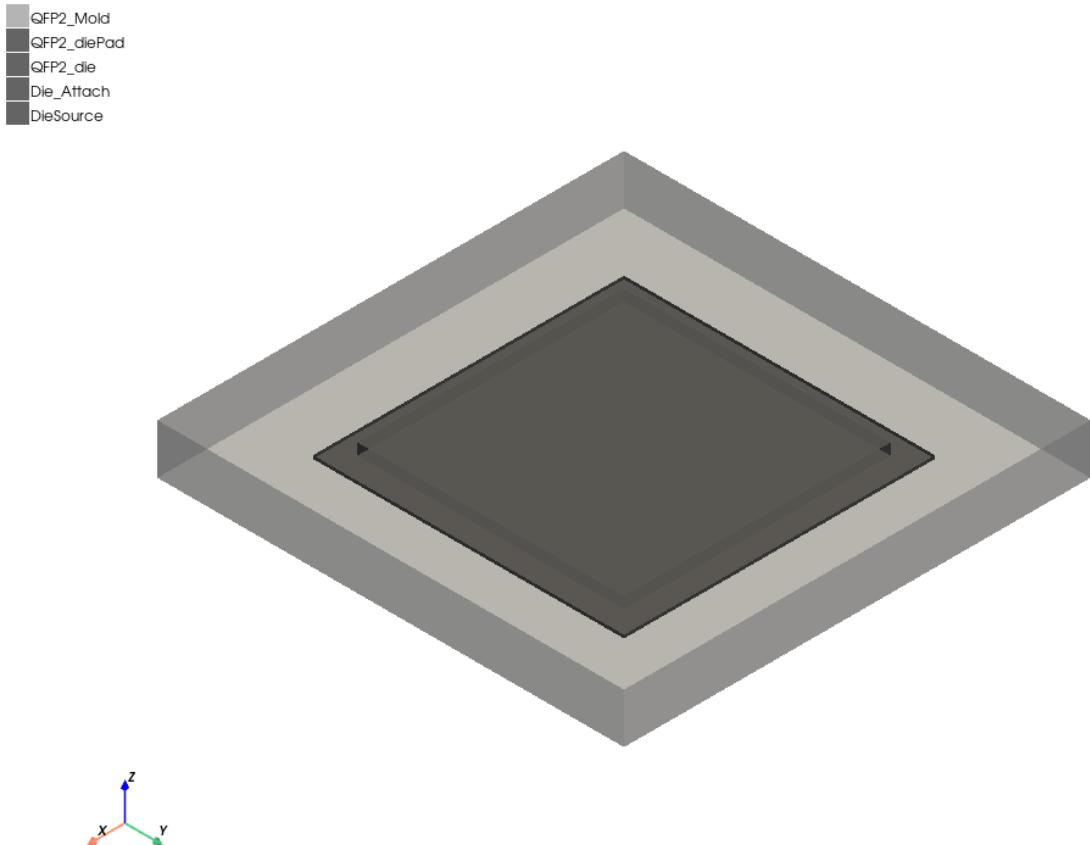
```
x_datalist,
y_datalist,
zlist=None,
vlist=None,
is_project_dataset=False,
xunit="cel",
yunit="W",
zunit="",
vunit="",
)

Assign source power condition to the die.
ipk.create_source_power(
 "DieSource",
 thermal_dependent_dataset="PowerDissipationDataset",
 source_name="DieSource"
)

Assign thickness to the die attach surface.
ipk.create_conducting_plate(face_id="Die_Attach",
 thermal_specification="Thickness",
 shell_conduction=True,
 thickness="0.05mm",
 solid_material="Epoxy Resin-Typical",
 bc_name="Die_Attach",
)

Assign monitor objects.
ipk.monitor.assign_point_monitor_in_object("QFP2_die", monitor_quantity="Temperature",
 monitor_name="DieCenter")
ipk.monitor.assign_surface_monitor("Die_Attach", monitor_quantity="Temperature", monitor_name="DieAttach")

Export the QFP 3D component and close project. Here the auxiliary dictionary allows
exporting not only the monitor
objects but also the dataset used for the power source assignment.
ipk.modeler.create_3dcomponent(
 os.path.join(temp_folder, "componentLibrary", "QFP.a3dcomp"),
 component_name="QFP",
 auxiliary_dict=True,
 datasets=["PowerDissipationDataset"]
)
ipk.release_desktop(False, False)
```



```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\icepak.py:591:_
DeprecationWarning: This method is deprecated in 0.7.8. Use the ``assign_conducting_
plate()`` method.
warnings.warn(
```

```
True
```

## Create electronic package

Download and open a project containing the electronic package.

```
ipk = Icepak(projectname=package_temp_name,
 specified_version=aedt_version,
 non_graphical=non_graphical)
ipk.plot(show=False, export_path=os.path.join(temp_folder, "electronic_package_missing_"
 "obj.jpg"))

The heatsink and the QFP are missing. They can be inserted as 3d components. The_
auxiliary files are needed since
the aim is to import also monitor objects and datasets. Also, a coordinate system is_
created for the heatsink so
that it is placed on top of the AGP.
```

(continues on next page)

(continued from previous page)

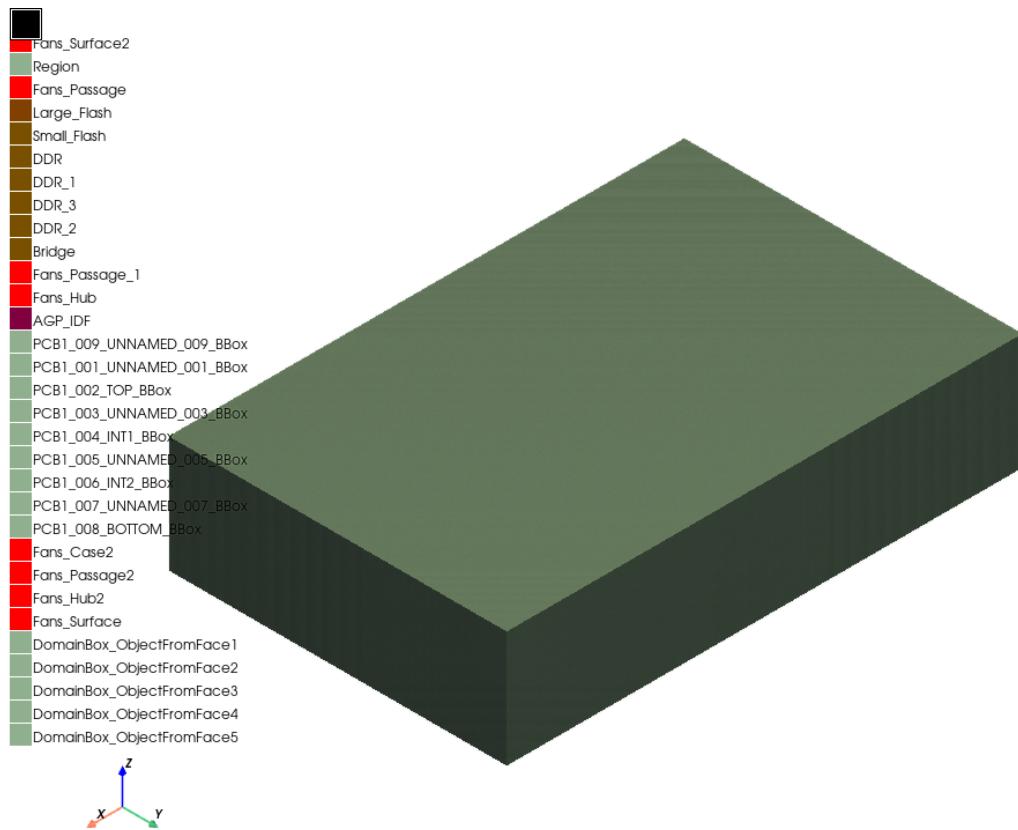
```

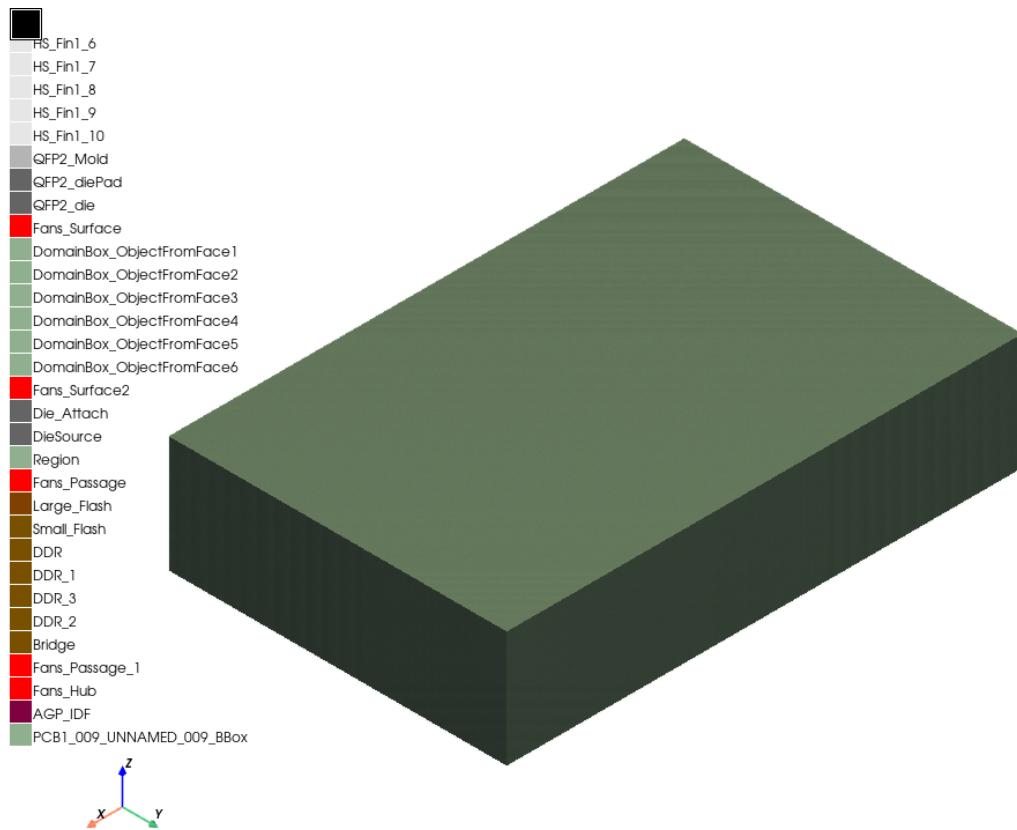
agp = ipk.modeler.get_object_from_name("AGP_IDF")
cs = ipk.modeler.create_coordinate_system(
 origin=[agp.bounding_box[0], agp.bounding_box[1], agp.bounding_box[-1]],
 name="HeatsinkCS",
 reference_cs="Global",
 x_pointing=[1, 0, 0],
 y_pointing=[0, 1, 0],
)
heatsink_obj = ipk.modeler.insert_3d_component(
 input_file=os.path.join(temp_folder, "componentLibrary", "Heatsink.a3dcomp"),
 coordinate_system="HeatsinkCS",
 auxiliary_parameters=True)

QFP2_obj = ipk.modeler.insert_3d_component(input_file=os.path.join(temp_folder,
 "componentLibrary", "QFP.a3dcomp"),
 coordinate_system="Global", auxiliary_
parameters=True)
ipk.plot(show=False, export_path=os.path.join(temp_folder, "electronic_package.jpg"))

Create a coordinate system at the xmin, ymin, zmin of the model
bounding_box = ipk.modeler.get_model_bounding_box()
cs_pcb_assembly = ipk.modeler.create_coordinate_system(
 origin=[bounding_box[0], bounding_box[1], bounding_box[2]],
 name="PCB_Assembly",
 reference_cs="Global",
 x_pointing=[1, 0, 0],
 y_pointing=[0, 1, 0],
)
Export of the whole assembly as 3d component and close project. First, a flattening is_
#needed because nested 3d
components are not natively supported. Then it is possible to export the whole package_
#as 3d component. Here the
auxiliary dictionary is needed to export monitor objects, datasets and native_
#components.
ipk.flatten_3d_components()
ipk.modeler.create_3dcomponent(
 component_file=os.path.join(temp_folder, "componentLibrary", "PCBAassembly.a3dcomp"),
 component_name="PCBAassembly",
 auxiliary_dict=True,
 included_cs=["Global", "HeatsinkCS", "PCB_Assembly"],
 reference_cs="PCB_Assembly"
)

```





```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 10576 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\modeled\cad\
 Primitives3D.py:1460: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\
 pyaedt_prj_86B\\componentLibrary\\Heatsink.a3dcomp.json' mode='r' encoding='cp1252'>
 aux_dict = json.load(open(auxiliary_parameters, "r"))
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\modeled\cad\
 Primitives3D.py:1460: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\
 pyaedt_prj_86B\\componentLibrary\\QFP.a3dcomp.json' mode='r' encoding='cp1252'>
 aux_dict = json.load(open(auxiliary_parameters, "r"))
```

True

## Release AEDT

Release AEDT.

```
ipk.release_desktop(True, True)
```

True

**Total running time of the script:** (2 minutes 42.622 seconds)

## **Icepak: Creating blocks and assigning materials and power**

This example shows how to create different types of blocks and assign power and material to them using a \*.csv input file

### **Perform required imports**

Perform required imports including the operating system, regular expression, csv, Ansys PyAEDT and its boundary objects.

```
import os
import re
import csv
from collections import OrderedDict
import pyaedt
from pyaedt.modules.Boundary import BoundaryObject
```

### **Set AEDT version**

Set AEDT version.

```
aedt_version = "2024.1"
```

### **Set non-graphical mode**

Set non-graphical mode. You can set non\_graphical either to True or False.

```
non_graphical = False
```

### **Download and open project**

Download the project, open it, and save it to the temporary folder.

```
temp_folder = pyaedt.generate_unique_folder_name()

ipk = pyaedt.Icepak(projectname=os.path.join(temp_folder, "Icepak_CSV_Import.aedt"),
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical
)

ipk.autosave_disable()

Create the PCB as a simple block.
board = ipk.modeler.create_box([-30.48, -27.305, 0], [146.685, 71.755, 0.4064], "board_outline",
 material="FR-4_Ref")
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 7036 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 ↪py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 ↪log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Blocks creation with a CSV file

The CSV file lists the name of blocks, their type and material properties. Block types (solid, network, hollow), block name, block starting and end points, solid material, and power are listed. Hollow and network blocks do not need the material name. Network blocks must have Rjb and Rjc values. Monitor points can be created for any types of block if the last column is assigned to be 1 (0 and 1 are the only options).

The following image does not show the entire rows and data and only serves as a sample.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
block_type	name	xs	ys	zs	xd	yd	zd	matname	power	Board_side	Rjb	Rjc	Monitor_point
2 solid	BOARD_OUTLINE_1	-30.48	-27.305	0	146.685	71.755	0.4064	FR-4_Ref	0				0
3 hollow	R8	31.75	-20.32	0.4064	15.24	2.54	2.54		0.25	maxZ	1	5	0
4 network	C1	-24.511	31.496	0	5.207	3.048	-0.508		0.25	maxZ	1	5	0
5 network	C2	-24.511	26.416	0	5.207	3.048	-0.508		0.25	maxZ	1	5	0
6 network	C3	-24.511	21.336	0	5.207	3.048	-0.508		0.25	maxZ	1	5	0
7 network	C8	-24.511	16.256	0	5.207	3.048	-0.508		0.25	maxZ	1	5	0
8 network	C22	34.036	28.829	0	3.048	5.207	-0.508		0.25	maxZ	2	10	0
9 network	C23	34.036	8.509	0	3.048	5.207	-0.508		0.25	maxZ	2	10	0
10 network	C24	34.036	-11.811	0	3.048	5.207	-0.508		0.25	maxZ	2	10	0
11 network	C25	64.516	-11.811	0	3.048	5.207	-0.508		0.25	maxZ	2	10	0
12 network	C26	64.516	8.509	0	3.048	5.207	-0.508		0.25	maxZ	2	10	0
13 network	C27	64.516	28.829	0	3.048	5.207	-0.508		0.25	maxZ	2	10	0
14 network	C28	49.276	28.829	0.4064	3.048	5.207	0.508		0.25	minZ	2	10	0
15 network	C29	49.276	8.509	0.4064	3.048	5.207	0.508		0.25	minZ	2	10	0
16 network	C30	49.276	-11.811	0.4064	3.048	5.207	0.508		0.25	minZ	2	10	0
17 network	C4	-18.415	-13.97	0.4064	3.81	2.54	2.4384		0.25	minZ	1.5	15	0
18 network	C5	-18.415	-10.16	0.4064	3.81	2.54	2.4384		0.25	minZ	1.5	15	0
19 network	C6	-18.415	-6.35	0.4064	3.81	2.54	2.4384		0.25	minZ	1.5	15	0
20 network	C7	-18.415	-2.54	0.4064	3.81	2.54	2.4384		0.25	minZ	1.5	15	0
21 solid	R10	55.88	19.05	0.4064	15.24	2.54	2.54	Copper	1.25				0
22 solid	R13	54.61	-20.32	0.4064	15.24	2.54	2.54	Copper	0.75				0
23 solid	R90	31.75	39.37	0.4064	15.24	2.54	2.54	Copper	1.1				0
24 solid	U1	16.551	10.2035	0.4064	10.16	20.32	5.08	Ceramic_material	0.2				1
25 solid	U2	-16.51	10.16	0.4064	10.16	27.94	5.08	Ceramic_material	0.1				1
26 solid	U3	77.47	12.741	0.4064	10.16	22.86	5.08	Ceramic_material	0.05				1
27 solid	U15	77.47	-20.279	0.4064	10.16	22.86	5.08	Ceramic_material	0.25				1
28 solid	U4	97.79	12.741	0.4064	10.16	22.86	5.08	Ceramic_material	0.15				1
29 solid	U16	100.33	-20.279	0.4064	10.16	22.86	5.08	Ceramic_material	0.18				1
30 solid	U5	46.99	-15.24	0	7.62	10.16	-2.54	Ceramic_material	0.175				1
31 solid	U6	62.23	5.08	0.4064	7.62	10.16	2.54	Ceramic_material	0.3				1
32 solid	U7	31.75	25.4	0.4064	7.62	10.16	2.54	Ceramic_material	0.3				1
33 solid	U8	31.75	5.08	0.4064	7.62	10.16	2.54	Ceramic_material	0.3				1
34 solid	U9	62.23	25.4	0.4064	7.62	10.16	2.54	Ceramic_material	0.3				1
35 solid	U11	46.99	25.4	0	7.62	10.16	-2.54	Ceramic_material	0.3				1
36 solid	U12	46.99	5.08	0	7.62	10.16	-2.54	Ceramic_material	0.3				1

In this step the code will loop over the csv file lines and creates the blocks. It will create solid blocks and assign BCs. Every row of the csv has information of a particular block.

```
filename = pyaedt.downloads.download_file('icepak', 'blocks-list.csv', destination=temp_
↪folder)

with open(filename, 'r') as csv_file:
 csv_reader = csv.DictReader(csv_file)
 for row in csv_reader:
 origin = [float(row["xs"]), float(row["ys"]), float(row["zs"])] # block_
↪starting point
 dimensions = [float(row["xd"]), float(row["yd"]), float(row["zd"])] # block_
↪lengths in 3 dimensions
 block_name = row["name"] # block name
```

(continues on next page)

(continued from previous page)

```

Define material name
if row["matname"]:
 material_name = row["matname"]
else:
 material_name = "copper"

creates the block with the given name, coordinates, material, and type
block = ipk.modeler.create_box(origin, dimensions, name=block_name,
material=material_name)

Assign boundary conditions
if row["block_type"] == "solid":
 ipk.assign_solid_block(block_name, row["power"] + "W", block_name)
elif row["block_type"] == "network":
 ipk.create_two_resistor_network_block(block_name,
 board.name,
 row["power"] + "W",
 row["Rjb"],
 row["Rjc"])
else:
 ipk.modeler[block.name].solve_inside = False
 ipk.assign_hollow_block(block_name, assignment_type="Total Power",
assignment_value=row["power"] + "W",
 boundary_name=block_name)

Create temperature monitor points if assigned value is 1 in the last column of
the csv file
if row["Monitor_point"] == '1':
 ipk.monitor.assign_point_monitor_in_object(
 row["name"],
 monitor_quantity="Temperature",
 monitor_name=row["name"])

```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\modeler\modeler3d.
→py:56: DeprecationWarning: The property `primitives` is deprecated.
Use `app.modeler` directly to instantiate primitives methods.
warnings.warn(mess, DeprecationWarning)
```

## Release AEDT

Release AEDT.

```
ipk.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (1 minutes 6.457 seconds)

## Icepak: Importing a PCB and its components via IDF and EDB

This example shows how to import a PCB and its components using IDF files (.ldb/.bdf). The .emn/.emp combination can also be used in a similar way.

### Perform required imports

Perform required imports including the operating system, Ansys PyAEDT packages.

```
Generic Python packages

import os

PyAEDT Packages
import pyaedt
from pyaedt import Icepak
from pyaedt import Desktop
from pyaedt import Hfss3dLayout
from pyaedt.modules.Boundary import BoundaryObject
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set non\_graphical either to True or False.

```
non_graphical = False
```

### Download and open project

Download the project, open it, and save it to the temporary folder.

```
temp_folder = pyaedt.generate_unique_folder_name()

ipk = pyaedt.Icepak(projectname=os.path.join(temp_folder, "Icepak_ECAD_Import.aedt"),
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical
)

ipk.autosave_disable() # Saves the
 ↵project
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 8732 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

True

## Import the IDF files

Sample \*.bdf and \*.ldf files are presented here.

```
.HEADER
BOARD_FILE 3.0 "apd 15.2" 2006/08/14.14:14:09 1
A1.brd THOU
.END_HEADER
.BOARD_OUTLINE ECAD
| 61.60
0 6404.98 -3676.99 0.000
0 6404.98 197.00 0.000
0 -195.00 197.00 0.000
0 -195.00 -4001.79 0.000
0 395.55 -4001.79 0.000
0 395.55 -3706.99 0.000
0 425.55 -3676.99 -90.000
0 1087.99 -3676.99 0.000
0 1117.99 -3706.99 -90.000
0 1117.99 -4001.79 0.000
0 1432.95 -4001.79 0.000
0 1432.95 -3748.84 0.000
0 1576.65 -3748.84 -180.000
0 1576.65 -4159.27 0.000
0 1596.33 -4178.96 0.000
0 1997.91 -4178.96 0.000
0 2017.59 -4159.27 0.000
0 2017.59 -3885.65 0.000
0 2092.38 -3885.65 -180.000
0 2092.38 -4159.27 0.000
0 2112.07 -4178.96 0.000
0 4875.84 -4178.96 0.000
0 4895.53 -4159.27 0.000
0 4895.53 -3879.55 0.000
0 5003.80 -3879.55 -180.000
0 5003.80 -4043.13 0.000
0 5023.48 -4062.82 0.000
0 5448.68 -4062.82 0.000
0 5503.80 -3967.35 0.000
0 5503.80 -3865.97 0.000
0 5348.29 -3865.97 0.000
0 5279.39 -3797.07 -90.000
^
```

```

.HEADER
LIBRARY_FILE 3.0 "apd 15.2" 2006/08/14.14:14:09 1
.END_HEADER
.ELECTRICAL
BRKT_ADB15_BDIN9_CDVI_DB15S DEVICE? THOU 1.00
0 -199.00 -454.00 0.000
0 -199.00 -147.00 0.000
0 150.00 -147.00 0.000
0 150.00 -454.00 0.000
0 -199.00 -454.00 0.000
.END_ELECTRICAL
.MECHANICAL
LOGO_VCCI-225X300 "" THOU 1.00
0 168.00 -115.00 0.000
0 168.00 111.00 12.166
0 -153.00 108.00 19.173
0 -153.00 -115.00 20.503
0 168.00 -115.00 18.523
.END_MECHANICAL
.MECHANICAL
LOGO_UL_ITE_204896_600 "" THOU 150.00
0 -238.27 -295.00 0.000
0 -238.27 169.96 0.000
0 310.25 169.96 0.000
0 310.25 -295.00 0.000
0 -238.27 -295.00 0.000
.END_MECHANICAL
.MECHANICAL
LOGO_ICES-003 "" THOU 1.00
0 -500.00 -85.00 0.000
0 -500.00 65.00 0.000
0 550.00 65.00 0.000
0 550.00 -85.00 0.000
0 -500.00 -85.00 0.000
.END_MECHANICAL
.MECHANICAL
MEC_HEX_JACK_SCREW "" THOU 150.00
.END_MECHANICAL
.ELECTRICAL
0402 R_0402-1K,1%,1/16W,N/A-1%,1/16A THOU 23.60
0 -44.25 -19.65 0.000
0 -44.25 19.65 0.000
0 44.25 19.65 0.000
0 44.25 -19.65 0.000
0 -44.25 -19.65 0.000
.END_ELECTRICAL
.ELECTRICAL

```

Imports the idf files with several filtering options including caps, resistors, inductors, power, size, There are also options for the PCB creation (number of layers, copper percentages, layer sizes). In this example, the default values are used for the PCB. The imported PCB here will be deleted later and replaced by a PCB that has the trace information for higher accuracy.

```

def_path = pyaedt.downloads.download_file('icepak/Icepak_ECAD_Import/A1_uprev.aedb','edb.
˓→def',temp_folder)
board_path = pyaedt.downloads.download_file('icepak/Icepak_ECAD_Import/','A1.bdf',temp_
˓→folder)
library_path = pyaedt.downloads.download_file('icepak/Icepak_ECAD_Import/','A1.ldf',temp_
˓→folder)

ipk.import_idf(board_path, library_path=None, control_path=None,
 filter_cap=False, filter_ind=False, filter_res=False,
 filter_height_under=None, filter_height_exclude_2d=False,
 power_under=None, create_filtered_as_non_model=False,

```

(continues on next page)

(continued from previous page)

```
high_surface_thick='0.07mm', low_surface_thick='0.07mm',
internal_thick='0.07mm', internal_layer_number=2,
high_surface_coverage=30, low_surface_coverage=30,
internal_layer_coverage=30, trace_material='Cu-Pure',
substrate_material='FR-4', create_board=True,
model_board_as_rect=False, model_device_as_rect=True,
cutoff_height='5mm', component_lib='')
```

True

**Fit to scale, save the project**

```
ipk.modeler.fit_all() # scales to fit all objects in AEDT
ipk.save_project() # saves the project
```

True

**Add an HFSS 3D Layout design with the layout information of the PCB**

```
Layout_name = 'A1_uprev' # 3D layout name available for import, the extension
 ↵ of .aedb should not be listed here

hfss3dL0 = Hfss3dLayout('Icepak_ECAD_Import', 'PCB_temp') # adding a dummy HFSS 3D
 ↵ layout to the current project

#edb_full_path = os.path.join(os.getcwd(), Layout_name+'.aedb\edb.def') # path to the
 ↵ EDB file
hfss3dL0.import_edb(def_path) # importing the EDB file
hfss3dL0.save_project() # save the new
 ↵ project so files are stored in the path

ipk.delete_design(name='PCB_temp', fallback_design=None) # deleting the
 ↵ dummy layout from the original project

This part creates a 3D component PCB in Icepak from the imported EDB file
1 watt is assigned to the PCB as power input

component_name = "PCB_ECAD"

odb_path = os.path.join(temp_folder, 'icepak/Icepak_ECAD_Import/'+Layout_name+'.aedt')
ipk.create_pcb_from_3dlayout(
 component_name, odb_path, Layout_name, resolution=2, extenttype="Polygon",
 ↵ outlinepolygon='poly_0',
 custom_x_resolution=None, custom_y_resolution=None, power_in=1)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\icepak.py:2560:_
DeprecationWarning: ``extenttype`` was deprecated in 0.6.43. Use ``extent_type``_
instead.
```

(continues on next page)

(continued from previous page)

```
warnings.warn()
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\icepak.py:2567:
 ↳DeprecationWarning: ``outlinepolygon`` was deprecated in 0.6.43. Use ``outline_
 ↳polygon`` instead.
 warnings.warn(

<pyaedt.modules.Boundary.NativeComponentPCB object at 0x00000258BE298B20>
```

## Delete PCB objects

Delete the PCB object from IDF import.

```
ipk.modeler.delete_objects_containing("IDF_BoardOutline", False)
```

```
True
```

## Compute power budget

```
Creates a setup to be able to calculate the power
ipk.create_setup("setup1")

power_budget, total = ipk.post.power_budget("W")
print(total)
```

```
7.0
```

## Release AEDT

Release AEDT.

```
ipk.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (2 minutes 17.153 seconds)

## Icepak: graphic card thermal analysis

This example shows how you can use PyAEDT to create a graphic card setup in Icepak and postprocess results. The example file is an Icepak project with a model that is already created and has materials assigned.

## Perform required imports

Perform required imports.

```
import os
import pyaedt
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Download and open project

Download the project, open it, and save it to the temporary folder.

```
temp_folder = pyaedt.generate_unique_folder_name()
project_temp_name = pyaedt.downloads.download_icepak(temp_folder)

ipk = pyaedt.Icepak(projectname=project_temp_name,
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical
)

ipk.autosave_disable()
```

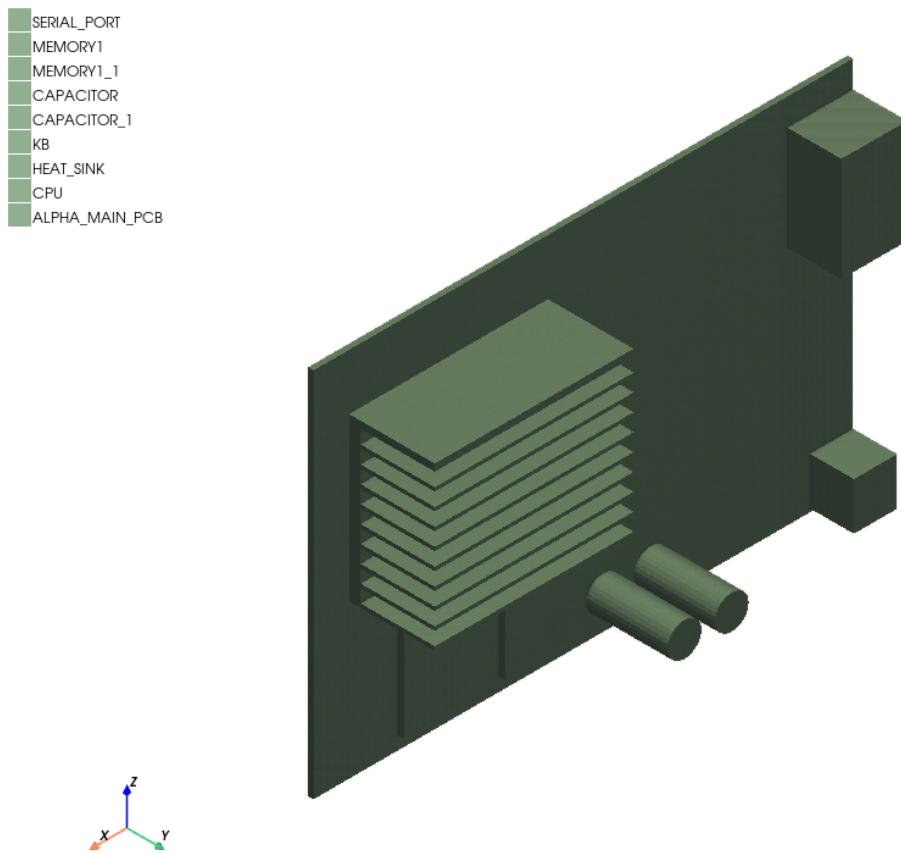
```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 1656 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

```
True
```

## Plot model

Plot the model.

```
ipk.plot(show=False, export_path=os.path.join(temp_folder, "Graphics_card.jpg"), plot_air_objects=False)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258AB146770>
```

## Create source blocks

Create source blocks on the CPU and memories.

```
ipk.create_source_block("CPU", "25W")
ipk.create_source_block(["MEMORY1", "MEMORY1_1"], "5W")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE29BC10>
```

## Assign openings and grille

Assign openings and a grille.

```
region = ipk.modeler["Region"]
ipk.assign_openings(air_faces=region.bottom_face_x.id)
ipk.assign_grille(air_faces=region.top_face_x.id, free_area_ratio=0.8)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258AB144FD0>
```

## Assign mesh regions

Assign a mesh region to the heat sink and CPU.

```
mesh_region = ipk.mesh.assign_mesh_region(assignment=["HEAT_SINK", "CPU"])
mesh_region.UserSpecifiedSettings = True
mesh_region.MaxElementSizeX = "3.35mm"
mesh_region.MaxElementSizeY = "1.75mm"
mesh_region.MaxElementSizeZ = "2.65mm"
mesh_region.MaxLevels = "2"
mesh_region.update()
```

```
True
```

## Assign point monitor

Assign a point monitor and set it up.

```
ipk.assign_point_monitor(point_position=["-35mm", "3.6mm", "-86mm"], monitor_name=
 ↪"TemperatureMonitor1")
ipk.assign_point_monitor(point_position=["80mm", "14.243mm", "-55mm"], monitor_type=
 ↪"Speed")
setup1 = ipk.create_setup()
setup1.props["Flow Regime"] = "Turbulent"
setup1.props["Convergence Criteria - Max Iterations"] = 5
setup1.props["Linear Solver Type - Pressure"] = "flex"
setup1.props["Linear Solver Type - Temperature"] = "flex"
ipk.save_project()
```

```
True
```

## Solve project and postprocess

Solve the project and plot temperatures.

```
quantity_name = "SurfTemperature"
surflist = [i.id for i in ipk.modeler["CPU"].faces]
surflist += [i.id for i in ipk.modeler["MEMORY1"].faces]
surflist += [i.id for i in ipk.modeler["MEMORY1_1"].faces]
surflist += [i.id for i in ipk.modeler["ALPHA_MAIN_PCB"].faces]

plot5 = ipk.post.create_fieldplot_surface(surflist, "SurfTemperature")

ipk.analyze()
```

```
True
```

## Release AEDT

Release AEDT.

```
ipk.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (2 minutes 45.520 seconds)

## Icepak: setup from Sherlock inputs

This example shows how you can create an Icepak project from Sherlock files (STEP and CSV) and an Aedb board.

### Perform required imports

Perform required imports and set paths.

```
import time
import os
import pyaedt
import datetime

Set paths
project_folder = pyaedt.generate_unique_folder_name()
input_dir = pyaedt.downloads.download_sherlock(destination=project_folder)
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` value either to `True` or `False`.

```
non_graphical = False
```

## Define variables

Define input variables. The following code creates all input variables that are needed to run this example.

```
material_name = "MaterialExport.csv"
component_properties = "TutorialBoardPartsList.csv"
component_step = "TutorialBoard.stp"
aedt_odb_project = "SherlockTutorial.aedt"
aedt_odb_design_name = "PCB"
stackup_thickness = 2.11836
outline_polygon_name = "poly_14188"
```

## Launch AEDT

Launch AEDT 2023 R2 in graphical mode.

```
d = pyaedt.launch_desktop(specified_version=aedt_version, non_graphical=non_graphical,
 new_desktop_session=True)

start = time.time()
material_list = os.path.join(input_dir, material_name)
component_list = os.path.join(input_dir, component_properties)
validation = os.path.join(project_folder, "validation.log")
file_path = os.path.join(input_dir, component_step)
project_name = os.path.join(project_folder, component_step[:-3] + "aedt")
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 6628 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create Icepak project

Create an Icepak project.

```
ipk = pyaedt.Icepak(project_name)
```

## Delete region to speed up import

Delete the region and disable autosave to speed up the import.

```
d.disable_autosave()
ipk.modeler.delete("Region")
component_name = "from_ODB"
```

## Import PCB from AEDB file

Import a PCB from an AEDB file.

```
odb_path = os.path.join(input_dir, aedt_odb_project)
ipk.create_pcb_from_3dlayout(component_name=component_name, project_name=odb_path,
 ↪design_name=aedt_odb_design_name,
 ↪extenttype="Polygon")
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\icepak.py:2560:
 ↪DeprecationWarning: ``extenttype`` was deprecated in 0.6.43. Use ``extent_type``
 ↪instead.
 warnings.warn(

<pyaedt.modules.Boundary.NativeComponentPCB object at 0x00000258AB1442B0>
```

## Create offset coordinate system

Create an offset coordinate system to match ODB++ with the Sherlock STEP file.

```
ipk.modeler.create_coordinate_system(origin=[0, 0, stackup_thickness / 2], mode="view",
 ↪view="XY")
```

```
<pyaedt.modeler.cad.Modeler.CoordinateSystem object at 0x00000258AB145DE0>
```

## Import CAD file

Import a CAD file.

```
ipk.modeler.import_3d_cad(file_path, refresh_all_ids=False)
```

```
True
```

## Save CAD file

Save the CAD file and refresh the properties from the parsing of the AEDT file.

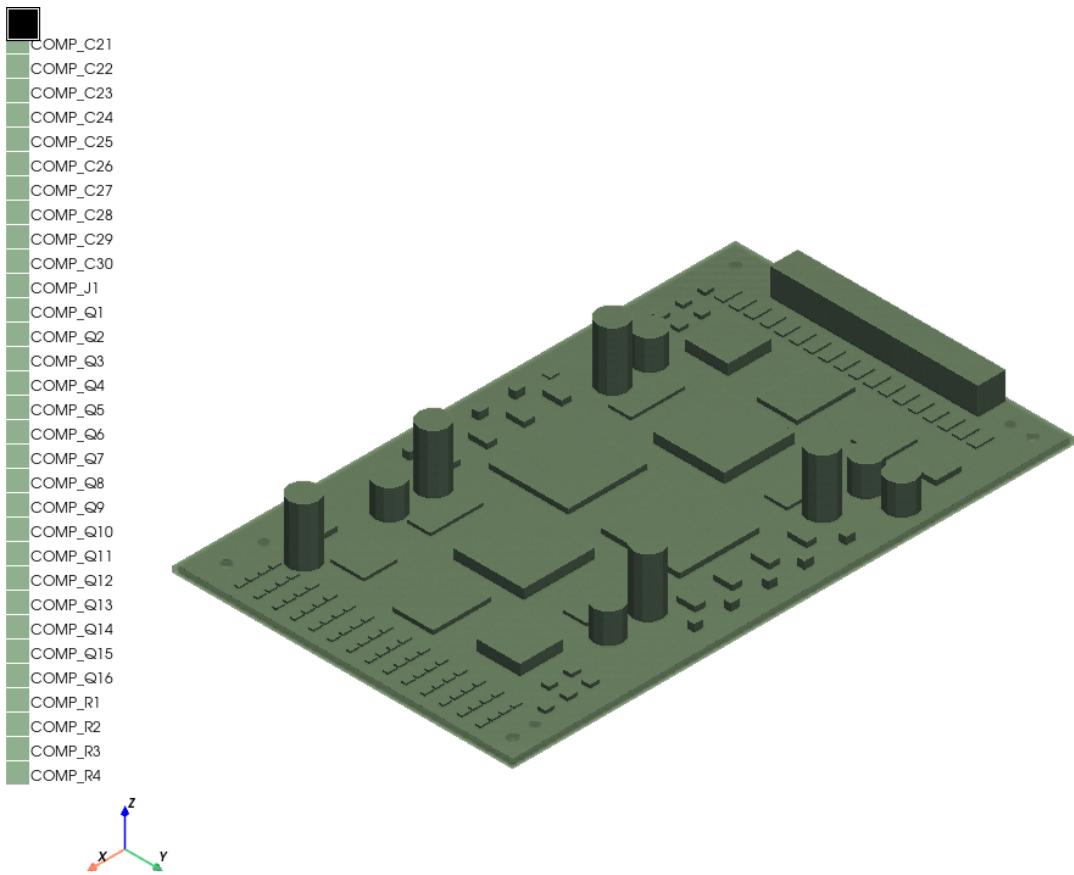
```
ipk.save_project(refresh_obj_ids_after_save=True)
```

```
True
```

## Plot model

Plot the model.

```
ipk.plot(show=False, export_path=os.path.join(project_folder, "Sherlock_Example.jpg"),
 plot_air_objects=False)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BE2FD6F0>
```

## Delete PCB objects

Delete the PCB objects.

```
ipk.modeler.delete_objects_containing("pcb", False)
```

```
True
```

## Create region

Create an air region.

```
ipk.modeler.create_air_region(*[20, 20, 300, 20, 20, 300])
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258BE2F3C40>
```

## Assign materials

Assign materials from Sherlock file.

```
ipk.assignmaterial_from_sherlock_files(component_list, material_list)
```

```
True
```

## Delete objects with no material assignments

Delete objects with no materials assignments.

```
no_material_objs = ipk.modeler.get_objects_by_material("")
ipk.modeler.delete(no_material_objs)
ipk.save_project()
```

```
True
```

## Assign power to component blocks

Assign power to component blocks.

```
all_objects = ipk.modeler.object_names
```

## Assign power blocks

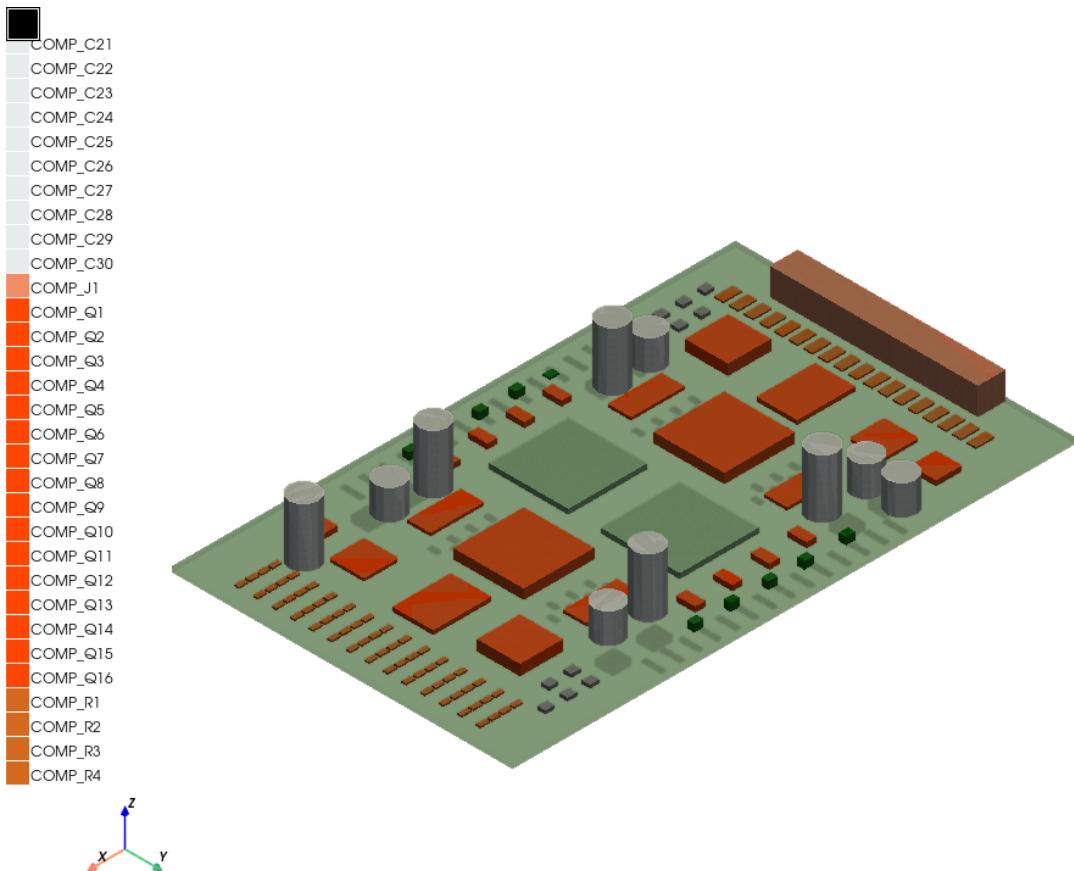
Assign power blocks from the Sherlock file.

```
total_power = ipk.assign_block_from_sherlock_file(csv_name=component_list)
```

## Plot model

Plot the model again now that materials are assigned.

```
ipk.plot(show=False, export_path=os.path.join(project_folder, "Sherlock_Example.jpg"),
plot_air_objects=False)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BE2F3670>
```

## Set up boundaries

Set up boundaries.

```
Mesh settings that is tailored for PCB
Max iterations is set to 20 for quick demonstration, please increase to at least 100
for better accuracy.

ipk.globalMeshSettings(3, gap_min_elements='1', no0grids=True, MLM_en=True,
 MLM_Type='2D', edge_min_elements='2', object='Region')

setup1 = ipk.create_setup()
setup1.props["Solution Initialization - Y Velocity"] = "1m_per_sec"
setup1.props["Radiation Model"] = "Discrete Ordinates Model"
setup1.props["Include Gravity"] = True
setup1.props["Secondary Gradient"] = True
setup1.props["Convergence Criteria - Max Iterations"] = 10
ipk.assign_openings(ipk.modeler.get_object_faces("Region"))
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C47B77C0>
```

### Create point monitor

```
point1 = ipk.assign_point_monitor(ipk.modeler["COMP_U10"].top_face_z.center, monitor_
 ↵name="Point1")
ipk.modeler.set_working_coordinate_system("Global")
line = ipk.modeler.create_polyline(
 [ipk.modeler["COMP_U10"].top_face_z.vertices[0].position, ipk.modeler["COMP_U10"].
 ↵top_face_z.vertices[2].position],
 non_model=True)
ipk.post.create_report(expressions="Point1.Temperature", primary_sweep_variable="X")
```

```
<pyaedt.modules.report_templates.Standard object at 0x00000258BE2FF790>
```

### Check for intersections

Check for intersections using validation and fix them by assigning priorities.

```
ipk.assign_priority_on_intersections()
```

```
True
```

### Compute power budget

```
power_budget, total = ipk.post.power_budget("W")
print(total)
```

```
98.50000000000057
```

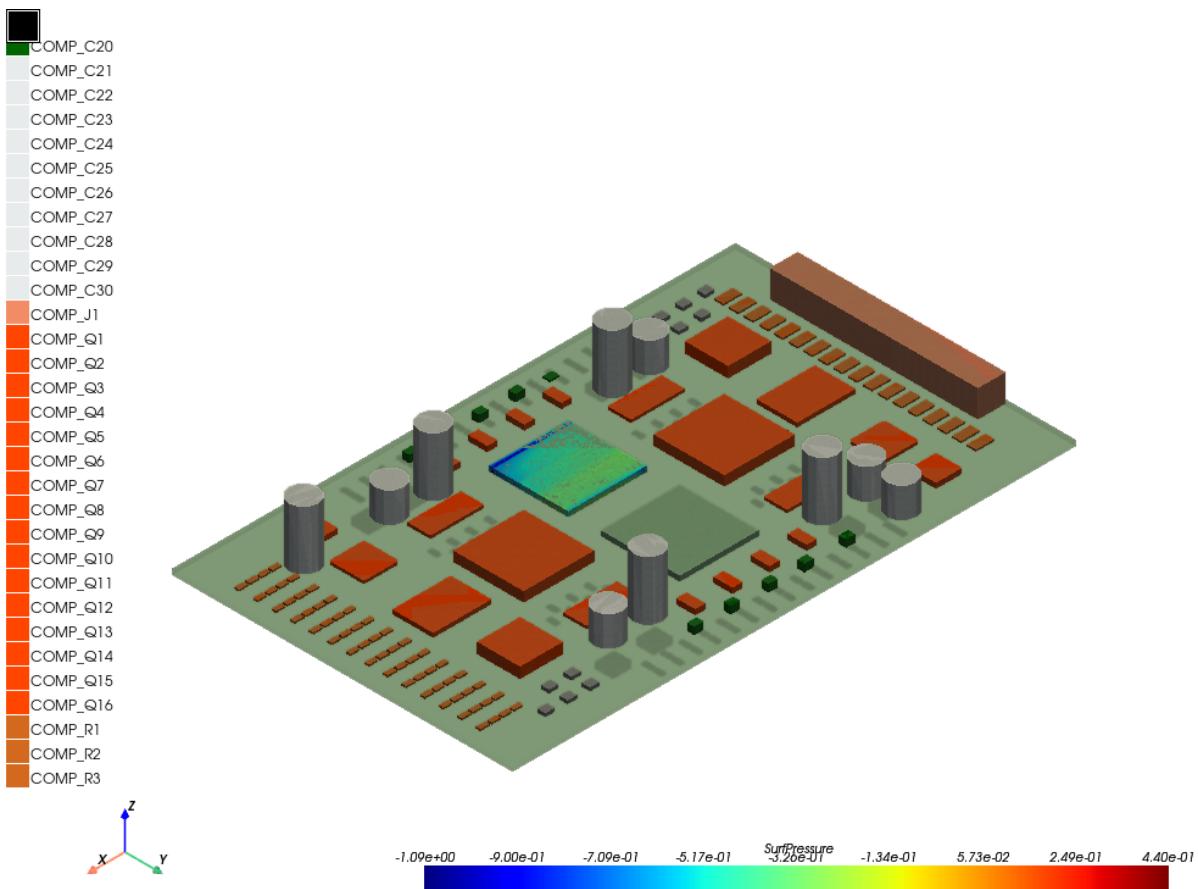
### Analyze the model

```
ipk.analyze(cores=4, tasks=4)
ipk.save_project()
```

```
True
```

## Get solution data and plots

```
plot1 = ipk.post.create_fieldplot_surface(ipk.modeler["COMP_U10"].faces, "SurfTemperature")
ipk.post.plot_field("SurfPressure", ipk.modeler["COMP_U10"].faces, show=False, export_
path=ipk.working_directory)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258C5696C20>
```

## Save project and release AEDT

Save the project and release AEDT.

```
ipk.save_project()

end = time.time() - start
print("Elapsed time: {}".format(datetime.timedelta(seconds=end)))
print("Project Saved in {}".format(ipk.project_file))
ipk.release_desktop()
```

```
Elapsed time: 0:20:07.893967
Project Saved in D:/Temp/pyaedt_prj_ZII/TutorialBoard.aedt

True
```

**Total running time of the script:** (20 minutes 23.493 seconds)

### 3.12.8 2D Extractor and Q3D Extractor examples

These examples use PyAEDT to show some end-to-end workflows for 2D Extractor and Q3D Extractor. This includes model generation, setup, and thermal postprocessing.

#### Q2D: Cable parameter identification

This example shows how you can use PyAEDT to perform these tasks:

- Create a Q2D design using the Modeler primitives and importing part of the geometry.
- Set up the entire simulation.
- Link the solution to a Simplorer design.

For cable information, see [4 Core Armoured Power Cable](#)

#### Perform required imports

```
import pyaedt
import math
```

#### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

#### Initialize core strand dimensions and positions

Initialize cable sizing - radii in mm.

```
c_strand_radius = 2.575
cable_n_cores = 4
core_n_strands = 6
core_xlpe_ins_thickness = 0.5
core_xy_coord = math.ceil(3 * c_strand_radius + 2 * core_xlpe_ins_thickness)
```

## Initialize filling and sheath dimensions

Initialize radii of further structures incrementally adding thicknesses.

```
filling_radius = 1.4142 * (core_xy_coord + 3 * c_strand_radius + core_xlpe_ins_thickness
+ 0.5)
inner_sheath_radius = filling_radius + 0.75
armour_thickness = 3
armour_radius = inner_sheath_radius + armour_thickness
outer_sheath_radius = armour_radius + 2
```

## Initialize armature strand dimensions

Initialize radii.

```
armour_centre_pos = inner_sheath_radius + armour_thickness / 2.0
arm_strand_rad = armour_thickness / 2.0 - 0.2
n_arm_strands = 30
```

## Initialize dictionaries

Initialize dictionaries that contain all the definitions for the design variables and output variables.

```
core_params = {
 "n_cores": str(cable_n_cores),
 "n_strands_core": str(core_n_strands),
 "c_strand_radius": str(c_strand_radius) + 'mm',
 "c_strand_xy_coord": str(core_xy_coord) + 'mm'
}
outer_params = {
 "filling_radius": str(filling_radius) + 'mm',
 "inner_sheath_radius": str(inner_sheath_radius) + 'mm',
 "armour_radius": str(armour_radius) + 'mm',
 "outer_sheath_radius": str(outer_sheath_radius) + 'mm'
}
armour_params = {
 "armour_centre_pos": str(armour_centre_pos) + 'mm',
 "arm_strand_rad": str(arm_strand_rad) + 'mm',
 "n_arm_strands": str(n_arm_strands)
}
```

## Initialize Q2D

Initialize Q2D, providing the version, path to the project, and the design name and type.

```
project_name = 'Q2D_ArmouredCableExample'
q2d_design_name = '2D_Extractor_Cable'
setup_name = "MySetupAuto"
sweep_name = "sweep1"
tb_design_name = 'CableSystem'
q2d = pyaedt.Q2d(projectname=project_name, designname=q2d_design_name, specified_
 ↴version=aedt_version)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: ↴
 subprocess 9612 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 ↴py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 ↴log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Define variables from dictionaries

Define design variables from the created dictionaries.

```
for k, v in core_params.items():
 q2d[k] = v
for k, v in outer_params.items():
 q2d[k] = v
for k, v in armour_params.items():
 q2d[k] = v
```

## Create object to access 2D modeler

Create the mod2D object to access the 2D modeler easily.

```
mod2D = q2d.modeler
mod2D.delete()
mod2D.model_units = "mm"
```

## Initialize required material properties

Cable insulators require the definition of specific materials since they are not included in the Sys Library. Plastic, PE (cross-linked, wire, and cable grade)

```
mat_pe_cable_grade = q2d.materials.add_material("plastic_pe_cable_grade")
mat_pe_cable_grade.conductivity = "1.40573e-16"
mat_pe_cable_grade.permittivity = "2.09762"
mat_pe_cable_grade.dielectric_loss_tangent = "0.000264575"
mat_pe_cable_grade.update()
```

(continues on next page)

(continued from previous page)

```
Plastic, PP (10% carbon fiber)
mat_pp = q2d.materials.add_material("plastic_pp_carbon_fiber")
mat_pp.conductivity = "0.0003161"
mat_pp.update()
```

True

### Create geometry for core strands, filling, and XLPE insulation

```
mod2D.create_coordinate_system(['c_strand_xy_coord', 'c_strand_xy_coord', '0mm'], name=
 ↪'CS_c_strand_1')
mod2D.set_working_coordinate_system('CS_c_strand_1')
c1_id = mod2D.create_circle(['0mm', '0mm', '0mm'], 'c_strand_radius', name='c_strand_1', ↪
 ↪material='copper')
c2_id = c1_id.duplicate_along_line(vector=['0mm', '2.0*c_strand_radius', '0mm'], ↪
 ↪ncopies=2)
mod2D.duplicate_around_axis(c2_id, axis="Z", angle=360 / core_n_strands, clones=6)
c_unite_name = mod2D.unite(q2d.get_all_conductors_names())

fill_id = mod2D.create_circle(['0mm', '0mm', '0mm'], '3*c_strand_radius', name='c_strand_ ↪
 ↪fill',
 material='plastic_pp_carbon_fiber')
fill_id.color = (255, 255, 0)
xlpe_id = mod2D.create_circle(['0mm', '0mm', '0mm'], '3*c_strand_radius+' + str(core_ ↪
 ↪xlpe_ins_thickness) + 'mm',
 name='c_strand_xlpe',
 material='plastic_pe_cable_grade')
xlpe_id.color = (0, 128, 128)

mod2D.set_working_coordinate_system('Global')
all_obj_names = q2d.get_all_conductors_names() + q2d.get_all_dielectrics_names()
mod2D.duplicate_around_axis(all_obj_names, axis="Z", angle=360 / cable_n_cores, clones=4)
cond_names = q2d.get_all_conductors_names()
```

### Create geometry for filling object

```
filling_id = mod2D.create_circle(['0mm', '0mm', '0mm'], 'filling_radius', name='Filling', ↪
 material='plastic_pp_carbon_fiber')
filling_id.color = (255, 255, 180)
```

### Create geometry for inner sheath object

```
inner_sheath_id = mod2D.create_circle(['0mm', '0mm', '0mm'], 'inner_sheath_radius', name=
 ↪'InnerSheath',
 material='PVC plastic')
inner_sheath_id.color = (0, 0, 0)
```

### Create geometry for armature fill

```
arm_fill_id = mod2D.create_circle(['0mm', '0mm', '0mm'], 'armour_radius', name=
 ↪'ArmourFilling',
 material='plastic_pp_carbon_fiber')
arm_fill_id.color = (255, 255, 255)
```

### Create geometry for outer sheath

```
outer_sheath_id = mod2D.create_circle(['0mm', '0mm', '0mm'], 'outer_sheath_radius', name=
 ↪'OuterSheath',
 material='PVC plastic')
outer_sheath_id.color = (0, 0, 0)
```

### Create geometry for armature steel strands

```
arm_strand_1_id = mod2D.create_circle(['0mm', 'armour_centre_pos', '0mm'], '1.1mm', name=
 ↪'arm_strand_1',
 material='steel_stainless')
arm_strand_1_id.color = (128, 128, 64)
arm_strand_1_id.duplicate_around_axis('Z', '360deg/n_arm_strands', clones='n_arm_strands
 ↪')
arm_strand_names = mod2D.get_objects_w_string('arm_strand')
```

## Create region

```
region = q2d.modeler.create_region([500, 500, 500, 500])
region.material_name = "vacuum"
```

## Assign conductors and reference ground

```
obj = [q2d.modeler.get_object_from_name(i) for i in cond_names]
[q2d.assign_single_conductor(assignment=i, name='C1' + str(obj.index(i) + 1), conductor_
 ↴type='SignalLine') for i
 ↴in obj]
obj = [q2d.modeler.get_object_from_name(i) for i in arm_strand_names]
q2d.assign_single_conductor(assignment=obj, name="gnd", conductor_type="ReferenceGround")
mod2D.fit_all()
```

## Assign design settings

```
lumped_length = "100m"
q2d_des_settings = q2d.design_settings
q2d_des_settings['LumpedLength'] = lumped_length
```

## Insert setup and frequency sweep

```
q2d_setup = q2d.create_setup(name=setup_name)
q2d_sweep = q2d_setup.add_sweep(name=sweep_name)
```

## Analyze setup

```
q2d.analyze(setup_name=setup_name)
```

## Add a Simplorer/Twin Builder design and the Q3D dynamic component

```
tb = pyaedt.TwinBuilder(designname=tb_design_name)
```

## Add a Q3D dynamic component

```
tb.add_q3d_dynamic_component(project_name, q2d_design_name, setup_name, sweep_name,
 ↴coupling_matrix_name="Original",
 ↴model_depth=lumped_length)
```

```
<pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object at 0x00000258BE2FD0C0>
```

## Save project and release desktop

```
tb.save_project()
tb.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (1 minutes 0.943 seconds)

## 2D Extractor: CPWG analysis

This example shows how you can use PyAEDT to create a CPWG (coplanar waveguide with ground) design in 2D Extractor and run a simulation.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

### Launch AEDT and 2D Extractor

Launch AEDT 2023 R2 in graphical mode and launch 2D Extractor. This example uses SI units.

```
q = pyaedt.Q2d(specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True,
 projectname=pyaedt.generate_unique_name("pyaedt_q2d_example"),
 designname="coplanar_waveguide")
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 11720 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
```

(continues on next page)

(continued from previous page)

```
→py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Define variables

Define variables.

```
e_factor = "e_factor"
sig_bot_w = "sig_bot_w"
co_gnd_w = "gnd_w"
clearance = "clearance"
cond_h = "cond_h"
d_h = "d_h"
sm_h = "sm_h"

for var_name, var_value in {
 "sig_bot_w": "150um",
 "e_factor": "2",
 "gnd_w": "500um",
 "clearance": "150um",
 "cond_h": "50um",
 "d_h": "150um",
 "sm_h": "20um",
}.items():
 q[var_name] = var_value

delta_w_half = "({0}/{1})".format(cond_h, e_factor)
sig_top_w = "({1}-{0}*2)".format(delta_w_half, sig_bot_w)
co_gnd_top_w = "({1}-{0}*2)".format(delta_w_half, co_gnd_w)
model_w = "{0}*2+{1}*2+{2}".format(co_gnd_w, clearance, sig_bot_w)
```

## Create primitives

Create primitives and define the layer heights.

```
layer_1_lh = 0
layer_1_uh = cond_h
layer_2_lh = layer_1_uh + "+" + d_h
layer_2_uh = layer_2_lh + "+" + cond_h
```

## Create signal

Create a signal.

```
base_line_obj = q.modeler.create_polyline(points=[[0, layer_2_lh, 0], [sig_bot_w, layer_
 ↪_2_lh, 0]], name="signal")
top_line_obj = q.modeler.create_polyline(points=[[0, layer_2_uh, 0], [sig_top_w, layer_2_
 ↪_uh, 0]])
q.modeler.move(assignment=[top_line_obj], vector=[delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])
q.modeler.move(assignment=[base_line_obj], vector=["{}+{}".format(co_gnd_w, clearance),_
 ↪0, 0])
```

True

## Create coplanar ground

Create a coplanar ground.

```
base_line_obj = q.modeler.create_polyline(points=[[0, layer_2_lh, 0], [co_gnd_w, layer_2_
 ↪_lh, 0]], name="co_gnd_left")
top_line_obj = q.modeler.create_polyline(points=[[0, layer_2_uh, 0], [co_gnd_top_w,_
 ↪layer_2_uh, 0]])
q.modeler.move(assignment=[top_line_obj], vector=[delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])

base_line_obj = q.modeler.create_polyline(points=[[0, layer_2_lh, 0], [co_gnd_w, layer_2_
 ↪_lh, 0]], name="co_gnd_right")
top_line_obj = q.modeler.create_polyline(points=[[0, layer_2_uh, 0], [co_gnd_top_w,_
 ↪layer_2_uh, 0]])
q.modeler.move(assignment=[top_line_obj], vector=[delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])
q.modeler.move(assignment=[base_line_obj], vector=["{}+{}*2+{}".format(co_gnd_w,_
 ↪clearance, sig_bot_w), 0, 0])
```

True

## Create reference ground plane

Create a reference ground plane.

```
q.modeler.create_rectangle(origin=[0, layer_1_lh, 0], sizes=[model_w, cond_h], name="ref_
 ↪gnd")
```

<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258AAEEB490>

## Create dielectric

Create a dielectric.

```
q.modeler.create_rectangle(
 origin=[0, layer_1_uh, 0], sizes=[model_w, d_h], name="Dielectric", material="FR4_
 ↵epoxy"
)
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258AAEE9360>
```

## Create conformal coating

Create a conformal coating.

```
sm_obj_list = []
ids = [1, 2, 3]
if aedt_version >= "2023.1":
 ids = [0, 1, 2]

for obj_name in ["signal", "co_gnd_left", "co_gnd_right"]:
 obj = q.modeler.get_object_from_name(obj_name)
 e_obj_list = []
 for i in ids:
 e_obj = q.modeler.create_object_from_edge(obj.edges[i])
 e_obj_list.append(e_obj)
 e_obj_1 = e_obj_list[0]
 q.modeler.unite(e_obj_list)
 new_obj = q.modeler.sweep_along_vector(e_obj_1.id, [0, sm_h, 0])
 sm_obj_list.append(new_obj)

new_obj = q.modeler.create_rectangle(origin=[co_gnd_w, layer_2_lh, 0], sizes=[clearance,
 ↵sm_h])
sm_obj_list.append(new_obj)

new_obj = q.modeler.create_rectangle(origin=[co_gnd_w, layer_2_lh, 0], sizes=[clearance,
 ↵sm_h])
q.modeler.move([new_obj], [sig_bot_w + "+" + clearance, 0, 0])
sm_obj_list.append(new_obj)

sm_obj = sm_obj_list[0]
q.modeler.unite(sm_obj_list)
sm_obj.material_name = "SolderMask"
sm_obj.color = (0, 150, 100)
sm_obj.name = "solder_mask"
```

## Assign conductor

Assign a conductor to the signal.

```
obj = q.modeler.get_object_from_name("signal")
q.assign_single_conductor(assignment=[obj], name=obj.name, conductor_type="SignalLine",
 solve_option="SolveOnBoundary",
 units="mm")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE27DA50>
```

## Create reference ground

Create a reference ground.

```
obj = [q.modeler.get_object_from_name(i) for i in ["co_gnd_left", "co_gnd_right", "ref_gnd"]]
q.assign_single_conductor(assignment=obj, name="gnd", conductor_type="ReferenceGround",
 solve_option="SolveOnBoundary",
 units="mm")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE27EDD0>
```

## Assign Huray model on signal

Assign the Huray model on the signal.

```
obj = q.modeler.get_object_from_name("signal")
q.assign_huray_finitecond_to_edges(obj.edges, radius="0.5um", ratio=3, name="b_" + obj.name)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE27CC40>
```

## Create setup, analyze, and plot

Create the setup, analyze it, and plot solution data.

```
setup = q.create_setup(name="new_setup")

sweep = setup.add_sweep(name="sweep1", sweep_type="Discrete")
sweep.props["RangeType"] = "LinearStep"
sweep.props["RangeStart"] = "1GHz"
sweep.props["RangeStep"] = "100MHz"
sweep.props["RangeEnd"] = "5GHz"
sweep.props["SaveFields"] = False
sweep.props["SaveRadFields"] = False
sweep.props["Type"] = "Interpolating"
```

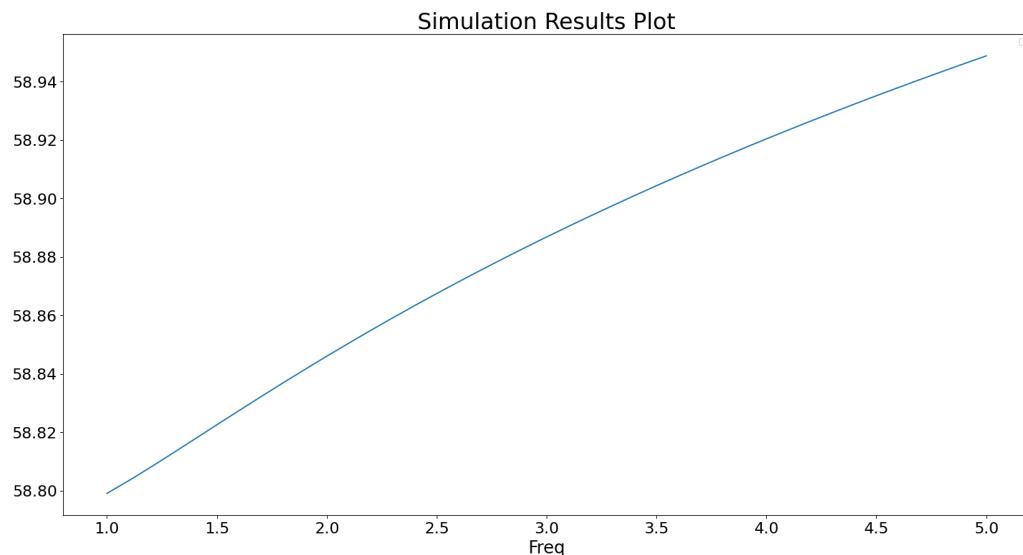
(continues on next page)

(continued from previous page)

```
sweep.update()

q.analyze()

a = q.post.get_solution_data(expressions="Z0(signal,signal)", context="Original")
a.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with ↵ an underscore are ignored when legend() is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Save project and close AEDT

Save the project and close AEDT.

```
home = os.path.expanduser("~/")
q.save_project(os.path.join(home, "Downloads", "pyaedt_example", q.project_name + ".aedt"))
q.release_desktop()
```

True

**Total running time of the script:** (1 minutes 19.382 seconds)

## 2D Extractor: stripline analysis

This example shows how you can use PyAEDT to create a differential stripline design in 2D Extractor and run a simulation.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
project_path = pyaedt.generate_unique_project_name()
```

### Launch AEDT and 2D Extractor

Launch AEDT 2023 R2 in graphical mode and launch 2D Extractor. This example uses SI units.

```
q = pyaedt.Q2d(projectname=project_path,
 designname="differential_stripline",
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 11672 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Define variables

Define variables.

```
e_factor = "e_factor"
sig_w = "sig_bot_w"
sig_gap = "sig_gap"
co_gnd_w = "gnd_w"
clearance = "clearance"
cond_h = "cond_h"
core_h = "core_h"
pp_h = "pp_h"

for var_name, var_value in {
 "e_factor": "2",
 "sig_bot_w": "150um",
 "sig_gap": "150um",
 "gnd_w": "500um",
 "clearance": "150um",
 "cond_h": "17um",
 "core_h": "150um",
 "pp_h": "150um",
}:
 q[var_name] = var_value

delta_w_half = "({}/{})".format(cond_h, e_factor)
sig_top_w = "({}-{}*2)".format(delta_w_half, sig_w)
co_gnd_top_w = "({}-{}*2)".format(delta_w_half, co_gnd_w)
model_w = "{}*2+{}*2+{}*2+{}".format(co_gnd_w, clearance, sig_w, sig_gap)
```

## Create primitives

Create primitives and define the layer heights.

```
layer_1_lh = 0
layer_1_uh = cond_h
layer_2_lh = layer_1_uh + "+" + core_h
layer_2_uh = layer_2_lh + "+" + cond_h
layer_3_lh = layer_2_uh + "+" + pp_h
layer_3_uh = layer_3_lh + "+" + cond_h
```

## Create positive signal

Create a positive signal.

```
base_line_obj = q.modeler.create_polyline([[0, layer_2_lh, 0], [sig_w, layer_2_lh, 0]],
 name="signal_p")
top_line_obj = q.modeler.create_polyline([[0, layer_2_uh, 0], [sig_top_w, layer_2_uh, 0]],
 name="signal_n")
q.modeler.move([top_line_obj], [delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])
q.modeler.move([base_line_obj], ["{}+{}".format(co_gnd_w, clearance), 0, 0])

Create negative signal
~~~~~
Create a negative signal.

base_line_obj = q.modeler.create_polyline(points=[[0, layer_2_lh, 0], [sig_w, layer_2_lh,
 0]], name="signal_p")
top_line_obj = q.modeler.create_polyline(points=[[0, layer_2_uh, 0], [sig_top_w, layer_2_uh,
 0]], name="signal_n")
q.modeler.move(assignment=[top_line_obj], vector=[delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])
q.modeler.move(assignment=[base_line_obj], vector=["{}+{}+{}+{}".format(co_gnd_w,
 clearance, sig_w, sig_gap), 0, 0])
```

True

## Create coplanar ground

Create a coplanar ground.

```
base_line_obj = q.modeler.create_polyline(points=[[0, layer_2_lh, 0], [co_gnd_w, layer_2_lh,
 0]], name="co_gnd_left")
top_line_obj = q.modeler.create_polyline(points=[[0, layer_2_uh, 0], [co_gnd_top_w, layer_2_uh,
 0]], name="co_gnd_right")
q.modeler.move([top_line_obj], [delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])

base_line_obj = q.modeler.create_polyline(points=[[0, layer_2_lh, 0], [co_gnd_w, layer_2_lh,
 0]], name="co_gnd_left")
top_line_obj = q.modeler.create_polyline(points=[[0, layer_2_uh, 0], [co_gnd_top_w, layer_2_uh,
 0]], name="co_gnd_right")
q.modeler.move(assignment=[top_line_obj], vector=[delta_w_half, 0, 0])
q.modeler.connect([base_line_obj, top_line_obj])
q.modeler.move(assignment=[base_line_obj], vector=["{}+{}*2+{}*2+{}".format(co_gnd_w,
 clearance, sig_w, sig_gap), 0, 0])
```

True

## Create reference ground plane

Create a reference ground plane.

```
q.modeler.create_rectangle(origin=[0, layer_1_lh, 0], sizes=[model_w, cond_h], name="ref_gnd_u")
q.modeler.create_rectangle(origin=[0, layer_3_lh, 0], sizes=[model_w, cond_h], name="ref_gnd_l")
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258C56B5120>
```

## Create dielectric

Create a dielectric.

```
q.modeler.create_rectangle(
 origin=[0, layer_1_uh, 0], sizes=[model_w, core_h], name="Core", material="FR4_epoxy")
)
q.modeler.create_rectangle(
 origin=[0, layer_2_uh, 0], sizes=[model_w, pp_h], name="Prepreg", material="FR4_epoxy"
)
q.modeler.create_rectangle(
 origin=[0, layer_2_lh, 0], sizes=[model_w, cond_h], name="Filling", material="FR4_epoxy"
)
```

```
<pyaedt.modeler.cad.object3d.Object3d object at 0x00000258C56B45E0>
```

## Assign conductors

Assign conductors to the signal.

```
obj = q.modeler.get_object_from_name("signal_p")
q.assign_single_conductor(assignment=[obj], name=obj.name, conductor_type="SignalLine",
 solve_option="SolveOnBoundary",
 units="mm")

obj = q.modeler.get_object_from_name("signal_n")
q.assign_single_conductor(assignment=[obj], name=obj.name, conductor_type="SignalLine",
 solve_option="SolveOnBoundary",
 units="mm")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C56B4A00>
```

## Create reference ground

Create a reference ground.

```
obj = [q.modeler.get_object_from_name(i) for i in ["co_gnd_left", "co_gnd_right", "ref_gnd_u", "ref_gnd_l"]]
q.assign_single_conductor(assignment=obj, name="gnd", conductor_type="ReferenceGround", solve_option="SolveOnBoundary", units="mm")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C56B7160>
```

## Assign Huray model on signals

Assign the Huray model on the signals.

```
obj = q.modeler.get_object_from_name("signal_p")
q.assign_huray_finitecond_to_edges(obj.edges, radius="0.5um", ratio=3, name="b_" + obj.name)

obj = q.modeler.get_object_from_name("signal_n")
q.assign_huray_finitecond_to_edges(obj.edges, radius="0.5um", ratio=3, name="b_" + obj.name)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C56DA470>
```

## Define differential pair

Define the differential pair.

```
matrix = q.insert_reduced_matrix(operation_name=q.MATRIXOPERATIONS.DiffPair, assignment=["signal_p", "signal_n"], reduced_matrix="diff_pair")
```

## Create setup, analyze, and plot

Create a setup, analyze, and plot solution data.

```
Create a setup.
setup = q.create_setup(name="new_setup")

Add a sweep.
sweep = setup.add_sweep(name="sweep1", sweep_type="Discrete")
sweep.props["RangeType"] = "LinearStep"
sweep.props["RangeStart"] = "1GHz"
sweep.props["RangeStep"] = "100MHz"
sweep.props["RangeEnd"] = "5GHz"
sweep.props["SaveFields"] = False
sweep.props["SaveRadFields"] = False
```

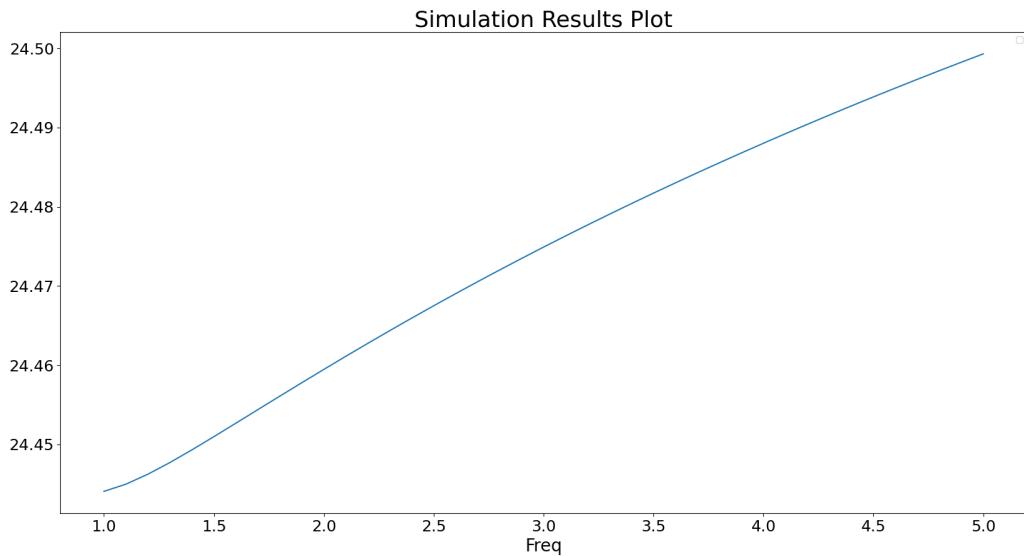
(continues on next page)

(continued from previous page)

```
sweep.props["Type"] = "Interpolating"
sweep.update()

Analyze the nominal design and plot characteristic impedance.
q.analyze()
plot_sources = matrix.get_sources_for_plot(category="Z0")
a = q.post.get_solution_data(expressions=plot_sources, context=matrix.name)
a.plot(snapshot_path=os.path.join(q.working_directory, "plot.jpg")) # Save plot as jpg

Add a parametric sweep and analyze.
parametric = q.parametrics.add(sweep_var="sig_bot_w", start_point=75, end_point=100,
 step=5, variation_type="LinearStep")
parametric.add_variation(sweep_var="sig_gap", start_point="100um", end_point="200um",
 step=5, variation_type="LinearCount")
q.analyze_setup(name=parametric.name)
```



No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.

True

## Save project and release AEDT

Save the project and release AEDT.

```
q.save_project()
q.release_desktop()
```

```
True
```

**Total running time of the script:** (4 minutes 24.125 seconds)

## Q3D Extractor: PCB DCIR analysis

This example shows how you can use PyAEDT to create a design in Q3D Extractor and run a DC IR Drop simulation starting from an EDB Project.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
from pyedb import Edb
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set up project files and path

Download needed project file and set up temporary project directory.

```
project_dir = pyaedt.generate_unique_folder_name()
aedb_project = pyaedt.downloads.download_file('edb/ANSYS-HSD_V1.aedb',
 destination=project_dir)
coil = pyaedt.downloads.download_file('inductance_3d_component', 'air_coil.a3dcomp')
res = pyaedt.downloads.download_file('resistors', 'Res_0402.a3dcomp')
project_name = pyaedt.generate_unique_name("HSD")
output_edb = os.path.join(project_dir, project_name + '_out.aedb')
output_q3d = os.path.join(project_dir, project_name + '_q3d.aedt')
```

## Open EDB

Open the EDB project and create a cutout on the selected nets before exporting to Q3D.

```
edb = Edb(aedb_project, edbversion=aedt_version)
edb.cutout(["1.2V_AVDLL_PLL", "1.2V_AVDDL", "1.2V_DVDDL", "NetR106_1"],
 ["GND"],
 output_aedb_path=output_edb,
 use_pyaedt_extent_computing=True,
)
edb.layout_validation.disjoint_nets("GND", keep_only_main_net=True)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyedb\dotnet\edb_core\
 ↵components.py:185: DeprecationWarning: Use new property :func:`instances` instead.
 warnings.warn("Use new property :func:`instances` instead.", DeprecationWarning)

[]
```

## Identify pin positions

Identify [x,y] pin locations on the components to define where to assign sources and sinks for Q3D.

```
pin_u11_scl = [i for i in edb.components["U11"].pins.values() if i.net_name == "1.2V_
 ↵AVDLL_PLL"]
pin_u9_1 = [i for i in edb.components["U9"].pins.values() if i.net_name == "1.2V_AVDDL"]
pin_u9_2 = [i for i in edb.components["U9"].pins.values() if i.net_name == "1.2V_DVDDL"]
pin_u11_r106 = [i for i in edb.components["U11"].pins.values() if i.net_name == "NetR106_
 ↵1"]
```

## Append Z Positions

Compute Q3D 3D position. The factor 1000 converts from meters to mm.

```
location_u11_scl = [i * 1000 for i in pin_u11_scl[0].position]
location_u11_scl.append(edb.components["U11"].upper_elevation * 1000)

location_u9_1_scl = [i * 1000 for i in pin_u9_1[0].position]
location_u9_1_scl.append(edb.components["U9"].upper_elevation * 1000)

location_u9_2_scl = [i * 1000 for i in pin_u9_2[0].position]
location_u9_2_scl.append(edb.components["U9"].upper_elevation * 1000)

location_u11_r106 = [i * 1000 for i in pin_u11_r106[0].position]
location_u11_r106.append(edb.components["U11"].upper_elevation * 1000)
```

## Identify pin positions for 3D components

Identify the pin positions where 3D components of passives are to be added.

```
location_l2_1 = [i * 1000 for i in edb.components["L2"].pins["1"].position]
location_l2_1.append(edb.components["L2"].upper_elevation * 1000)
location_l4_1 = [i * 1000 for i in edb.components["L4"].pins["1"].position]
location_l4_1.append(edb.components["L4"].upper_elevation * 1000)

location_r106_1 = [i * 1000 for i in edb.components["R106"].pins["1"].position]
location_r106_1.append(edb.components["R106"].upper_elevation * 1000)
```

## Save and close EDB

Save and close EDB. Then, open EDT in HFSS 3D Layout to generate the 3D model.

```
edb.save_edb_as(output_edb)
edb.close_edb()

h3d = pyaedt.Hfss3dLayout(output_edb, specified_version=aedt_version, non_
 ↪graphical=False, new_desktop_session=True)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: ↪
 ↪subprocess 7300 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 ↪py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 ↪log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Export to Q3D

Create a dummy setup and export the layout in Q3D. The `keep_net_name` parameter reassigns Q3D net names from HFSS 3D Layout.

```
setup = h3d.create_setup()
setup.export_to_q3d(output_q3d, keep_net_name=True)
h3d.close_project()
```

```
True
```

## Open Q3D

Launch the newly created q3d project.

```
q3d = pyaedt.Q3d(output_q3d)
q3d.modeler.delete("GND")
q3d.delete_all_nets()
```

```
True
```

## Insert inductors

Create new coordinate systems and place 3D component inductors.

```
q3d.modeler.create_coordinate_system(location_l2_1, name="L2")
comp = q3d.modeler.insert_3d_component(coil, coordinate_system="L2")
comp.rotate(q3d.AXIS.Z, -90)
comp.parameters["n_turns"] = "3"
comp.parameters["d_wire"] = "100um"
q3d.modeler.set_working_coordinate_system("Global")
q3d.modeler.create_coordinate_system(location_l4_1, name="L4")
comp2 = q3d.modeler.insert_3d_component(coil, coordinate_system="L4")
comp2.rotate(q3d.AXIS.Z, -90)
comp2.parameters["n_turns"] = "3"
comp2.parameters["d_wire"] = "100um"
q3d.modeler.set_working_coordinate_system("Global")

q3d.modeler.set_working_coordinate_system("Global")
q3d.modeler.create_coordinate_system(location_r106_1, name="R106")
comp3 = q3d.modeler.insert_3d_component(res, geometry_parameters={'$Resistance': 2000}, ↴
 ↪coordinate_system="R106")
comp3.rotate(q3d.AXIS.Z, -90)

q3d.modeler.set_working_coordinate_system("Global")
```

```
True
```

## Delete dielectrics

Delete all dielectric objects since not needed in DC analysis.

```
q3d.modeler.delete(q3d.modeler.get_objects_by_material("Megtron4"))
q3d.modeler.delete(q3d.modeler.get_objects_by_material("Megtron4_2"))
q3d.modeler.delete(q3d.modeler.get_objects_by_material("Megtron4_3"))
q3d.modeler.delete(q3d.modeler.get_objects_by_material("Solder Resist"))

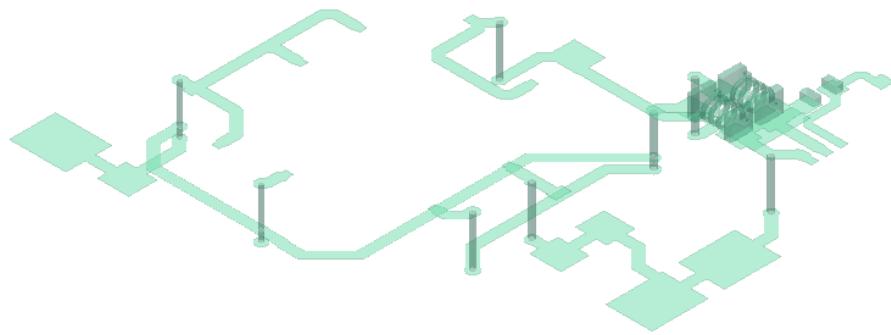
objs_copper = q3d.modeler.get_objects_by_material("copper")
objs_copper_names = [i.name for i in objs_copper]
q3d.plot(show=False, objects=objs_copper_names, plot_as_separate_objects=False,
```

(continues on next page)

(continued from previous page)

```
export_path=os.path.join(q3d.working_directory, "Q3D.jpg"), plot_air_
objects=False)
```

 Model\_AllObjs\_AllMats



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BE299900>
```

### Assign source and sink

Use previously calculated positions to identify faces, select the net 1\_Top and assign sources and sinks on nets.

```
sink_f = q3d.modeler.create_circle(q3d.PLANE.XY, location_u11_scl, 0.1)
source_f1 = q3d.modeler.create_circle(q3d.PLANE.XY, location_u9_1_scl, 0.1)
source_f2 = q3d.modeler.create_circle(q3d.PLANE.XY, location_u9_2_scl, 0.1)
source_f3 = q3d.modeler.create_circle(q3d.PLANE.XY, location_u11_r106, 0.1)
sources_objs = [source_f1, source_f2, source_f3]
q3d.auto_identify_nets()

identified_net = q3d.nets[0]

q3d.sink(sink_f, net_name=identified_net)
```

(continues on next page)

(continued from previous page)

```
source1 = q3d.source(source_f1, net_name=identified_net)

source2 = q3d.source(source_f2, net_name=identified_net)
source3 = q3d.source(source_f3, net_name=identified_net)
sources_bounds = [source1, source2, source3]

q3d.edit_sources(dcrl={"{}": "{}".format(source1.props["Net"], source1.name): "-1.0A",
 "{}": "{}".format(source2.props["Net"], source2.name): "-1.0A",
 "{}": "{}".format(source3.props["Net"], source3.name): "-1.0A"})
```

True

## Create setup

Create a setup and a frequency sweep from DC to 2GHz. Analyze project.

```
setup = q3d.create_setup()
setup.dc_enabled = True
setup.capacitance_enabled = False
setup.ac_rl_enabled = False
setup.props["SaveFields"] = True
setup.props["DC"]["Cond"]["MaxPass"] = 3
setup.analyze()
```

## Field Calculator

We will create a named expression using field calculator.

```
drop_name = "Vdrop3_3"
fields = q3d.ofieldsreporter
q3d.ofieldsreporter.CalcStack("clear")
q3d.ofieldsreporter.EnterQty("Phidc")
q3d.ofieldsreporter.EnterScalar(3.3)
q3d.ofieldsreporter.CalcOp("+")
q3d.ofieldsreporter.AddNamedExpression(drop_name, "DC R/L Fields")
```

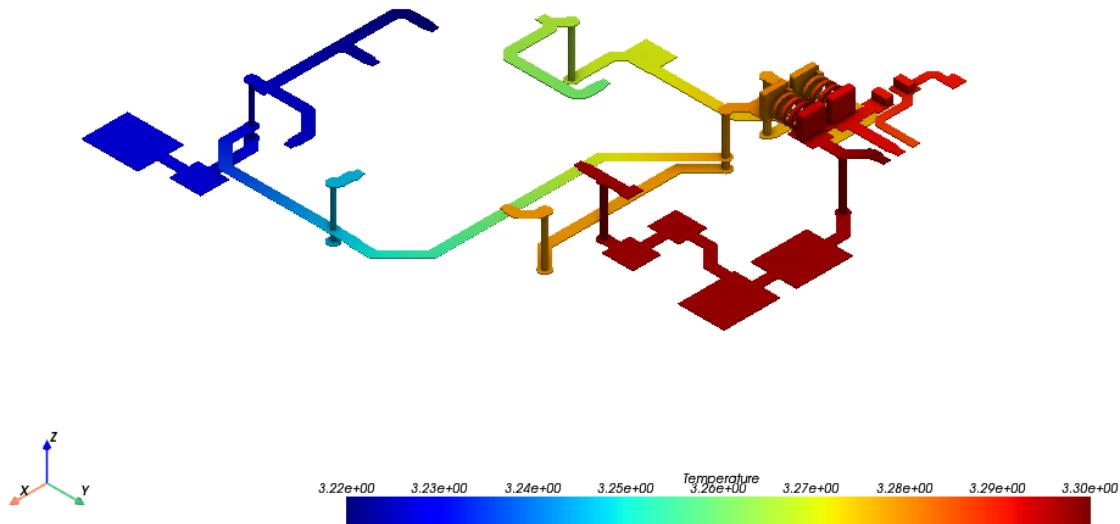
## Phi plot

Compute ACL solutions and plot them.

```
plot1 = q3d.post.create_fieldplot_surface(q3d.modeler.get_objects_by_material("copper"),
 quantity=drop_name,
 intrinsics={"Freq": "1GHz"})

q3d.post.plot_field_from_fieldplot(plot1.name, project_path=q3d.working_directory, mesh_
 ↪plot=False, image_format="jpg",
 ↪view="isometric", show=False, plot_cad_objs=False,
 ↪log_scale=False)
```

 vdrop3\_3\_0FVDJN



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258C623AA70>
```

## Computing Voltage on Source Circles

Using Field Calculator we can compute the voltage on source circles and get the value using `get_solution_data` method.

```
curves = []
for source_circle, source_bound in zip(sources_objs, sources_bounds):
 source_sheet_name = source_circle.name

 curves.append("V{}".format(source_bound.name))

 q3d.fieldsreporter.CalcStack("clear")
 q3d.fieldsreporter.CopyNamedExprToStack(drop_name)
 q3d.fieldsreporter.EnterSurf(source_sheet_name)
 q3d.fieldsreporter.CalcOp("Maximum")
 q3d.fieldsreporter.AddNamedExpression("V{}".format(source_bound.name), "DC R/L"
 Fields")
```

```
data = q3d.post.get_solution_data(
 curves,
```

(continues on next page)

(continued from previous page)

```
 q3d.nominal_adaptive,
 variations={"Freq": "1GHz"},
 report_category="DC R/L Fields",
)
for curve in curves:
 print(data.data_real(curve))
```

```
[3.2604028421728573]
[3.221779035034136]
[3.2863716884360032]
```

## Close AEDT

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

```
q3d.save_project()
q3d.release_desktop()
```

```
True
```

**Total running time of the script:** (7 minutes 29.549 seconds)

## Q3D Extractor: busbar analysis

This example shows how you can use PyAEDT to create a busbar design in Q3D Extractor and run a simulation.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

## Set debugger mode

PyAEDT allows to enable a debug logger which logs all methods called and argument passed. This example shows how to enable it.

```
pyaedt.settings.enable_debug_logger = True
pyaedt.settings.enable_debug_methods_argument_logger = True
pyaedt.settings.enable_debug_internal_methods_logger = False
```

## Launch AEDT and Q3D Extractor

Launch AEDT 2023 R2 in graphical mode and launch Q3D Extractor. This example uses SI units.

```
q = pyaedt.Q3d(projectname=pyaedt.generate_unique_project_name(),
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 8400 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create primitives

Create polylines for three busbars and a box for the substrate.

```
b1 = q.modeler.create_polyline([[0, 0, 0], [-100, 0, 0]], name="Bar1", material="copper",
 ↪ xsection_type="Rectangle",
 ↪ xsection_width="5mm", xsection_height="1mm")
q.modeler["Bar1"].color = (255, 0, 0)

q.modeler.create_polyline([[0, -15, 0], [-150, -15, 0]], name="Bar2", material="aluminum",
 ↪ xsection_type="Rectangle",
 ↪ xsection_width="5mm", xsection_height="1mm")
q.modeler["Bar2"].color = (0, 255, 0)

q.modeler.create_polyline([[0, -30, 0], [-175, -30, 0], [-175, -10, 0]], name="Bar3",
 ↪ material="copper",
 ↪ xsection_type="Rectangle", xsection_width="5mm", xsection_height="1mm")
```

(continues on next page)

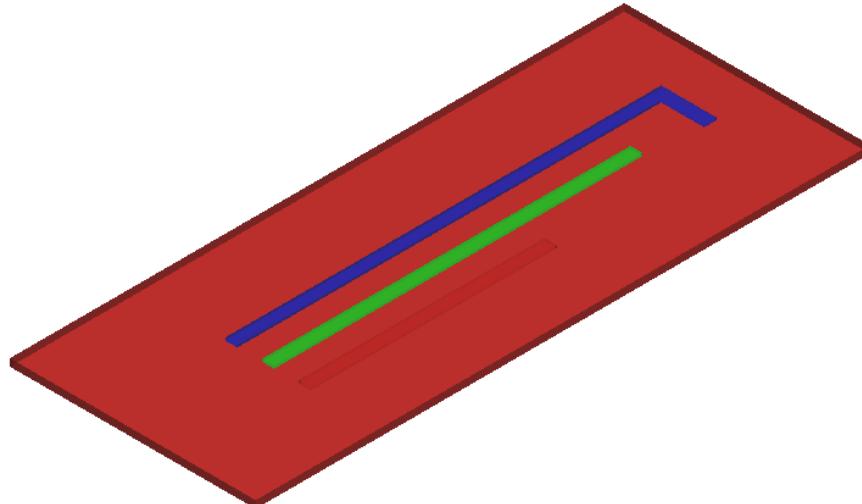
(continued from previous page)

```
↳height="1mm")
q.modeler["Bar3"].color = (0, 0, 255)

q.modeler.create_box([50, 30, -0.5], [-250, -100, -3], name="substrate", material="FR4_
↳epoxy")
q.modeler["substrate"].color = (128, 128, 128)
q.modeler["substrate"].transparency = 0.8

q.plot(show=False, export_path=os.path.join(q.working_directory, "Q3D.jpg"), plot_air_
↳objects=False)
```

Bar1  
Bar2  
Bar3  
substrate



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258F2A05810>
```

## Set up boundaries

Identify nets and assign sources and sinks to all nets. There is a source and sink for each busbar.

```
q.auto_identify_nets()

q.source("Bar1", direction=q.AxisDir.XPos, name="Source1")
q.sink("Bar1", direction=q.AxisDir.XNeg, name="Sink1")

q.source("Bar2", direction=q.AxisDir.XPos, name="Source2")
q.sink("Bar2", direction=q.AxisDir.XNeg, name="Sink2")
q.source("Bar3", direction=q.AxisDir.XPos, name="Source3")
bar3_sink = q.sink("Bar3", direction=q.AxisDir.YPos)
bar3_sink.name = "Sink3"
```

## Print information

Use the different methods available to print net and terminal information.

```
print(q.nets)
print(q.net_sinks("Bar1"))
print(q.net_sinks("Bar2"))
print(q.net_sinks("Bar3"))
print(q.net_sources("Bar1"))
print(q.net_sources("Bar2"))
print(q.net_sources("Bar3"))
```

```
['Bar1', 'Bar2', 'Bar3']
['Sink1']
['Sink2']
['Sink3']
['Source1']
['Source2']
['Source3']
```

## Create setup

Create a setup for Q3D Extractor and add a sweep that defines the adaptive frequency value.

```
setup1 = q.create_setup(props={"AdaptiveFreq": "100MHz"})
sw1 = setup1.add_sweep()
sw1.props["RangeStart"] = "1MHz"
sw1.props["RangeEnd"] = "100MHz"
sw1.props["RangeStep"] = "5MHz"
sw1.update()
```

```
True
```

## Get curves to plot

Get the curves to plot. The following code simplifies the way to get curves.

```
data_plot_self = q.matrices[0].get_sources_for_plot(get_self_terms=True, get_mutual_
 ↴terms=False)
data_plot_mutual = q.get_traces_for_plot(get_self_terms=False, get_mutual_terms=True, ↴
 ↴category="C")
```

## Create rectangular plot

Create a rectangular plot and a data table.

```
q.post.create_report(expressions=data_plot_self)
q.post.create_report(expressions=data_plot_mutual, plot_type="Data Table", context=
 ↴"Original")
```

```
<pyaedt.modules.report_templates.Standard object at 0x00000258C2E41DB0>
```

## Solve setup

Solve the setup.

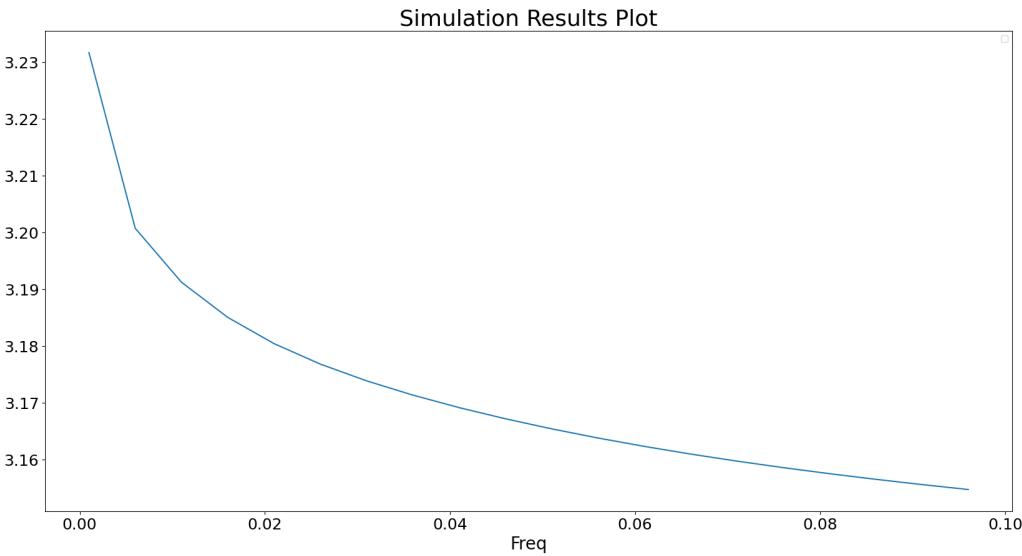
```
q.analyze()
q.save_project()
```

```
True
```

## Get report data

Get the report data into a data structure that allows you to manipulate it.

```
a = q.post.get_solution_data(expressions=data_plot_self, context="Original")
a.plot()
```



```
No artists with labels found to put in legend. Note that artists whose label start with _
an underscore are ignored when legend() is called with no argument.
```

```
<Figure size 2000x1000 with 1 Axes>
```

## Close AEDT

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

```
pyaedt.settings.enable_debug_logger = False
pyaedt.settings.enable_debug_methods_argument_logger = False
q.release_desktop(close_projects=True, close_desktop=True)
```

```
True
```

**Total running time of the script:** (2 minutes 8.991 seconds)

## Q3D Extractor: PCB analysis

This example shows how you can use PyAEDT to create a design in Q3D Extractor and run a simulation starting from an EDB Project.

### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Setup project files and path

Download of needed project file and setup of temporary project directory.

```
project_dir = pyaedt.generate_unique_folder_name()
aedb_project = pyaedt.downloads.download_file('edb/ANSYS-HSD_V1.aedb',
 destination=project_dir)

project_name = pyaedt.generate_unique_name("HSD")
output_edb = os.path.join(project_dir, project_name + '.aedb')
output_q3d = os.path.join(project_dir, project_name + '_q3d.aedt')
```

### Open EDB

Open the edb project and created a cutout on the selected nets before exporting to Q3D.

```
edb = pyaedt.Edb(aedb_project, edbversion=aedt_version)
edb.cutout(["CLOCK_I2C_SCL", "CLOCK_I2C_SDA"], ["GND"], output_aedb_path=output_edb,
 use_pyaedt_extent_computing=True,)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyedb\dotnet\edb_core\
components.py:185: DeprecationWarning: Use new property :func:`instances` instead.
warnings.warn("Use new property :func:`instances` instead.", DeprecationWarning)

[[0.05144426092331123, 0.02870618555822053], [0.051507567732030704, 0.
028387920738624846], [0.05165833184521132, 0.028023943971865067], [0.05183861438361972,
0.027754132085975027], [0.051963277357883246, 0.02760222981232462], [0.
053777229532324555, 0.025788277637883304], [0.05392913180597483, 0.025663614663619865],
[0.05415580049377229, 0.025512159488548385], [0.05441782786354536, 0.
02538745261280736], [0.06473069662795325, 0.022204467947163603], [0.06483058727470829,
```

(continues on next page)

(continued from previous page)

```

↪ [0.02217918947920218], [0.06500618407822036, 0.022144261103311254], [0.
↪ 06520174524046095, 0.02212500001], [0.07179825761953895, 0.02212500001], [0.
↪ 07199381878177934, 0.022144261103311212], [0.0723120836013751, 0.02220756791203067], ↵
↪ [0.07267606036813512, 0.022358332025211397], [0.07294587225402517, 0.
↪ 02253861456361987], [0.07309777452767545, 0.02266327753788331], [0.0740599412389342, 0.
↪ 023625444249142077], [0.07421954256727642, 0.023833440749776783], [0.07443614991723896,
↪ 0.02420861568520458], [0.0745724466120828, 0.02471728187526527], [0.0745724466120828, ↵
↪ 0.026282715184734545], [0.07443614991723915, 0.02679138137479491], [0.
↪ 07415343353250461, 0.02728106051728747], [0.07378106403728785, 0.02765343001250433], ↵
↪ [0.07347367549883911, 0.027830900867923555], [0.07337953919844722, 0.
↪ 027878813335211135], [0.05488997543599179, 0.0360913298157531], [0.054678487766160536, ↵
↪ 0.03615841638650295], [0.05444461588177951, 0.03620493639668876], [0.054249054719539, ↵
↪ 0.03622419749], [0.05351810219436954, 0.03622419749], [0.05325817127675032, 0.
↪ 03618997693549525], [0.05283971800976488, 0.03607785272053635], [0.052383660380235446, ↵
↪ 0.03581454772536155], [0.05198383903463851, 0.03541472637976482], [0.05172053403946369,
↪ 0.03495866875023531], [0.05161337712231406, 0.034558753691050305], [0.
↪ 05157992975286406, 0.03433747779237247], [0.05142573106834311, 0.030307359602748676], ↵
↪ [0.05142499983, 0.0302691225047116], [0.05142499983, 0.028901746720461016]]
```

## Identify pins position

Identify [x,y] pin locations on the components to define where to assign sources and sinks for Q3D and append Z elevation.

```

pin_u13_scl = [i for i in edb.components["U13"].pins.values() if i.net_name == "CLOCK_
↪ I2C_SCL"]
pin_u1_scl = [i for i in edb.components["U1"].pins.values() if i.net_name == "CLOCK_I2C_
↪ SCL"]
pin_u13_sda = [i for i in edb.components["U13"].pins.values() if i.net_name == "CLOCK_
↪ I2C_SDA"]
pin_u1_sda = [i for i in edb.components["U1"].pins.values() if i.net_name == "CLOCK_I2C_
↪ SDA"]
```

## Append Z Positions

Note: The factor 100 converts from meters to mm

```

location_u13_scl = [i * 1000 for i in pin_u13_scl[0].position]
location_u13_scl.append(edb.components["U13"].upper_elevation * 1000)

location_u1_scl = [i * 1000 for i in pin_u1_scl[0].position]
location_u1_scl.append(edb.components["U1"].upper_elevation * 1000)

location_u13_sda = [i * 1000 for i in pin_u13_sda[0].position]
location_u13_sda.append(edb.components["U13"].upper_elevation * 1000)

location_u1_sda = [i * 1000 for i in pin_u1_sda[0].position]
location_u1_sda.append(edb.components["U1"].upper_elevation * 1000)
```

## Save and close Edb

Save, close Edb and open it in Hfss 3D Layout to generate the 3D model.

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

h3d = pyaedt.Hfss3dLayout(output_edb, specified_version=aedt_version, non_graphical=True,
 new_desktop_session=True)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 7148 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Export to Q3D

Create a dummy setup and export the layout in Q3D. keep\_net\_name will reassign Q3D nets names from Hfss 3D Layout.

```
setup = h3d.create_setup()
setup.export_to_q3d(output_q3d, keep_net_name=True)
h3d.close_project()
```

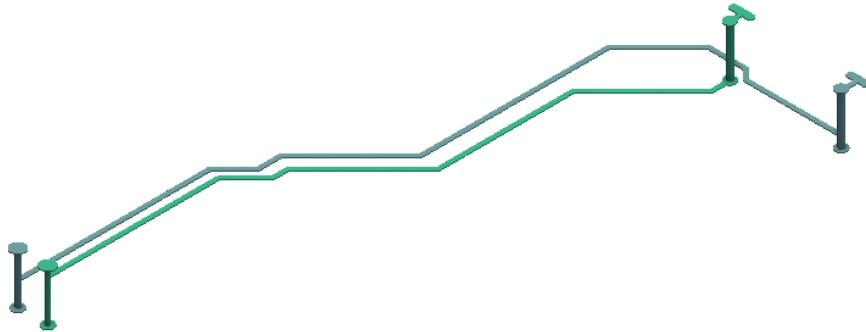
```
True
```

## Open Q3D

Launch the newly created q3d project and plot it.

```
q3d = pyaedt.Q3d(output_q3d)
q3d.plot(show=False, objects=["CLOCK_I2C_SCL", "CLOCK_I2C_SDA"],
 export_path=os.path.join(q3d.working_directory, "Q3D.jpg"), plot_air_
 objects=False)
```

CLOCK_I2C_SCL
CLOCK_I2C_SDA



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258BE2A2CE0>
```

## Assign Source and Sink

Use previously calculated position to identify faces and assign sources and sinks on nets.

```
f1 = q3d.modeler.get_faceid_from_position(location_u13_scl, assignment="CLOCK_I2C_SCL")
q3d.source(f1, net_name="CLOCK_I2C_SCL")
f1 = q3d.modeler.get_faceid_from_position(location_u13_sda, assignment="CLOCK_I2C_SDA")
q3d.source(f1, net_name="CLOCK_I2C_SDA")
f1 = q3d.modeler.get_faceid_from_position(location_u1_scl, assignment="CLOCK_I2C_SCL")
q3d.sink(f1, net_name="CLOCK_I2C_SCL")
f1 = q3d.modeler.get_faceid_from_position(location_u1_sda, assignment="CLOCK_I2C_SDA")
q3d.sink(f1, net_name="CLOCK_I2C_SDA")
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258BE2A04F0>
```

## Create Setup

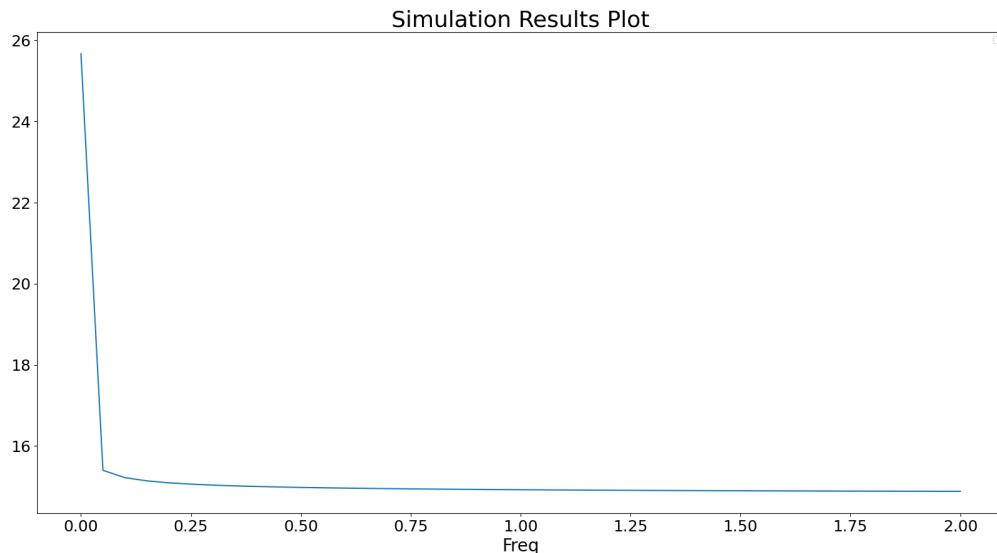
Create a setup and a frequency sweep from DC to 2GHz. Analyze project.

```
setup = q3d.create_setup()
setup.dc_enabled = True
setup.capacitance_enabled = False
sweep = setup.add_sweep()
sweep.add_subrange("LinearStep", 0, end=2, count=0.05, unit="GHz", save_single_=
fields=False, clear=True)
setup.analyze()
```

## ACL Report

Compute ACL solutions and plot them.

```
traces_acl = q3d.post.available_report_quantities(quantities_category="ACL Matrix")
solution = q3d.post.get_solution_data(traces_acl)
solution.plot()
```



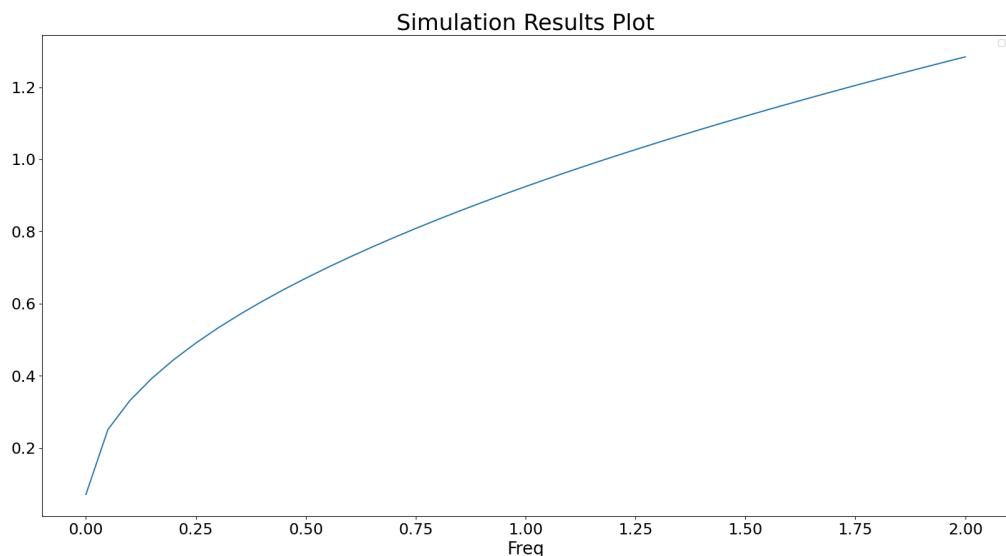
No artists with labels found to put in legend. Note that artists whose label start with `_` are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## ACR Report

Compute ACR solutions and plot them.

```
traces_acr = q3d.post.available_report_quantities(quantities_category="ACR Matrix")
solution2 = q3d.post.get_solution_data(traces_acr)
solution2.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Close AEDT

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

```
q3d.release_desktop()
```

```
True
```

**Total running time of the script:** (10 minutes 47.682 seconds)

### 3.12.9 Multiphysics examples

These examples use PyAEDT to create some multiphysics workflows. They might use an electromagnetic tool like HFSS or Maxwell and a thermal or structural tool like Icepak or Mechanical.

#### Multiphysics: HFSS-Mechanical multiphysics analysis

This example shows how you can use PyAEDT to create a multiphysics workflow that includes Circuit, HFSS, and Mechanical.

##### Perform required imports

Perform required imports.

```
import os
import pyaedt
```

##### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

##### Set non-graphical mode

Set non-graphical mode. You can set non\_graphical either to True or False.

```
non_graphical = False
```

##### Download and open project

Download and open the project. Save it to the temporary folder.

```
project_temp_name = pyaedt.downloads.download_via_wizard(pyaedt.generate_unique_folder_
↳_name())
```

##### Start HFSS

Start HFSS and initialize the PyAEDT object.

```
hfss = pyaedt.Hfss(projectname=project_temp_name, specified_version=aedt_version, non_
↳graphical=non_graphical,
 new_desktop_session=True)
pin_names = hfss.excitations
hfss.change_material_override(True)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 9064 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
True
```

## Start Circuit

Start Circuit and add the HFSS dynamic link component to it.

```
circuit = pyaedt.Circuit()
hfss_comp = circuit.modeler.schematic.add_subcircuit_dynamic_link(hfss)
```

## Set up dynamic link options

Set up dynamic link options. The argument for the `set_sim_option_on_hfss_subcircuit` method can be the component name, component ID, or component object.

```
circuit.modeler.schematic.refresh_dynamic_link(hfss_comp.composed_name)
circuit.modeler.schematic.set_sim_option_on_hfss_subcircuit(hfss_comp)
hfss_setup_name = hfss.setups[0].name + " : " + hfss.setups[0].sweeps[0].name
circuit.modeler.schematic.set_sim_solution_on_hfss_subcircuit(hfss_comp.composed_name, hfss_setup_name)
```

```
True
```

## Create ports and excitations

Create ports and excitations. Find component pin locations and create interface ports on them. Define the voltage source on the input port.

```
circuit.modeler.schematic.create_interface_port(
 name="Excitation_1", location=[hfss_comp.pins[0].location[0], hfss_comp.pins[0].location[1]]
)
circuit.modeler.schematic.create_interface_port(
 name="Excitation_2", location=[hfss_comp.pins[1].location[0], hfss_comp.pins[1].location[1]]
)
circuit.modeler.schematic.create_interface_port(
 name="Port_1", location=[hfss_comp.pins[2].location[0], hfss_comp.pins[2].location[1]]
)
circuit.modeler.schematic.create_interface_port(
```

(continues on next page)

(continued from previous page)

```
 name="Port_2", location=[hfss_comp.pins[3].location[0], hfss_comp.pins[3].
 ↪location[1]]
)

voltage = 1
phase = 0
ports_list = ["Excitation_1", "Excitation_2"]
source = circuit.assign_voltage_sinusoidal_excitation_to_ports(ports_list)
source.ac_magnitude = voltage
source.phase = phase
```

## Create setup

Create a setup.

```
setup_name = "MySetup"
LNA_setup = circuit.create_setup(name=setup_name)
bw_start = 4.3
bw_stop = 4.4
n_points = 1001
unit = "GHz"
sweep_list = ["LINC", str(bw_start) + unit, str(bw_stop) + unit, str(n_points)]
LNA_setup.props["SweepDefinition"]["Data"] = " ".join(sweep_list)
```

## Solve and push excitations

Solve the circuit and push excitations to the HFSS model to calculate the correct value of losses.

```
circuit.analyze()
circuit.push_excitations(instance="S1", setup=setup_name)
```

```
True
```

## Start Mechanical

Start Mechanical and copy bodies from the HFSS project.

```
mech = pyaedt.Mechanical()
mech.copy_solid_bodies_from(hfss)
mech.change_material_override(True)
```

```
True
```

## Get losses from HFSS and assign convection to Mechanical

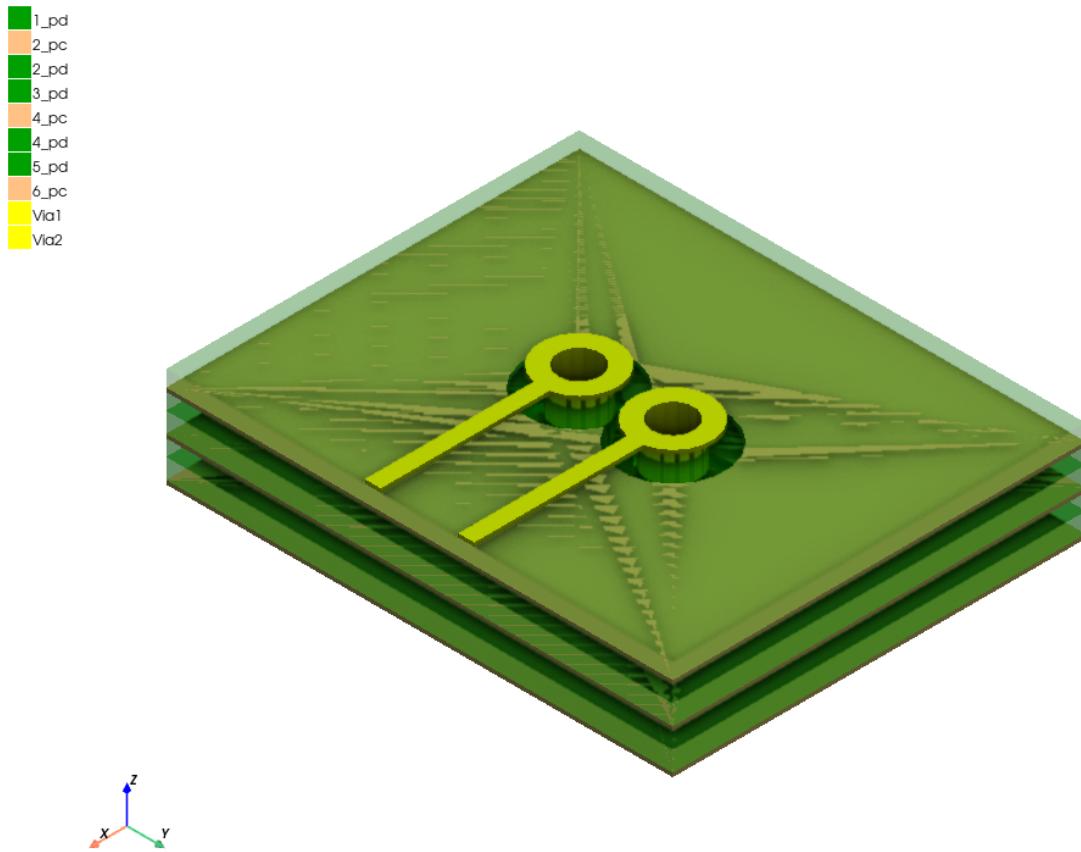
Get losses from HFSS and assign the convection to Mechanical.

```
mech.assign_em_losses(design=hfss.design_name, setup=hfss.setups[0].name, sweep=
 "LastAdaptive",
 map_frequency=hfss.setups[0].props["Frequency"], surface_
 objects=hfss.get_all_conductors_names())
diels = ["1_pd", "2_pd", "3_pd", "4_pd", "5_pd"]
for el in diels:
 mech.assign_uniform_convection(assignment=[mech.modeler[el].top_face_y, mech.
 modeler[el].bottom_face_y],
 convection_value=3)
```

## Plot model

Plot the model.

```
mech.plot(show=False, export_path=os.path.join(mech.working_directory, "Mech.jpg"), plot_.
 air_objects=False)
```



```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258A9B14130>
```

### Solve and plot thermal results

Solve and plot the thermal results.

```
mech.create_setup()
mech.save_project()
mech.analyze()
surfaces = []
for name in mech.get_all_conductors_names():
 surfaces.extend(mech.modeler.get_object_faces(name))
mech.post.create_fieldplot_surface(assignment=surfaces, quantity="Temperature")
```

```
<pyaedt.modules.solutions.FieldPlot object at 0x00000258C47B69B0>
```

### Release AEDT

Release AEDT.

```
mech.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (5 minutes 26.692 seconds)

### Multiphysics: HFSS-Mechanical MRI analysis

The goal of this workshop is to use a coil tuned to 63.8 MHz to determine the temperature rise in a gel phantom near an implant given a background SAR of 1 W/kg.

Steps to follow Step 1: Simulate coil loaded by empty phantom: Scale input to coil ports to produce desired background SAR of 1 W/kg at location that will later contain the implant. Step 2: Simulate coil loaded by phantom containing implant in proper location: View SAR in tissue surrounding implant. Step 3: Thermal simulation: Link HFSS to transient thermal solver to find temperature rise in tissue near implant vs. time.

### Perform required imports

Perform required imports.

```
import os.path

from pyaedt import Hfss, Mechanical, Icepak, downloads
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

## Project load

Open the ANSYS Electronics Desktop 2018.2 Open project `background_SAR.aedt` Project contains phantom and air-box Phantom consists of two objects: phantom and implant\_box Separate objects are used to selectively assign mesh operations Material properties defined in this project already contain #electrical and thermal properties.

```
project_path = downloads.download_file(directory="mri")
hfss = Hfss(os.path.join(project_path, "background_SAR.aedt"), specified_version=aedt_
version, non_graphical=non_graphical,
new_desktop_session=True)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
↳ subprocess 5908 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
↳ py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
↳ log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Insert 3D component

The MRI Coil is saved as a separate 3D Component 3D Components store geometry (including parameters), material properties, boundary conditions, mesh assignments, and excitations 3D Components make it easy to reuse and share parts of a simulation

```
hfss.modeler.insert_3d_component(os.path.join(project_path, "coil.a3dcomp"))
```

```
<pyaedt.modeler.cad.components_3d.UserDefinedComponent object at 0x00000258C2DA4880>
```

## Expression Cache

On the expression cache tab, define additional convergence criteria for self impedance of the four coil ports. Set each of these convergence criteria to 2.5 ohm. For this demo number of passes is limited to 2 to reduce simulation time.

```
im_traces = hfss.get_traces_for_plot(get_mutual_terms=False, category="im(Z", first_
→element_filter="Coil1_p*")

hfss.setups[0].enable_expression_cache(
 report_type="Modal Solution Data",
 expressions=im_traces,
 isconvergence=True,
 isrelativeconvergence=False,
 conv_criteria=2.5,
 use_cache_for_freq=False)
hfss.setups[0].props["MaximumPasses"] = 2
```

## Edit Sources

The 3D Component of the MRI Coil contains all the ports, but the sources for these ports are not yet defined. Browse to and select sources.csv. These sources were determined by tuning this coil at 63.8 MHz. Notice the \*input\_scale multiplier to allow quick adjustment of the coil excitation power.

```
hfss.edit_sources_from_file(os.path.join(project_path, "sources.csv"))
```

```
True
```

## Run Simulation

Save and analyze the project.

```
hfss.save_project(os.path.join(project_path, "solved.aedt"))
hfss.analyze(cores=6)
```

```
True
```

## Plot SAR on cut plane in phantom

Ensure that the SAR averaging method is set to Gridless Plot averagedSAR on GlobalYZ plane. Draw Point1 at origin of the implant coordinate system

```
hfss.sar_setup(-1, tissue_mass=1, material_density=1, average_sar_method=1)
hfss.post.create_fieldplot_cutplane(assignment="implant:YZ", quantity="Average_SAR",_
→filter_objects=["implant_box"])

hfss.modeler.set_working_coordinate_system("implant")
hfss.modeler.create_point([0, 0, 0], name="Point1")
```

(continues on next page)

(continued from previous page)

```
hfss.post.plot_field(quantity="Average_SAR", assignment="implant:YZ", plot_type="CutPlane"
 ↵", show=False,
 show_legend=False, filter_objects=["implant_box"])
```

```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258C2208EB0>
```

## Adjust Input Power to MRI Coil

The goal is to adjust the MRI coils input power, so that the averageSAR at Point1 is 1 W/kg Note that SAR and input power are linearly related To determine required input, calculate input\_scale = 1/AverageSAR at Point1

```
sol_data = hfss.post.get_solution_data(expressions="Average_SAR",
 primary_sweep_variable="Freq",
 context="Point1",
 report_category="Fields")
sol_data.data_real()

hfss["input_scale"] = 1 / sol_data.data_real()[0]
```

## Phantom with Implant

Import implant geometry. Subtract rod from implant\_box. Assign titanium to the imported object rod. Analyze the project.

```
hfss.modeler.import_3d_cad(os.path.join(project_path, "implant_rod.sat"))

hfss.modeler["implant_box"].subtract("rod", keep_originals=True)
hfss.modeler["rod"].material_name = "titanium"
hfss.analyze(cores=6)
hfss.save_project()
```

```
True
```

## Thermal Simulation

Initialize a new Mechanical Transient Thermal analysis. Mechanical Transient Thermal is available in AEDT from 2023 R2 as a Beta feature.

```
mech = Mechanical(solution_type="Transient Thermal", specified_version=aedt_version)
```

## Copy geometries

Copy bodies from the HFSS project. 3D Component will not be copied.

```
mech.copy_solid_bodies_from(hfss)
```

```
True
```

## Link sources to EM losses

Link sources to the EM losses. Assign external convection.

```
exc = mech.assign_em_losses(design=hfss.design_name, setup=hfss.setups[0].name, sweep=
 ↪ "LastAdaptive",
 map_frequency=hfss.setups[0].props["Frequency"],
 surface_objects=mech.get_all_conductors_names())
mech.assign_uniform_convection(mech.modeler["Region"].faces, convection_value=1)
```

```
<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C73B8340>
```

## Create Setup

Create a new setup and edit properties. Simulation will be for 60 seconds.

```
setup = mech.create_setup()
setup.add_mesh_link("backgroundSAR")
mech.create_dataset1d_design("PowerMap", [0, 239, 240, 360], [1, 1, 0, 0])
exc.props["LossMultiplier"] = "pwl(PowerMap,Time)"

mech.modeler.set_working_coordinate_system("implant")
mech.modeler.create_point([0, 0, 0], name="Point1")
setup.props["Stop Time"] = 60
setup.props["Time Step"] = "10s"
setup.props["SaveFieldsType"] = "Every N Steps"
setup.props["N Steps"] = "2"
```

## Analyze Mechanical

Analyze the project.

```
mech.analyze(cores=6)
```

```
True
```

## Plot fields

Plot Temperature on cut plane. Plot Temperature on point.

```
mech.post.create_fieldplot_cutplane("implant:YZ", "Temperature", filter_objects=[
 "implant_box"])
mech.save_project()

data = mech.post.get_solution_data("Temperature", primary_sweep_variable="Time", context=
 "Point1",
 report_category="Fields")
#data.plot()

mech.post.plot_animated_field(quantity="Temperature", assignment="implant:YZ", plot_type=
 "CutPlane",
 intrinsics={"Time": "10s"}, variation_variable="Time",
 variations=["10s", "20s", "30s", "40s", "50s", "60s"],
 show=False, filter_objects=["implant_box"])
```

```
<pyaedt.generic.plot.ModelPlotter object at 0x00000258C73BB4C0>
```

## Thermal Simulation

Initialize a new Icepak Transient Thermal analysis.

```
ipk = Icepak(solution_type="Transient", specified_version=aedt_version)
ipk.design_solutions.problem_type = "TemperatureOnly"
```

## Copy geometries

Copy bodies from the HFSS project. 3D Component will not be copied.

```
ipk.modeler.delete("Region")
ipk.copy_solid_bodies_from(hfss)
```

```
True
```

## Link sources to EM losses

Link sources to the EM losses. Assign external convection.

```
exc = ipk.assign_em_losses(design=hfss.design_name, setup=hfss.setups[0].name, sweep=
 "LastAdaptive",
 map_frequency=hfss.setups[0].props["Frequency"],
 surface_objects=ipk.get_all_conductors_names())
```

## Create Setup

Create a new setup and edit properties. Simulation will be for 60 seconds.

```
setup = ipk.create_setup()

setup.props["Stop Time"] = 60
setup.props["N Steps"] = 2
setup.props["Time Step"] = 5
setup.props['Convergence Criteria - Energy'] = 1e-12
```

## Mesh Region

Create a new mesh region and change accuracy level to 4.

```
bound = ipk.modeler["implant_box"].bounding_box
mesh_box = ipk.modeler.create_box(bound[:3], [bound[3] - bound[0], bound[4] - bound[1],
 ↵ bound[5] - bound[2]])
mesh_box.model = False
mesh_region = ipk.mesh.assign_mesh_region([mesh_box.name])
mesh_region.UserSpecifiedSettings = False
mesh_region.Level = 4
mesh_region.update()
```

True

## Point Monitor

Create a new point monitor.

```
ipk.modeler.set_working_coordinate_system("implant")
ipk.monitor.assign_point_monitor([0, 0, 0], monitor_name="Point1")
ipk.assign_openings(ipk.modeler["Region"].top_face_z)
```

<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C4A91C90>

## Analyze and plot fields

Analyze the project. Plot temperature on cut plane. Plot temperature on monitor point.

```
ipk.analyze(cores=4, tasks=4)
ipk.post.create_fieldplot_cutplane("implant:YZ", "Temperature", filter_objects=["implant_"
 ↵ box"])
ipk.save_project()

data = ipk.post.get_solution_data("Point1.Temperature", primary_sweep_variable="Time",

 ↵ report_category="Monitor")
#data.plot()
```

(continues on next page)

(continued from previous page)

```
ipk.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (27 minutes 47.433 seconds)

### Multiphysics: Maxwell 3D - Icepak electrothermal analysis

This example uses PyAEDT to set up a simple Maxwell design consisting of a coil and a ferrite core. Coil current is set to 100A, and coil resistance and ohmic loss are analyzed. Ohmic loss is mapped to Icepak, and a thermal analysis is performed. Icepak calculates a temperature distribution, and it is mapped back to Maxwell (2-way coupling). Coil resistance and ohmic loss are analyzed again in Maxwell. Results are printed in AEDT Message Manager.

#### Perform required imports

Perform required imports.

```
import pyaedt
from pyaedt.generic.constants import AXIS
```

#### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

#### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

#### Launch AEDT and Maxwell 3D

Launch AEDT and Maxwell 3D. The following code sets up the project and design names, the solver, and the version. It also creates an instance of the `Maxwell3d` class named `m3d`.

```
project_name = "Maxwell-Icepak-2way-Coupling"
maxwell_design_name = "1 Maxwell"
icepak_design_name = "2 Icepak"

m3d = pyaedt.Maxwell3d(
 projectname=project_name,
 designname=maxwell_design_name,
 solution_type="EddyCurrent",
```

(continues on next page)

(continued from previous page)

```
 specified_version=aedt_version,
 non_graphical=non_graphical,
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 9344 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create geometry in Maxwell

Create the coil, coil terminal, core, and region.

```
coil = m3d.modeler.create_rectangle(
 orientation="XZ", origin=[70, 0, -11], sizes=[11, 110], name="Coil"
)

coil.sweep_around_axis(axis=AXIS.Z)

coil_terminal = m3d.modeler.create_rectangle(
 orientation="XZ", origin=[70, 0, -11], sizes=[11, 110], name="Coil_terminal"
)

core = m3d.modeler.create_rectangle(
 orientation="XZ", origin=[45, 0, -18], sizes=[7, 160], name="Core"
)
core.sweep_around_axis(axis=AXIS.Z)

Magnetic flux is not concentrated by the core in +z-direction. Therefore, more padding
is needed in that direction.
region = m3d.modeler.create_region(pad_percent=[20, 20, 500, 20, 20, 100])
```

## Assign materials

Create a material: Copper AWG40 Litz wire, strand diameter = 0.08mm, 24 parallel strands. Assign materials: Assign Coil to AWG40 copper, core to ferrite, and region to vacuum.

```
no_strands = 24
strand_diameter = 0.08

cu_litz = m3d.materials.duplicate_material("copper", "copper_litz")
cu_litz.stack_type = "Litz Wire"
cu_litz.wire_diameter = str(strand_diameter) + "mm"
cu_litz.wire_type = "Round"
cu_litz.strand_number = no_strands
```

(continues on next page)

(continued from previous page)

```
m3d.assign_material(region.name, "vacuum")
m3d.assign_material(coil.name, "copper_litz")
m3d.assign_material(core.name, "ferrite")
```

```
True
```

## Assign excitation

Assign coil current, coil consists of 20 turns, total current 10A. Note that each coil turn consists of 24 parallel Litz strands, see above.

```
no_turns = 20
coil_current = 10
m3d.assign_coil(["Coil_terminal"], conductors_number=no_turns, name="Coil_terminal")
m3d.assign_winding(is_solid=False, current=coil_current, name="Winding1")

m3d.add_winding_coils(assignment="Winding1", coils=["Coil_terminal"])
```

```
True
```

## Assign mesh operations

Mesh operations are not necessary in eddy current solver because of auto-adaptive meshing. However, with appropriate mesh operations, less adaptive passes are needed.

```
m3d.mesh.assign_length_mesh(["Core"], maximum_length=15, maximum_elements=None, name=
 ↪"Inside_Core")
m3d.mesh.assign_length_mesh(["Coil"], maximum_length=30, maximum_elements=None, name=
 ↪"Inside_Coil")
```

```
<pyaedt.modules.Mesh.MeshOperation object at 0x00000258F6AB1840>
```

## Set conductivity temperature coefficient

Set conductivity as a function of temperature. Resistivity increases by 0.393% per K.

```
cu_resistivity_temp_coefficient = 0.00393
cu_litz.conductivity.add_thermal_modifier_free_form("1.0/(1.0+{}*(Temp-20))".format(cu_
 ↪resistivity_temp_coefficient))
```

```
True
```

## Set object temperature and enable feedback

Set the temperature of the objects to default temperature (22deg C) and enable temperature feedback for two-way coupling.

```
m3d.modeler.set_objects_temperature(["Coil"])
```

```
True
```

## Assign matrix

Resistance and inductance calculation.

```
m3d.assign_matrix(["Winding1"], matrix_name="Matrix1")
```

```
<pyaedt.modules.Boundary.MaxwellParameters object at 0x00000258F6AB0430>
```

## Create and analyze simulation setup

Simulation frequency 150kHz.

```
setup = m3d.create_setup(name="Setup1")
setup.props["Frequency"] = "150kHz"
m3d.analyze_setup("Setup1")
```

```
True
```

## Postprocessing

Calculate analytical DC resistance and compare it with the simulated coil resistance, print them in the message manager, as well as ohmic loss in coil before temperature feedback.

```
report = m3d.post.create_report(expressions="Matrix1.R(Winding1,Winding1)")
solution = report.get_solution_data()
resistance = solution.data_magnitude()[0]

report_loss = m3d.post.create_report(expressions="StrandedLossAC")
solution_loss = report_loss.get_solution_data()
em_loss = solution_loss.data_magnitude()[0]

Analytical calculation of the DC resistance of the coil
cu_cond = float(cu_litz.conductivity.value)
average radius of a coil turn = 0.125m
l_conductor = no_turns*2*0.125*3.1415
R = resistivity * length / area / no_strand
r_analytical_DC = (1.0 / cu_cond) * l_conductor / (3.1415 * (strand_diameter / 1000 / 2)**2) / no_strands
```

(continues on next page)

(continued from previous page)

```
Print results in the Message Manager
m3d.logger.info("*****Coil analytical DC resistance = {:.2f}Ohm".format(r_analytical_
 ↴DC))
m3d.logger.info("*****Coil resistance at 150kHz BEFORE temperature feedback = {:.2f}_
 ↴Ohm".format(resistance))
m3d.logger.info("*****Ohmic loss in coil BEFORE temperature feedback = {:.2f}W".
 ↴format(em_loss / 1000))
```

## Icepak design

Insert Icepak design, copy solid objects from Maxwell, and modify region dimensions.

```
ipk = pyaedt.Icepak(designname=icepak_design_name)
ipk.copy_solid_bodies_from(m3d, no_pec=False)

Set domain dimensions suitable for natural convection using the diameter of the coil
ipk.modeler["Region"].delete()
coil_dim = coil.bounding_dimension[0]
ipk.modeler.create_region(0, False)
ipk.modeler.edit_region_dimensions([coil_dim / 2, coil_dim / 2, coil_dim / 2, coil_dim /_
 ↴2, coil_dim * 2, coil_dim])
```

True

## Map coil losses

Map ohmic losses from Maxwell to the Icepak design.

```
ipk.assign_em_losses(design="1 Maxwell", setup=m3d.setups[0].name, sweep="LastAdaptive",_
 ↴assignment=["Coil"])
```

<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C73EC490>

## Boundary conditions

Assign opening.

```
faces = ipk.modeler["Region"].faces
face_names = [face.id for face in faces]
ipk.assign_free_opening(face_names, boundary_name="Opening1")
```

<pyaedt.modules.Boundary.BoundaryObject object at 0x00000258C73EC0D0>

## Assign monitor

Temperature monitor on the coil surface

```
temp_monitor = ipk.assign_point_monitor([70, 0, 0], monitor_name="PointMonitor1")
```

## Icepak solution setup

```
solution_setup = ipk.create_setup()
solution_setup.props["Convergence Criteria - Max Iterations"] = 50
solution_setup.props["Flow Regime"] = "Turbulent"
solution_setup.props["Turbulent Model Eqn"] = "ZeroEquation"
solution_setup.props["Radiation Model"] = "Discrete Ordinates Model"
solution_setup.props["Include Flow"] = True
solution_setup.props["Include Gravity"] = True
solution_setup.props["Solution Initialization - Z Velocity"] = "0.0005m_per_sec"
solution_setup.props["Convergence Criteria - Flow"] = 0.0005
solution_setup.props["Flow Iteration Per Radiation Iteration"] = "5"
```

## Add 2-way coupling and solve the project

Enable mapping temperature distribution back to Maxwell. Default number Maxwell <-> Icepak iterations is 2, but for increased accuracy it can be increased (number\_of\_iterations).

```
ipk.assign_2way_coupling()
ipk.analyze_setup(name=solution_setup.name)
```

```
True
```

## Postprocessing

Plot temperature on the object surfaces.

```
surface_list = []
for name in ["Coil", "Core"]:
 surface_list.extend(ipk.modeler.get_object_faces(name))

surf_temperature = ipk.post.create_fieldplot_surface(surface_list, quantity=
 "SurfTemperature",
 plot_name="Surface Temperature")

velocity_cutplane = ipk.post.create_fieldplot_cutplane(assignment=["Global:XZ"],
 quantity="Velocity Vectors",
 plot_name="Velocity Vectors")

surf_temperature.export_image()
velocity_cutplane.export_image(orientation="right")
```

(continues on next page)

(continued from previous page)

```
report_temp = ipk.post.create_report(expressions="PointMonitor1.Temperature", primary_
↪sweep_variable="X")
solution_temp = report_temp.get_solution_data()
temp = solution_temp.data_magnitude()[0]
m3d.logger.info("*****Coil temperature = {:.2f}deg C".format(temp))
```

## Get new resistance from Maxwell

Temperature of the coil increases, and consequently also coil resistance increases.

```
report_new = m3d.post.create_report(expressions="Matrix1.R(Winding1,Winding1)")
solution_new = report_new.get_solution_data()
resistance_new = solution_new.data_magnitude()[0]
resistance_increase = (resistance_new - resistance)/resistance * 100

report_loss_new = m3d.post.create_report(expressions="StrandedLossAC")
solution_loss_new = report_loss_new.get_solution_data()
em_loss_new = solution_loss_new.data_magnitude()[0]

m3d.logger.info("*****Coil resistance at 150kHz AFTER temperature feedback = {:.2f}Ohm
↪".format(resistance_new))
m3d.logger.info("*****Coil resistance increased by {:.2f}%".format(resistance_
↪increase))
m3d.logger.info("*****Ohmic loss in coil AFTER temperature feedback = {:.2f}W".
↪format(em_loss_new/1000))
```

## Release desktop

```
ipk.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (6 minutes 59.761 seconds)

### 3.12.10 Circuit examples

These examples use PyAEDT to show some end-to-end workflows for Circuit. This includes schematic generation, setup, and postprocessing.

#### Circuit: AMI PostProcessing

This example shows how you can use PyAEDT to perform advanced postprocessing of AMI simulations.

##### Perform required imports

Perform required imports and set the local path to the path for PyAEDT.

```
sphinx_gallery_thumbnail_path = 'Resources/spectrum_plot.png'

import os
from matplotlib import pyplot as plt
import numpy as np

import pyaedt

Set local path to path for PyAEDT
temp_folder = pyaedt.generate_unique_folder_name()
project_path = pyaedt.downloads.download_file("ami", "ami_usb.aedtz", temp_folder)
```

##### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

##### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False. The Boolean parameter `new_thread` defines whether to create a new instance of AEDT or try to connect to an existing instance of it.

```
non_graphical = False
NewThread = True
```

##### Launch AEDT with Circuit and enable Pandas as the output format

All outputs obtained with the `get_solution_data` method will have the Pandas format. Launch AEDT with Circuit. The `pyaedt.Desktop` class initializes AEDT and starts the specified version in the specified mode.

```
pyaedt.settings.enable_pandas_output = True
cir = pyaedt.Circuit(projectname=os.path.join(project_path), non_graphical=non_graphical,
 specified_version=aedt_version, new_desktop_session=NewThread)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 11476 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Solve AMI setup

Solve the transient setup.

```
cir.analyze()
```

```
True
```

## Get AMI report

Get AMI report data

```
plot_name = "WaveAfterProbe<b_input_43.int_ami_rx>"
cir.solution_type = "NexximAMI"
original_data = cir.post.get_solution_data(expressions=plot_name, domain="Time",
 variations=cir.available_variations.nominal)
original_data_value = original_data.full_matrix_real_imag[0]
original_data_sweep = original_data.primary_sweep_values
print(original_data_value)
```

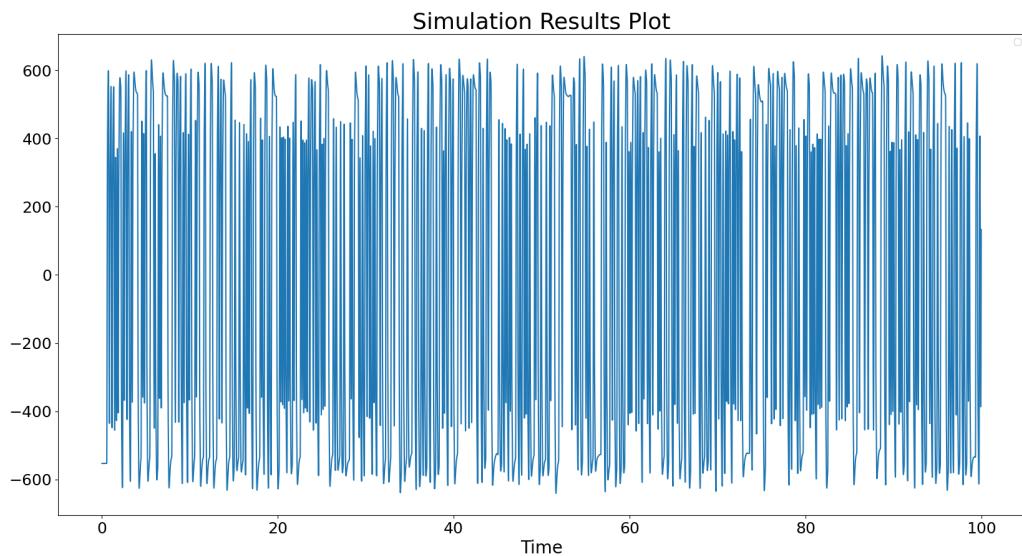
	WaveAfterProbe<b_input_43.int_ami_rx>
0.000000	-553.298382
0.003125	-553.298382
0.006250	-553.298382
0.009375	-553.298382
0.012500	-553.298382
...	...
99.984375	-25.138119
99.987500	19.046320
99.990625	60.268984
99.993750	98.348353
99.996875	133.328724

```
[32000 rows x 1 columns]
```

## Plot data

Create a plot based on solution data.

```
fig = original_data.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with ↵ an underscore are ignored when legend() is called with no argument.

## Sample WaveAfterProbe waveform using receiver clock

Extract waveform at specific clock time plus half unit interval

```
probe_name = "b_input_43"
source_name = "b_output4_42"
plot_type = "WaveAfterProbe"
setup_name = "AMIAAnalysis"
ignore_bits = 100
unit_interval = 0.1e-9
sample_waveform = cir.post.sample_ami_waveform(setup=setup_name, probe=probe_name,
 ↵source=source_name,
 variation_list_w_value=cir.available_
 ↵variations.nominal,
 unit_interval=unit_interval, ignore_
 ↵bits=ignore_bits,
 plot_type=plot_type)
```

## Plot waveform and samples

Create the plot from a start time to stop time in seconds

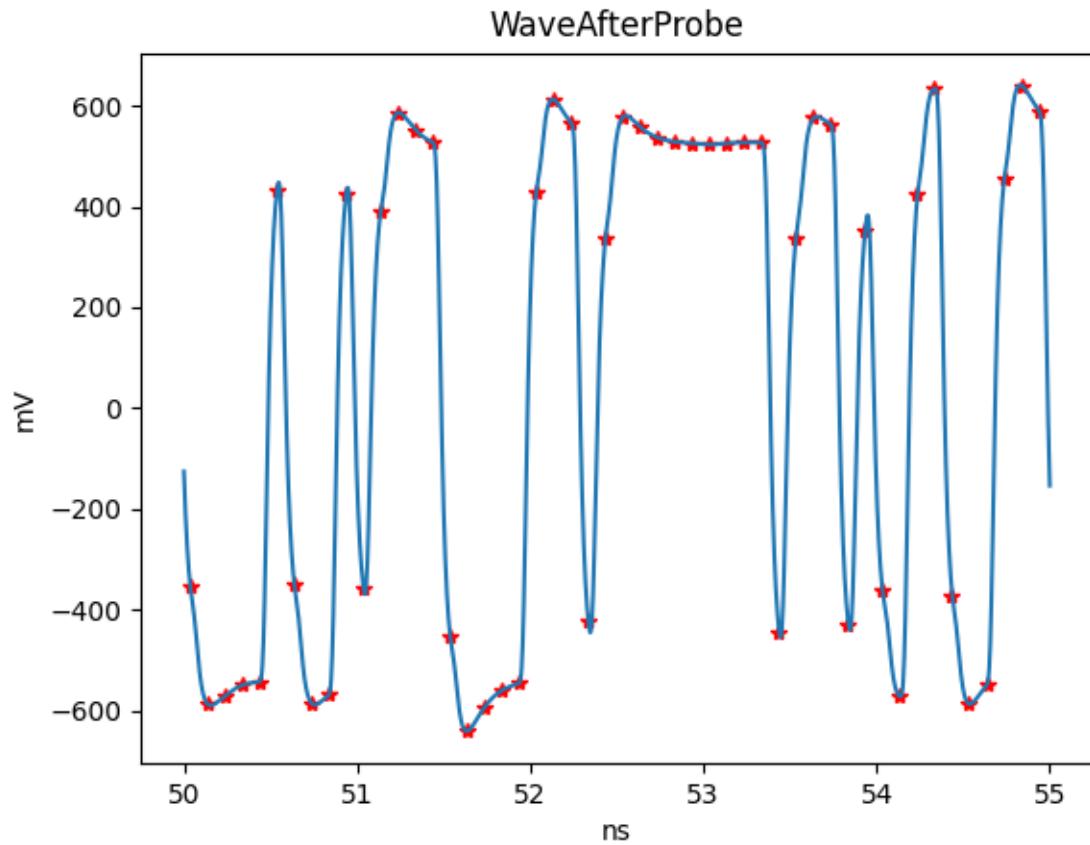
```
tstop = 55e-9
tstart = 50e-9
scale_time = pyaedt.constants.unit_converter(1, unit_system="Time", input_units="s",
 output_units=original_data.units_sweeps[
 "Time"])
scale_data = pyaedt.constants.unit_converter(1, unit_system="Voltage", input_units="V",
 output_units=original_data.units_data[plot_
 name])

tstop_ns = scale_time * tstop
tstart_ns = scale_time * tstart

for time in original_data_value[plot_name].index:
 if tstart_ns <= time[0]:
 start_index_original_data = time[0]
 break
for time in original_data_value[plot_name][start_index_original_data:].index:
 if time[0] >= tstop_ns:
 stop_index_original_data = time[0]
 break
for time in sample_waveform[0].index:
 if tstart <= time:
 sample_index = sample_waveform[0].index == time
 start_index_waveform = sample_index.tolist().index(True)
 break
for time in sample_waveform[0].index:
 if time >= tstop:
 sample_index = sample_waveform[0].index == time
 stop_index_waveform = sample_index.tolist().index(True)
 break

original_data_zoom = original_data_value[start_index_original_data:stop_index_original_
 _data]
sampled_data_zoom = sample_waveform[0].values[start_index_waveform:stop_index_waveform] *
 scale_data
sampled_time_zoom = sample_waveform[0].index[start_index_waveform:stop_index_waveform] *
 scale_time

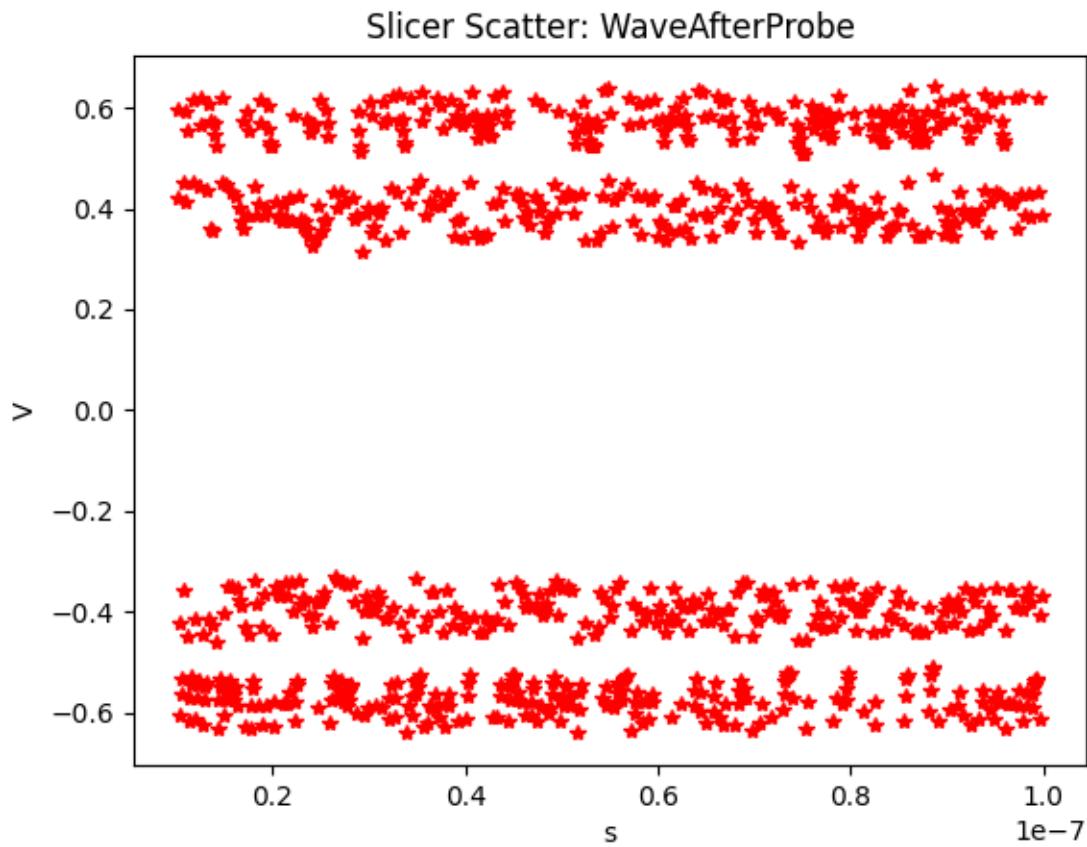
fig, ax = plt.subplots()
ax.plot(sampled_time_zoom, sampled_data_zoom, "r*")
ax.plot(np.array(list(original_data_zoom.index.values)), original_data_zoom.values)
ax.set_title('WaveAfterProbe')
ax.set_xlabel(original_data.units_sweeps["Time"])
ax.set_ylabel(original_data.units_data[plot_name])
plt.show()
```



## Plot Slicer Scatter

Create the plot from a start time to stop time in seconds

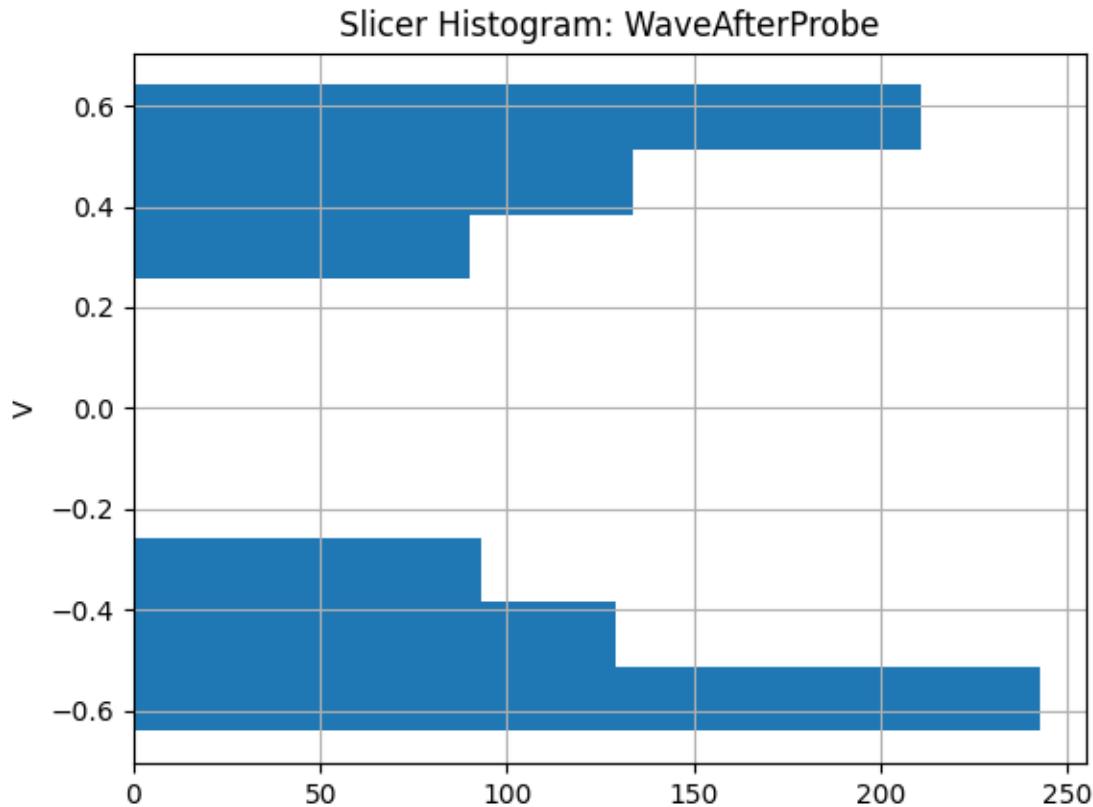
```
fig, ax2 = plt.subplots()
ax2.plot(sample_waveform[0].index, sample_waveform[0].values, "r*")
ax2.set_title('Slicer Scatter: WaveAfterProbe')
ax2.set_xlabel("s")
ax2.set_ylabel("V")
plt.show()
```



### Plot scatter histogram

Create the plot from a start time to stop time in seconds.

```
fig, ax4 = plt.subplots()
ax4.set_title('Slicer Histogram: WaveAfterProbe')
ax4.hist(sample_waveform[0].values, orientation='horizontal')
ax4.set_ylabel("V")
ax4.grid()
plt.show()
```



## Get Transient report

Get Transient report data

```
plot_name = "V(b_input_43.int_ami_rx.eye_probe.out)"
cir.solution_type = "NexximTransient"
original_data = cir.post.get_solution_data(expressions=plot_name,
 domain="Time",
 setup_sweep_name="NexximTransient",
 variations=cir.available_variations.nominal)
```

## Sample waveform using a user-defined clock

Extract waveform at specific clock time plus half unit interval.

```
original_data.enable_pandas_output = False
original_data_value = original_data.data_real()
original_data_sweep = original_data.primary_sweep_values
waveform_unit = original_data.units_data[plot_name]
waveform_sweep_unit = original_data.units_sweeps["Time"]
tics = np.arange(20e-9, 100e-9, 1e-10, dtype=float)
```

(continues on next page)

(continued from previous page)

```
sample_waveform = cir.post.sample_waveform(
 waveform_data=original_data_value,
 waveform_sweep=original_data_sweep,
 waveform_unit=waveform_unit,
 waveform_sweep_unit=waveform_sweep_unit,
 unit_interval=unit_interval,
 clock_tics=ticks,
 pandas_enabled=False,
)

```

## Plot waveform and samples

Create the plot from a start time to stop time in seconds.

```
tstop = 40.0e-9
tstart = 25.0e-9
scale_time = pyaedt.constants.unit_converter(1, unit_system="Time", input_units="s",
 output_units=waveform_sweep_unit)
scale_data = pyaedt.constants.unit_converter(1, unit_system="Voltage", input_units="V",
 output_units=waveform_unit)

tstop_ns = scale_time * tstop
tstart_ns = scale_time * tstart

for time in original_data_sweep:
 if tstart_ns <= time:
 start_index_original_data = original_data_sweep.index(time)
 break
for time in original_data_sweep[start_index_original_data:]:
 if time >= tstop_ns:
 stop_index_original_data = original_data_sweep.index(time)
 break
cont = 0
for frame in sample_waveform:
 if tstart <= frame[0]:
 start_index_waveform = cont
 break
 cont += 1
for frame in sample_waveform[start_index_waveform:]:
 if frame[0] >= tstop:
 stop_index_waveform = cont
 break
 cont += 1

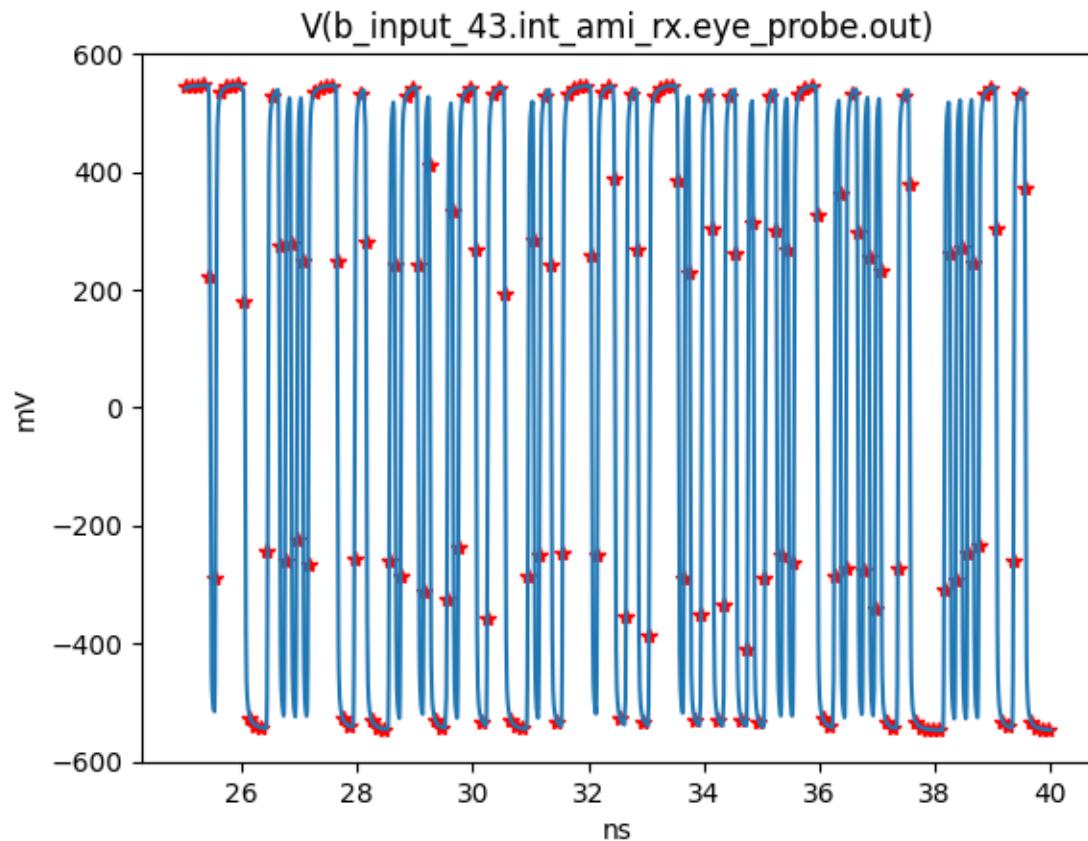
original_data_zoom = original_data_value[start_index_original_data:stop_index_original_
 -data]
original_sweep_zoom = original_data_sweep[start_index_original_data:stop_index_original_
 -data]
original_data_zoom_array = np.array(list(map(list, zip(original_sweep_zoom, original_
 -data_zoom))))
```

(continues on next page)

(continued from previous page)

```
original_data_zoom_array[:, 0] *= 1
sampled_data_zoom_array = np.array(sample_waveform[start_index_waveform:stop_index_
waveform])
sampled_data_zoom_array[:, 0] *= scale_time
sampled_data_zoom_array[:, 1] *= scale_data

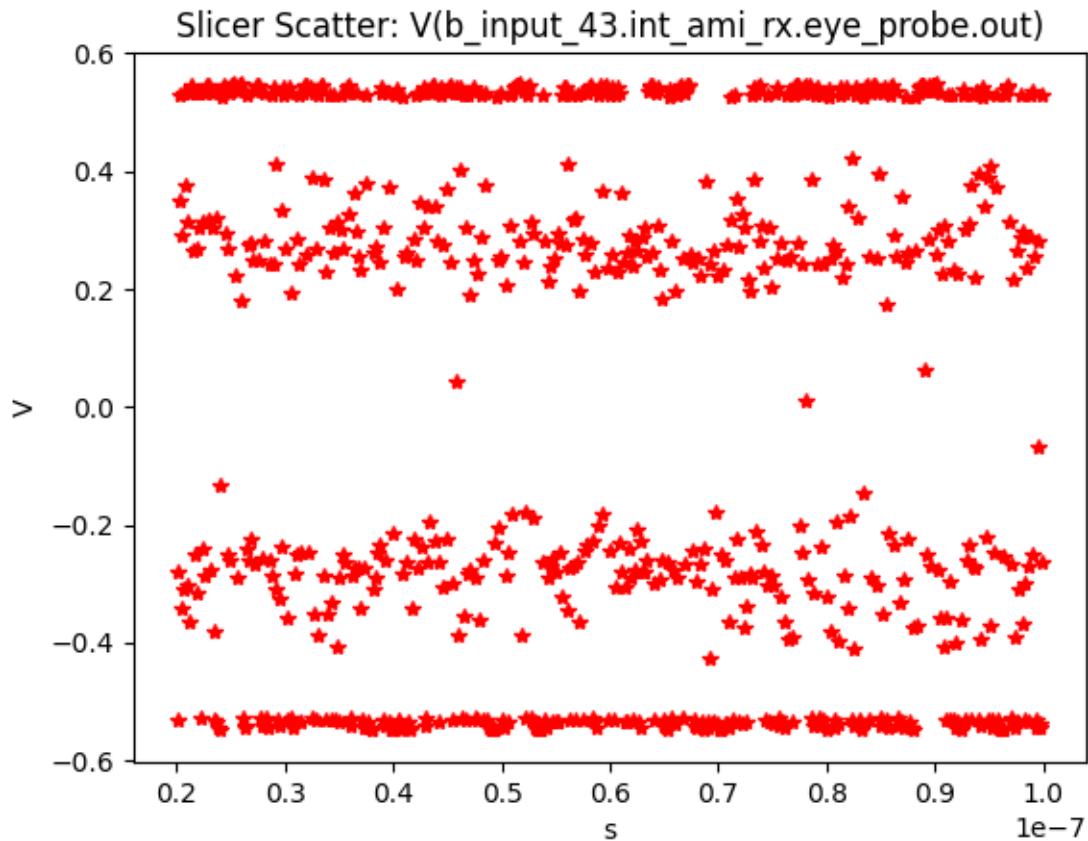
fig, ax = plt.subplots()
ax.plot(sampled_data_zoom_array[:, 0], sampled_data_zoom_array[:, 1], "r*")
ax.plot(original_sweep_zoom, original_data_zoom_array[:, 1])
ax.set_title(plot_name)
ax.set_xlabel(waveform_sweep_unit)
ax.set_ylabel(waveform_unit)
plt.show()
```



## Plot slicer scatter

Create the plot from a start time to stop time in seconds.

```
sample_waveform_array = np.array(sample_waveform)
fig, ax2 = plt.subplots()
ax2.plot(sample_waveform_array[:, 0], sample_waveform_array[:, 1], "r*")
ax2.set_title('Slicer Scatter: ' + plot_name)
ax2.set_xlabel("s")
ax2.set_ylabel("V")
plt.show()
```



## Save project and close AEDT

Save the project and close AEDT.

```
cir.save_project()
print("Project Saved in {}".format(cir.project_path))
cir.release_desktop()
```

Project Saved in D:/Temp/pyaedt\_prj\_N1Z/ami/

(continues on next page)

(continued from previous page)

True
------

**Total running time of the script:** (1 minutes 17.016 seconds)

### Circuit: schematic creation and analysis

This example shows how you can use PyAEDT to create a circuit design and run a Nexxim time-domain simulation.

#### Perform required imports

Perform required imports.

```
sphinx_gallery_thumbnail_path = 'Resources/circuit.png'

import pyaedt
```

#### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

#### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`. The Boolean parameter `new_thread` defines whether to create a new instance of AEDT or try to connect to an existing instance of it.

```
non_graphical = False
new_thread = True
```

#### Launch AEDT and Circuit

Launch AEDT and Circuit. The `pyaedt.Desktop` class initializes AEDT and starts the specified version in the specified mode.

```
desktop = pyaedt.launch_desktop(aedt_version, non_graphical, new_thread)
aedt_app = pyaedt.Circuit(projectname=pyaedt.generate_unique_project_name())
aedt_app.modeler.schematic.schematic_units = "mil"
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
↳ subprocess 11096 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 ↳ py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 ↳ log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create circuit setup

Create and customize an LNA (linear network analysis) setup.

```
setup1 = aedt_app.create_setup("MyLNA")
setup1.props["SweepDefinition"]["Data"] = "LINC 0GHz 4GHz 10001"
```

## Create components

Create components, such as an inductor, resistor, and capacitor.

```
inductor = aedt_app.modeler.schematic.create_inductor(name="L1", value=1e-9, location=[0,
 ↵ 0])
resistor = aedt_app.modeler.schematic.create_resistor(name="R1", value=50, location=[500,
 ↵ 0])
capacitor = aedt_app.modeler.schematic.create_capacitor(name="C1", value=1e-12, ↵
 ↵location=[1000, 0])
```

## Get all pins

Get all pins of a specified component.

```
pins_resistor = resistor.pins
```

## Create port and ground

Create a port and a ground, which are needed for the circuit analysis.

```
port = aedt_app.modeler.components.create_interface_port(name="myport", location=[-200,
 ↵ 0])
gnd = aedt_app.modeler.components.create_gnd(location=[1200, -100])
```

## Connect components

Connect components with wires.

```
port.pins[0].connect_to_component(assignment=inductor.pins[0], use_wire=True)
inductor.pins[1].connect_to_component(assignment=resistor.pins[1], use_wire=True)
resistor.pins[0].connect_to_component(assignment=capacitor.pins[0], use_wire=True)
capacitor.pins[1].connect_to_component(assignment=gnd.pins[0], use_wire=True)
```

```
True
```

## Create transient setup

Create a transient setup.

```
setup2 = aedt_app.create_setup(name="MyTransient", setup_type=aedt_app.SETUPS.
 ↪NexximTransient)
setup2.props["TransientData"] = ["0.01ns", "200ns"]
setup3 = aedt_app.create_setup(name="MyDC", setup_type=aedt_app.SETUPS.NexximDC)
```

## Solve transient setup

Solve the transient setup.

```
aedt_app.analyze_setup("MyLNA")
aedt_app.export_fullwave_spice()
```

```
'D:/Temp/pyaedt_prj_8US/Project_6HB.pyaedt\\Circuit_Design_UFD\\Circuit Design_UFD.sp'
```

## Create report

Create a report that plots solution data.

```
solutions = aedt_app.post.get_solution_data(expressions=aedt_app.get_traces_for_
 ↪plot(category="S"))
solutions.enable_pandas_output = True
real, imag = solutions.full_matrix_real_imag
print(real)
```

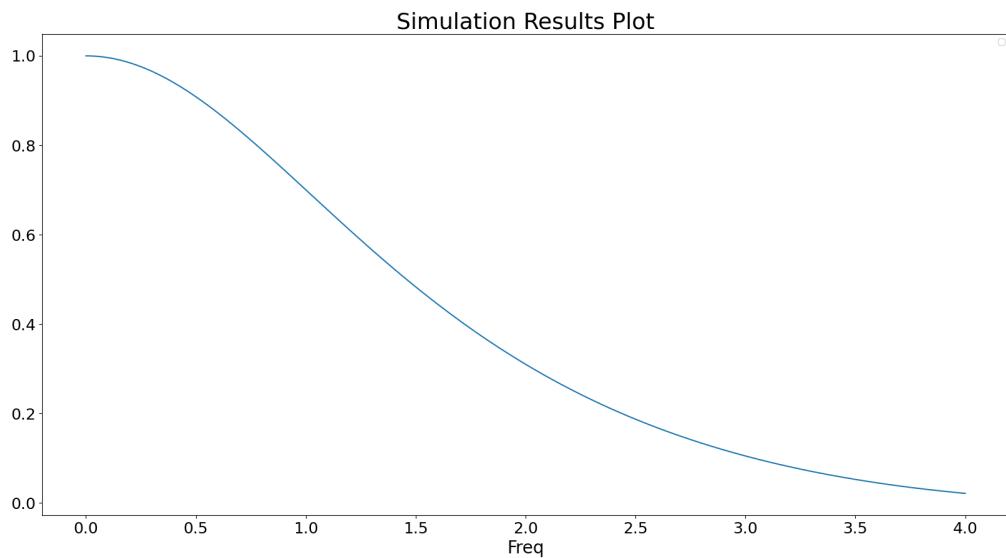
```
S(myport,myport)
0.0000 1.000000
0.0004 1.000000
0.0008 1.000000
0.0012 0.999999
0.0016 0.999999
... ...
3.9984 0.021101
3.9988 0.021083
3.9992 0.021065
3.9996 0.021046
4.0000 0.021028

[10001 rows x 1 columns]
```

## Plot data

Create a plot based on solution data.

```
fig = solutions.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

## Close AEDT

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

```
desktop.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 15.569 seconds)

## Circuit: Simulate multi-zones layout with Siwave

This example shows how you can use PyAEDT simulate multi-zones with Siwave.

## Perform required imports

Perform required imports, which includes importing a section.

```
from pyaedt import Edb, Circuit
import os.path
import pyaedt
```

## Download file

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

```
temp_folder = pyaedt.generate_unique_folder_name()
edb_file = pyaedt.downloads.download_file(destination=temp_folder, directory="edb/siwave_
˓→multi_zones.aedb")
working_directory = os.path.join(temp_folder, "workdir")
aedt_file = os.path.splitext(edb_file)[0] + ".aedt"
circuit_project_file = os.path.join(working_directory, os.path.splitext(os.path.
˓→basename(edb_file))[0] +
 "multizone_clipped_circuit.aedt")
print(edb_file)
```

```
D:\Temp\pyaedt_prj_8CL\edb/siwave_multi_zones.aedb
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Ground net

Common reference net used across all sub-designs, Mandatory for this work flow.

```
common_reference_net = "GND"
```

## Project load

Load initial Edb file, checking if aedt file exists and remove to allow Edb loading.

```
if os.path.isfile(aedt_file):
 os.remove(aedt_file)
edb = Edb(edbversion=aedt_version, edbpath=edb_file)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyedb\dotnet\edb_core\
˓→components.py:185: DeprecationWarning: Use new property :func:`instances` instead.
 warnings.warn("Use new property :func:`instances` instead.", DeprecationWarning)
```

## Project zones

Copy project zone into sub project.

```
edb_zones = edb.copy_zones(working_directory=working_directory)
```

## Split zones

Clip sub-designs along with corresponding zone definition and create port of clipped signal traces.

```
defined_ports, project_connexions = edb.cutout_multizone_layout(edb_zones, common_
↳ reference_net)
```

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyedb\dotnet\edb_core\
↳ stackup.py:636: DeprecationWarning: `stackup_mode` is deprecated. Use `mode` method
↳ instead.
 warnings.warn("`stackup_mode` is deprecated. Use `mode` method instead.",_
↳ DeprecationWarning)
```

## Circuit

Create circuit design, import all sub-project as EM model and connect all corresponding pins in circuit.

```
circuit = Circuit(specified_version=aedt_version, projectname=circuit_project_file)
circuit.connect_circuit_models_from_multi_zone_cutout(project_connections=project_
↳ connexions,
 edb_zones_dict=edb_zones,_
↳ ports=defined_ports,
 model_inc=70)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
↳ subprocess 2348 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
↳ py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
↳ log' mode='a' encoding='cp1252'>
 self._logger = val
```

True

## Setup

Add Nexxim LNA simulation setup.

```
circuit_setup= circuit.create_setup("Pyedt_LNA")
```

## Frequency sweep

Add frequency sweep from 0GHz to 20GHz with 10MHz frequency step.

```
circuit_setup.props["SweepDefinition"]["Data"] = "LIN {} {} {}".format("0GHz", "20GHz",
 "10MHz")
```

## Start simulation

Analyze all siwave projects and solves the circuit.

```
circuit.analyze()
```

```
True
```

## Define differential pairs

```
circuit.set_differential_pair(assignment="U0.via_38.B2B_SIGP", reference="U0.via_39.B2B_
 SIGN", differential_mode="U0")
circuit.set_differential_pair(assignment="U1.via_32.B2B_SIGP", reference="U1.via_33.B2B_
 SIGN", differential_mode="U1")
```

```
True
```

## Plot results

```
circuit.post.create_report(expressions=["dB(S(U0,U0))", "dB(S(U1,U0))"], context=
 "Differential Pairs")
```

```
<pyaedt.modules.report_templates.Standard object at 0x000002591EE8F670>
```

## Release AEDT desktop

```
circuit.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 47.091 seconds)

## Circuit: schematic subcircuit management

This example shows how you can use PyAEDT to add a subcircuit to a circuit design. It pushes down the child subcircuit and pops up to the parent design.

### Perform required import

Perform the required import.

```
import os
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`.

```
non_graphical = False
```

### Launch AEDT with Circuit

Launch AEDT 2023 R2 in graphical mode with Circuit.

```
circuit = pyaedt.Circuit(projectname=pyaedt.generate_unique_project_name(),
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=True
)
circuit.modeler.schematic_units = "mil"
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 3316 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 _py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 _log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Add subcircuit

Add a new subcircuit to the previously created circuit design, creating a child circuit. Push this child circuit down into the child subcircuit.

```
subcircuit = circuit.modeler.schematic.create_subcircuit(location=[0.0, 0.0])
subcircuit_name = subcircuit.composed_name
circuit.push_down(subcircuit)
```

```
True
```

## Parametrize subcircuit

Parametrize the subcircuit and add a resistor, inductor, and a capacitor with the parameter values in the following code example. Connect them in series and then use the `pop_up #` method to get back to the parent design.

```
circuit.variable_manager.set_variable(variable_name="R_val", expression="35ohm")
circuit.variable_manager.set_variable(variable_name="L_val", expression="1e-7H")
circuit.variable_manager.set_variable(variable_name="C_val", expression="5e-10F")
p1 = circuit.modeler.schematic.create_interface_port(name="In")
r1 = circuit.modeler.schematic.create_resistor(value="R_val")
l1 = circuit.modeler.schematic.create_inductor(value="L_val")
c1 = circuit.modeler.schematic.create_capacitor(value="C_val")
p2 = circuit.modeler.schematic.create_interface_port(name="Out")
circuit.modeler.schematic.connect_components_in_series(assignment=[p1, r1, l1, c1, p2], ↵
 use_wire=True)
circuit.pop_up()
```

```
True
```

## Release AEDT

Release AEDT.

```
circuit.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (0 minutes 36.465 seconds)

## Circuit: transient analysis and eye plot

This example shows how you can use PyAEDT to create a circuit design, run a Nexxim time-domain simulation, and create an eye diagram.

## Perform required imports

Perform required imports.

```
import os
from matplotlib import pyplot as plt
import numpy as np
import pyaedt
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode, "PYAEDT\_NON\_GRAPHICAL" is needed to generate documentation only. You can set `non_graphical` either to True or False.

```
non_graphical = False
```

## Launch AEDT with Circuit

Launch AEDT 2023 R2 in graphical mode with Circuit.

```
cir = pyaedt.Circuit(projectname=pyaedt.generate_unique_project_name(),
 specified_version=aedt_version,
 new_desktop_session=True,
 non_graphical=non_graphical
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 10908 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Read IBIS file

Read an IBIS file and place a buffer in the schematic.

```
ibis = cir.get_ibis_model_from_file(os.path.join(cir.desktop_install_dir, 'buflib', 'IBIS
↪', 'u26a_800.ibs'))
ibs = ibis.buffers["DQ_u26a_800"].insert(0, 0)
```

## Place ideal transmission line

Place an ideal transmission line in the schematic and parametrize it.

```
tr1 = cir.modeler.components.components_catalog["Ideal Distributed:TRLK_NX"].place("tr1")
tr1.parameters["P"] = "50mm"
```

## Create resistor and ground

Create a resistor and ground in the schematic.

```
res = cir.modeler.components.create_resistor(name="R1", value="1Meg")
gnd1 = cir.modeler.components.create_gnd()
```

## Connect elements

Connect elements in the schematic.

```
tr1.pins[0].connect_to_component(ibs.pins[0])
tr1.pins[1].connect_to_component(res.pins[0])
res.pins[1].connect_to_component(gnd1.pins[0])
```

```
(True, <pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object at
↪0x00000258F25683A0>, <pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object
↪at 0x00000258F25697B0>)
```

## Place probe

Place a probe and rename it to Vout.

```
pr1 = cir.modeler.components.components_catalog["Probes:VPROBE"].place("vout")
pr1.parameters["Name"] = "Vout"
pr1.pins[0].connect_to_component(res.pins[0])
pr2 = cir.modeler.components.components_catalog["Probes:VPROBE"].place("Vin")
pr2.parameters["Name"] = "Vin"
pr2.pins[0].connect_to_component(ibs.pins[0])
```

```
(True, <pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object at
↪0x000002591EE8D2A0>, <pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object
↪at 0x000002591EE8FB20>)
```

## Create setup and analyze

Create a transient analysis setup and analyze it.

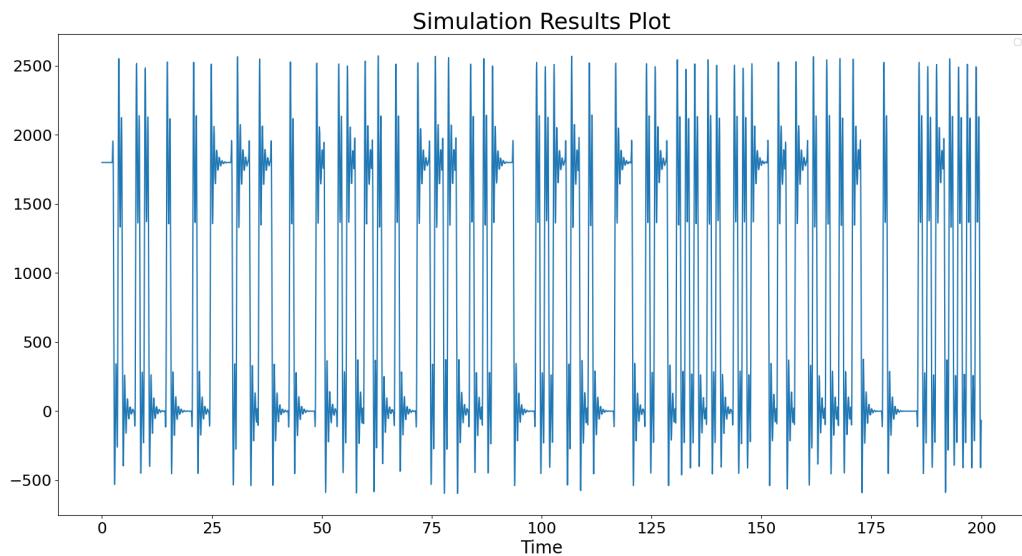
```
trans_setup = cir.create_setup(name="TransientRun", setup_type="NexximTransient")
trans_setup.props["TransientData"] = ["0.01ns", "200ns"]
cir.analyze_setup("TransientRun")
```

True

## Create report outside AEDT

Create a report outside AEDT using the `get_solution_data` method. This method allows you to get solution data and plot it outside AEDT without needing a UI.

```
report = cir.post.create_report("V(Vout)", domain="Time")
if not non_graphical:
 report.add_cartesian_y_marker(0)
solutions = cir.post.get_solution_data(domain="Time")
solutions.plot("V(Vout)")
```



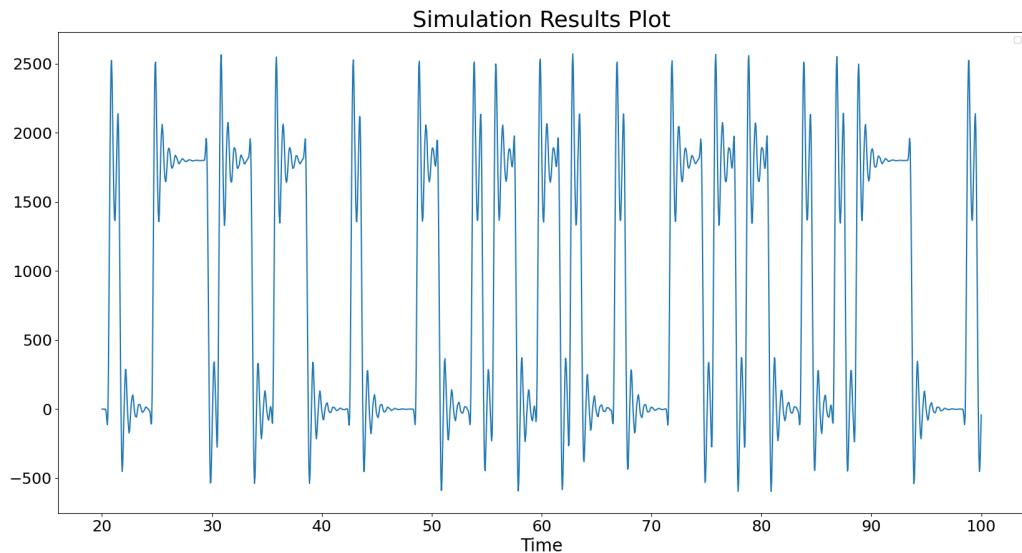
No artists with labels found to put in legend. Note that artists whose label start with `_` an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Create report inside AEDT

Create a report inside AEDT using the `new_report` object. This object is fully customizable and usable with most of the reports available in AEDT. The standard report is the main one used in Circuit and Twin Builder.

```
new_report = cir.post.reports_by_category.standard("V(Vout)")
new_report.domain = "Time"
new_report.create()
if not non_graphical:
 new_report.add_limit_line_from_points([60, 80], [1, 1], "ns", "V")
 vout = new_report.traces[0]
 vout.set_trace_properties(trace_style=vout.LINESTYLE.Dot, width=2, trace_type=vout.
 ↪TRACETYPE.Continuous,
 color=(0, 0, 255))
 vout.set_symbol_properties(style=vout.SYMBOLSTYLE.Circle, fill=True, color=(255, 255,
 ↪0))
 ll = new_report.limit_lines[0]
 ll.set_line_properties(style=ll.LINESTYLE.Solid, width=4, hatch_above=True, ↪
 ↪violation_emphasis=True, hatch_pixels=2,
 color=(0, 0, 255))
new_report.time_start = "20ns"
new_report.time_stop = "100ns"
new_report.create()
sol = new_report.get_solution_data()
sol.plot()
```



No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when `legend()` is called with no argument.

<Figure size 2000x1000 with 1 Axes>

## Create eye diagram inside AEDT

Create an eye diagram inside AEDT using the `new_eye` object.

```
new_eye = cir.post.reports_by_category.eye_diagram("V(Vout)")
new_eye.unit_interval = "1e-9s"
new_eye.time_stop = "100ns"
new_eye.create()
```

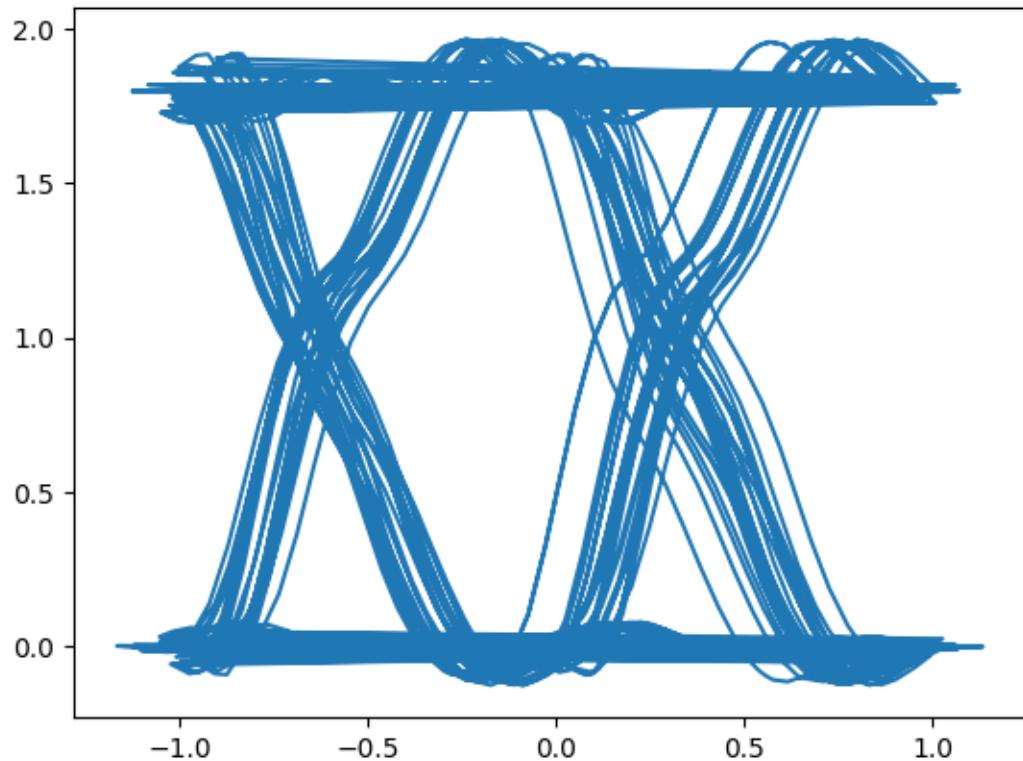
True

## Create eye diagram outside AEDT

Create the same eye diagram outside AEDT using Matplotlib and the `get_solution_data` method.

```
unit_interval = 1
offset = 0.25
tstop = 200
tstart = 0
t_steps = []
i = tstart + offset
while i < tstop:
 i += 2 * unit_interval
 t_steps.append(i)

t = [[i for i in solutions.intrinsics["Time"] if k - 2 * unit_interval < i <= k] for k in t_steps]
ys = [[i / 1000 for i, j in zip(solutions.data_real(), solutions.intrinsics["Time"]) if
 k - 2 * unit_interval < j <= k] for k in t_steps]
fig, ax = plt.subplots(sharesx=True)
cellst = np.array([])
cellsv = np.array([])
for a, b in zip(t, ys):
 an = np.array(a)
 an = an - an.mean()
 bn = np.array(b)
 cellst = np.append(cellst, an)
 cellsv = np.append(cellsv, bn)
plt.plot(cellst.T, cellsv.T, zorder=0)
plt.show()
```



## Release AEDT

Release AEDT.

```
cir.save_project()
cir.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 8.834 seconds)

## Circuit: netlist to schematic import

This example shows how you can import netlist data into a circuit design. HSPICE files are fully supported. Mentor files are partially supported.

## Perform required imports

Perform required imports and set paths.

```
import os

import pyaedt

netlist = pyaedt.downloads.download_netlist()

project_name = pyaedt.generate_unique_project_name()
print(project_name)
```

D:\Temp\pyaedt\_prj\_K2L\Project\_1E9.aedt

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`. The Boolean parameter `NewThread` defines whether to create a new instance of AEDT or try to connect to an existing instance of it.

```
non_graphical = False
NewThread = True
```

## Launch AEDT with Circuit

Launch AEDT with Circuit. The `pyaedt.Desktop` class initializes AEDT and starts it on the specified version in the specified graphical mode.

```
desktop = pyaedt.launch_desktop(aedt_version, non_graphical, NewThread)
aedtapp = pyaedt.Circuit(projectname=project_name)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: `subprocess` 3728 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Define variable

Define a design variable by using a \$ prefix.

```
aedtapp["Voltage"] = "5"
```

## Create schematic from netlist file

Create a schematic from a netlist file. The `create_schematic_from_netlist` method reads the netlist file and parses it. All components are parsed but only these categories are mapped: R, L, C, Q, U, J, V, and I.

```
aedtapp.create_schematic_from_netlist(netlist)
```

```
['NAME:Models_Netlist', 'Info:=', ['Type:=', 0, 'NumTerminals:=', 0, 'DataSource:=', '',
 'ModifiedOn:=', '', 'Manufacturer:=', '', 'Symbol:=', 'Models_Netlist', 'ModelNames:=',
 '', 'Footprint:=', '', 'Description:=', '', 'InfoTopic:=', '', 'InfoHelpFile:=', '',
 'IconFile:=', '', 'Library:=', '', 'OriginalLocation:=', 'Project', 'IEEE:=', '',
 'Author:=', '', 'OriginalAuthor:=', '', 'CreationDate:=', '', 'ExampleFile:=', '',
 'HiddenComponent:=', 0, 'CircuitEnv:=', 0, 'GroupID:=', 0], 'CircuitEnv:=', 0,
 'Refbase:=', 'x', 'NumParts:=', 1, 'ModSinceLib:=', True, 'CompExtID:=', 1, [
 'NAME:Parameters', 'ButtonProp:=', ['CosimDefinition', 'D', '', 'Edit', 'Edit', 40501,
 'ButtonPropClientData:=', []], 'MenuProp:=', ['CoSimulator', 'D', '', 'DefaultNetlist',
 0]], ['NAME:CosimDefinitions', ['NAME:CosimDefinition', 'CosimulatorType:=', 4,
 'CosimDefName:=', 'DefaultNetlist', 'IsDefinition:=', True, 'Connect:=', True, 'Data:=',
 ['Nexxim Circuit:=', 'x@ID '], 'GRef:=', ['Nexxim Circuit:=', '']], 'DefaultCosim:=',
 'DefaultNetlist']]
['NAME:STS6NF20V', 'Info:=', ['Type:=', 0, 'NumTerminals:=', 8, 'DataSource:=', '',
 'ModifiedOn:=', 1591858313, 'Manufacturer:=', '', 'Symbol:=', 'STS6NF20V',
 'ModelNames:=', '', 'Footprint:=', '', 'Description:=', '', 'InfoTopic:=', '',
 'InfoHelpFile:=', '', 'IconFile:=', '', 'Library:=', '', 'OriginalLocation:=', 'Project',
 'IEEE:=', '', 'Author:=', '', 'OriginalAuthor:=', '', 'CreationDate:=', 1591858313,
 'ExampleFile:=', '', 'HiddenComponent:=', 0, 'CircuitEnv:=', 0, 'GroupID:=', 0],
 'CircuitEnv:=', 0, 'Refbase:=', 'Q', 'NumParts:=', 1, 'ModSinceLib:=', True,
 'Terminal:=', ['Pin1', 'Pin1', 'A', False, 0, 1, '', 'Electrical', '0'], 'Terminal:=',
 ['Pin2', 'Pin2', 'A', False, 0, 1, '', 'Electrical', '0'], 'Terminal:=', ['Pin3', 'Pin3',
 'A', False, 0, 1, '', 'Electrical', '0'], 'Terminal:=', ['Pin4', 'Pin4', 'A', False,
 0, 1, '', 'Electrical', '0'], 'Terminal:=', ['Pin5', 'Pin5', 'A', False, 0, 1, '',
 'Electrical', '0'], 'Terminal:=', ['Pin6', 'Pin6', 'A', False, 0, 1, '', 'Electrical',
 '0'], 'Terminal:=', ['Pin7', 'Pin7', 'A', False, 0, 1, '', 'Electrical', '0'],
 'Terminal:=', ['Pin8', 'Pin8', 'A', False, 0, 1, '', 'Electrical', '0'], 'CompExtID:=',
 1, ['NAME:Parameters', 'TextValueProp:=', ['MOD', 'D', '', 'STS6NF20V'], 'ButtonProp:=',
 ['CosimDefinition', 'D', '', 'Edit', 'Edit', 40501, 'ButtonPropClientData:=', []],
 'MenuProp:=', ['CoSimulator', 'D', '', 'DefaultNetlist', 0]], ['NAME:CosimDefinitions',
 ['NAME:CosimDefinition', 'CosimulatorType:=', 4, 'CosimDefName:=', 'DefaultNetlist',
 'IsDefinition:=', True, 'Connect:=', True, 'Data:=', ['Nexxim Circuit:=', 'Q@ID %0
 STS6NF20V %1 STS6NF20V %2 STS6NF20V %3 STS6NF20V %4 STS6NF20V %5 STS6NF20V %6
 STS6NF20V %7 STS6NF20V @MOD '], 'GRef:=', ['Nexxim Circuit:=', '']], 'DefaultCosim:=',
 'DefaultNetlist']]
['NAME:CON4', 'Info:=', ['Type:=', 0, 'NumTerminals:=', 5, 'DataSource:=', '',
 'ModifiedOn:=', 1591858313, 'Manufacturer:=', '', 'Symbol:=', 'CON4', 'ModelNames:=', '
 ', 'Footprint:=', '', 'Description:=', '', 'InfoTopic:=', '', 'InfoHelpFile:=', '',
 'IconFile:=', '', 'Library:=', '', 'OriginalLocation:=', 'Project', 'IEEE:=', '']
```

(continues on next page)

(continued from previous page)

```

→ 'Author:='', '', 'OriginalAuthor:='', '', 'CreationDate:=' , 1591858313, 'ExampleFile:=' ,
→ '', 'HiddenComponent:=' , 0, 'CircuitEnv:=' , 0, 'GroupID:=' , 0], 'CircuitEnv:=' , 0,
→ 'Refbase:=' , 'J', 'NumParts:=' , 1, 'ModSinceLib:=' , True, 'Terminal:=' , ['Pin1', 'Pin1
→ ', 'A', False, 0, 1, '' , 'Electrical', '0'], 'Terminal:=' , ['Pin2', 'Pin2', 'A', False,
→ 0, 1, '' , 'Electrical', '0'], 'Terminal:=' , ['Pin3', 'Pin3', 'A', False, 0, 1, '' ,
→ 'Electrical', '0'], 'Terminal:=' , ['Pin4', 'Pin4', 'A', False, 0, 1, '' , 'Electrical',
→ '0'], 'Terminal:=' , ['Pin5', 'Pin5', 'A', False, 0, 1, '' , 'Electrical', '0'],
→ 'CompExtID:=' , 1, ['NAME:Parameters', 'TextValueProp:=' , ['MOD', 'D', '' , 'CON4'],
→ 'ButtonProp:=' , ['CosimDefinition', 'D', '' , 'Edit', 'Edit', 40501,
→ 'ButtonPropClientData:=' , []], 'MenuProp:=' , ['CoSimulator', 'D', '' , 'DefaultNetlist',
→ 0]], ['NAME:CosimDefinitions', ['NAME:CosimDefinition', 'CosimulatorType:=' , 4,
→ 'CosimDefName:=' , 'DefaultNetlist', 'IsDefinition:=' , True, 'Connect:=' , True, 'Data:='
→ , ['Nexxim Circuit:=' , 'J@ID %0 CON4 %1 CON4 %2 CON4 %3 CON4 %4 CON4 @MOD '], 'GRef:='
→ , ['Nexxim Circuit:=' , '']], 'DefaultCosim:=' , 'DefaultNetlist']]]

['NAME:DCH010505SN7', 'Info:=' , ['Type:=' , 0, 'NumTerminals:=' , 4, 'DataSource:=' , '' ,
→ 'ModifiedOn:=' , 1591858313, 'Manufacturer:=' , '' , 'Symbol:=' , 'DCH010505SN7',
→ 'ModelNames:=' , '' , 'Footprint:=' , '' , 'Description:=' , '' , 'InfoTopic:=' , '' ,
→ 'InfoHelpFile:=' , '' , 'IconFile:=' , '' , 'Library:=' , '' , 'OriginalLocation:=' , 'Project
→ ' , 'IEEE:=' , '' , 'Author:=' , '' , 'OriginalAuthor:=' , '' , 'CreationDate:=' , 1591858313,
→ 'ExampleFile:=' , '' , 'HiddenComponent:=' , 0, 'CircuitEnv:=' , 0, 'GroupID:=' , 0],
→ 'CircuitEnv:=' , 0, 'Refbase:=' , 'U', 'NumParts:=' , 1, 'ModSinceLib:=' , True,
→ 'Terminal:=' , ['Pin1', 'Pin1', 'A', False, 0, 1, '' , 'Electrical', '0'], 'Terminal:=' ,
→ ['Pin2', 'Pin2', 'A', False, 0, 1, '' , 'Electrical', '0'], 'Terminal:=' , ['Pin3', 'Pin3
→ ', 'A', False, 0, 1, '' , 'Electrical', '0'], 'Terminal:=' , ['Pin4', 'Pin4', 'A', False,
→ 0, 1, '' , 'Electrical', '0'], 'CompExtID:=' , 1, ['NAME:Parameters', 'TextValueProp:=' ,
→ ['MOD', 'D', '' , 'DCH010505SN7'], 'ButtonProp:=' , ['CosimDefinition', 'D', '' , 'Edit',
→ 'Edit', 40501, 'ButtonPropClientData:=' , []], 'MenuProp:=' , ['CoSimulator', 'D', '' ,
→ 'DefaultNetlist', 0]], ['NAME:CosimDefinitions', ['NAME:CosimDefinition',
→ 'CosimulatorType:=' , 4, 'CosimDefName:=' , 'DefaultNetlist', 'IsDefinition:=' , True,
→ 'Connect:=' , True, 'Data:=' , ['Nexxim Circuit:=' , 'U@ID %0 DCH010505SN7 %1
→ DCH010505SN7 %2 DCH010505SN7 %3 DCH010505SN7 @MOD '], 'GRef:=' , ['Nexxim Circuit:=' ,
→ '']], 'DefaultCosim:=' , 'DefaultNetlist']]]

True

```

## Close project and release AEDT

After adding any other desired functionalities, close the project and release AEDT.

```
desktop.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 35.047 seconds)

## Circuit: automatic report creation

This example shows how you can use PyAEDT to create reports automatically using a JSON file.

### Perform required imports

Perform required imports and set the local path to the path for PyAEDT.

```
import os
from IPython.display import Image
import pyaedt

Set local path to path for PyAEDT
temp_folder = pyaedt.generate_unique_folder_name()
project_path = pyaedt.downloads.download_custom_reports(destination=temp_folder)
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to True or False. The Boolean parameter `new_thread` defines whether to create a new instance of AEDT or try to connect to an existing instance of it.

```
non_graphical = True
NewThread = True
```

### Launch AEDT with Circuit

Launch AEDT with Circuit. The `pyaedt.Desktop` class initializes AEDT and starts the specified version in the specified mode.

```
cir = pyaedt.Circuit(projectname=os.path.join(project_path, 'CISPR25_Radiated_Emissions_'
 ↪Example23R1.aedtz'),
 non_graphical=non_graphical,
 specified_version=aedt_version,
 new_desktop_session=True
)
cir.analyze()
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: ↪
 subprocess 8396 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 ↪py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 (continues on next page)
```

(continued from previous page)

```
 ↪log' mode='a' encoding='cp1252'>
 self._logger = val
```

True

### Create spectrum report

Create a spectrum report. You can use a JSON file to create a simple setup or a fully customized one. The following code creates a simple setup and changes the JSON file to customize it. In a spectrum report, you can add limitlines and notes and edit axes, the grid, and the legend. You can create custom reports in non-graphical mode in AEDT 2023 R2 and later.

```
report1 = cir.post.create_report_from_configuration(os.path.join(project_path, 'Spectrum_
↪CISPR_Basic.json'))
out = cir.post.export_report_to_jpg(cir.working_directory, report1.plot_name)
Image(out)
```

<IPython.core.display.Image object>

### Create spectrum report

Every aspect of the report can be customized.

```
report1_full = cir.post.create_report_from_configuration(os.path.join(project_path,
↪'Spectrum_CISPR_Custom.json'))
out = cir.post.export_report_to_jpg(cir.working_directory, report1_full.plot_name)
Image(out)
```

<IPython.core.display.Image object>

### Create transient report

Create a transient report. You can read and modify the JSON file before running the script. The following code modifies the traces before generating the report. You can create custom reports in non-graphical mode in AEDT 2023 R2 and later.

```
props = pyaedt.general_methods.read_json(os.path.join(project_path, 'Transient_CISPR_
↪Custom.json'))

report2 = cir.post.create_report_from_configuration(report_settings=props, solution_name=
↪"NexximTransient")
out = cir.post.export_report_to_jpg(cir.working_directory, report2.plot_name)
Image(out)
```

<IPython.core.display.Image object>

## Create transient report

Property dictionary can be customized in any aspect and new report can be created easily. In this example the curve name is customized.

```
props["expressions"] = {"V(Battery)": {}, "V(U1_VDD)": {}}
props["plot_name"] = "Battery Voltage"
report3 = cir.post.create_report_from_configuration(report_settings=props, solution_name=
 ↪ "NexximTransient")
out = cir.post.export_report_to_jpg(cir.working_directory, report3.plot_name)
Image(out)
```

```
<IPython.core.display.Image object>
```

## Create eye diagram

Create an eye diagram. If the JSON file contains an eye mask, you can create an eye diagram and fully customize it.

```
report4 = cir.post.create_report_from_configuration(os.path.join(project_path,
 ↪ 'EyeDiagram_CISPR_Basic.json'))
out = cir.post.export_report_to_jpg(cir.working_directory, report4.plot_name)
Image(out)
```

```
<IPython.core.display.Image object>
```

## Create eye diagram

You can create custom reports in non-graphical mode in AEDT 2023 R2 and later.

```
report4_full = cir.post.create_report_from_configuration(os.path.join(project_path,
 ↪ 'EyeDiagram_CISPR_Custom.json'))

out = cir.post.export_report_to_jpg(cir.working_directory, report4_full.plot_name)
Image(out)
```

```
<IPython.core.display.Image object>
```

This is how the spectrum looks like .. image:: Resources/spectrum\_plot.png

## Save project and close AEDT

Save the project and close AEDT.

```
cir.save_project()
print("Project Saved in {}".format(cir.project_path))
cir.release_desktop()
```

```
Project Saved in D:/Temp/pyaedt_prj_V0W/custom_reports/
```

```
True
```

**Total running time of the script:** (1 minutes 3.498 seconds)

### Circuit: Touchstone file management

This example shows how you can use objects in a Touchstone file without opening AEDT.

To provide the advanced postprocessing features needed for this example, Matplotlib and NumPy must be installed on your machine.

This example runs only on Windows using CPython.

### Perform required imports

Perform required imports and set the local path to the path for PyAEDT.

```
from pyaedt import downloads
example_path = downloads.download_touchstone()
```

### Import libraries and Touchstone file

Import Matplotlib, NumPy, and the Touchstone file.

```
from pyaedt.generic.touchstone_parser import read_touchstone
```

### Read Touchstone file

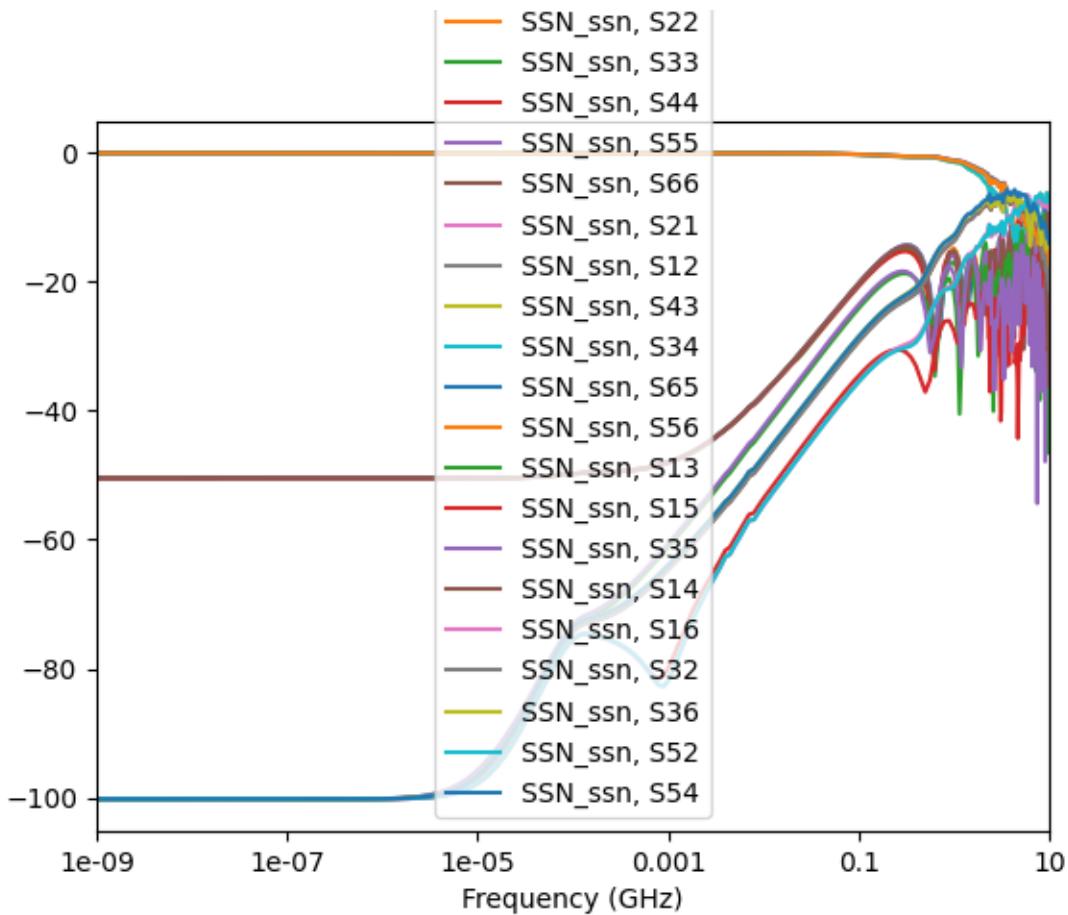
Read the Touchstone file.

```
data = read_touchstone(example_path)
```

### Get curve plot

Get the curve plot by category. The following code shows how to plot lists of the return losses, insertion losses, fext, and next based on a few inputs and port names.

```
data.plot_return_losses()
data.plot_insertion_losses()
data.plot_next_xtalk_losses("U1")
data.plot_fext_xtalk_losses(tx_prefix="U1", rx_prefix="U7")
```



True

## Get curve worst cases

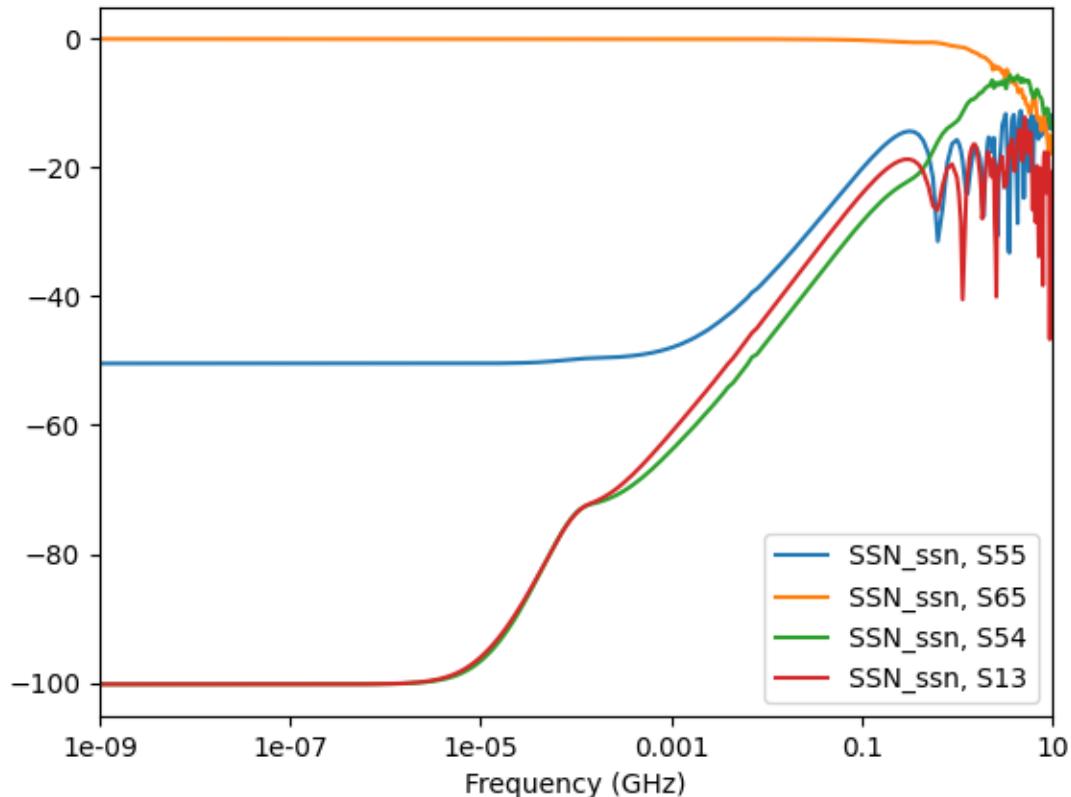
Get curve worst cases.

```
worst_rl, global_mean = data.get_worst_curve(
 freq_min=1, freq_max=20, worst_is_higher=True, curve_list=data.get_return_loss_
 index())
worst_il, mean2 = data.get_worst_curve(freq_min=1,
 freq_max=20,
 worst_is_higher=False,
 curve_list=data.get_insertion_loss_index())
worst_fext, mean3 = data.get_worst_curve(freq_min=1,
 freq_max=20,
 worst_is_higher=True,
 curve_list=data.get_fext_xtalk_index_from_
 prefix(tx_prefix="U1",
 rx_prefix="U7")
```

(continues on next page)

(continued from previous page)

```
)
worst_next, mean4 = data.get_worst_curve(
 freq_min=1, freq_max=20, worst_is_higher=True, curve_list=data.get_next_xtalk_index(
 ↵"U1")
)
```



Total running time of the script: (0 minutes 1.939 seconds)

### Circuit: PCIe virtual compliance

This example shows how to generate a compliance report in PyAEDT using the `VirtualCompliance` class.

## Perform required imports

Perform required imports and set paths.

```
import os.path
import pyaedt
from pyaedt.generic.compliance import VirtualCompliance
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`. The Boolean parameter `new_thread` defines whether to create a new instance of AEDT or try to connect to an existing instance of it.

```
non_graphical = True
new_thread = True
```

## Download example files

Download the project and files needed to run the example.

```
workdir = pyaedt.downloads.download_file('pcie_compliance')
projectdir = os.path.join(workdir, "project")
```

## Launch AEDT

Launch AEDT.

```
d = pyaedt.Desktop(aedt_version, new_desktop_session=new_thread, non_graphical=non_
graphical)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
subprocess 7736 is still running
_warn("subprocess %s is still running" % self.pid,
```

## Open and solve layout

Open the HFSS 3D Layout project and analyze it using the SIwave solver. Before solving, this code ensures that the model is solved from DC to 70GHz and that causality and passivity are enforced.

```
h3d = pyaedt.Hfss3dLayout(os.path.join(projectdir, "PCIE_GEN5_only_layout.aedtz"),
 specified_version=241)
h3d.remove_all_unused_definitions()
h3d.edit_cosim_options(simulate_missing_solution=False)
h3d.setups[0].sweeps[0].props["EnforcePassivity"] = True
h3d.setups[0].sweeps[0].props["Sweeps"]["Data"] = 'LIN 0MHz 70GHz 0.1GHz'
h3d.setups[0].sweeps[0].props["EnforceCausality"] = True
h3d.setups[0].sweeps[0].update()
h3d.analyze()
h3d = pyaedt.Hfss3dLayout(specified_version=241)
touchstone_path = h3d.export_touchstone()
```

## Create LNA project

Use the LNA setup to retrieve Touchstone files and generate frequency domain reports.

```
cir = pyaedt.Circuit(projectname=h3d.project_name, designname="Touchstone")
status, diff_pairs, comm_pairs = cir.create_lna_schematic_from_snp(input_file=touchstone_
 path, start_frequency=0,
 stop_frequency=70,
 auto_assign_diff_pairs=True,
 separation=".,",
 pattern=["component", "pin", "net"],
 analyze=True)

insertion = cir.get_all_insertion_loss_list(trlist=diff_pairs,
 reclist=diff_pairs,
 tx_prefix="X1",
 rx_prefix="U1",
 math_formula="dB",
 net_list=["RX0", "RX1", "RX2", "RX3"]
)
return_diff = cir.get_all_return_loss_list(excitation_names=diff_pairs,
 excitation_name_prefix="X1",
 math_formula="dB",
 net_list=["RX0", "RX1", "RX2", "RX3"]
)
return_comm = cir.get_all_return_loss_list(excitation_names=comm_pairs,
 excitation_name_prefix="COMMON_X1",
 math_formula="dB",
 net_list=["RX0", "RX1", "RX2", "RX3"])
```

## Create TDR project

Create a TDR project to compute transient simulation and retrieve the TDR measurement on a differential pair. The original circuit schematic is duplicated and modified to achieve this target.

```
result, tdr_probe_name = cir.create_tdr_schematic_from_snp(input_file=touchstone_path,
 probe_pins=["X1.A2.PCIe_Gen4_"
 ↵RX0_P"],
 probe_ref_pins=["X1.A3.PCIe_"
 ↵Gen4_RX0_N"],
 termination_pins=["U1.AP26."
 ↵PCIe_Gen4_RX0_P",
 ↵PCIe_Gen4_RX0_N"],
 differential=True, rise_
 ↵time=35, use_convolution=True,
 analyze=True, design_name="TDR"
 ↵")
```

## Create AMI project

Create an Ibis AMI project to compute an eye diagram simulation and retrieve eye mask violations.

```
result, eye_curve_tx, eye_curve_rx = cir.create_ami_schematic_from_snp(input_
file=touchstone_path,
 ibis_ami=os.path.
 ↵join(projectdir, "models",
 ↵"pcieg5_32gt.ibs"),
 component_name=
 "Spec_Model", tx_buffer_name="1p",
 ↵",
 ↵PCIe_Gen4_TX0_CAP_P"],
 tx_pins=["U1.AM25.",
 ↵PCIe_Gen4_TX0_CAP_N],
 tx_refs=["U1.AL25.
 ↵PCIe_Gen4_TX0_P"],
 rx_pins=["X1.B2.
 ↵PCIe_Gen4_TX0_N"],
 rx_refs=["X1.B3.
 ↵buffer=False, differential=True,
 ↵"random_bit_count=2.5e3 random_seed=1",
 ↵25ps", use_convolution=True,
 ↵design_name="AMI")
 cir.save_project()
```

True

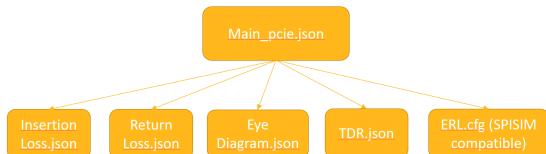
## Create virtual compliance report

Initialize the `VirtualCompliance` class and set up the main project information needed to generate the report.

### Virtual Compliance Python Class

- A class to generate report(s) based on configuration files.
- 2 types of configuration files:
  - Standard-specific configuration file (eg. PCIE Gen.5 )
  - Post-processing configuration file
- The standard-specific configuration file contains information about the structure of the report, data output, pass-fail criteria for evaluation in Ansys SPISIM.
- This Framework enables anyone to create or customize the Virtual Compliance Report by modifying the configuration files.

### Configuration file example



- Plot configuration files are reusable. Relatively minor modifications are required to address a new standard (eg. Eye mask or limit line)
- The standard-specific configuration file can be modified to generate custom metrics or a subset of the relevant quantities for a given standard.

```

template = os.path.join(workdir, "pcie_gen5_templates", "main.json")
v = VirtualCompliance(cir.desktop_class, str(template))

```

## Customize project and design

Define the path to the project file and the design names to be used in each report generation.

```

import os.path
from pathlib import Path
workdir = Path(__file__).parent
workdir = r'C:\ansysdev\Models\Compliance\pcie\wf_pcic'
from pyaedt import Desktop
from pyaedt.generic.compliance import VirtualCompliance

d = Desktop(specified_version=241, new_desktop_session=False) Launch Electronics Desktop 2024R1

template = os.path.join(workdir, "pcie_gen5_templates", "main.json") Initialize VirtualCompliance class

v = VirtualCompliance(d, str(template))
v.project_file = 'C:\\ansysdev\\Models\\Compliance\\pcie\\wf_pcic\\project\\PCIE_GEN5.aedt' Optional override parameters

v.reports["insertion losses"].traces = [
 "dB(S(U1_RX0,X1_RX0))",
 "dB(S(U1_RX1,X1_RX1))",
 "dB(S(U1_RX3,X1_RX3))"
]

v.specs_folder = r'C:\\ansysdev\\Models\\Compliance\\pcie\\wf_pcic\\readme_pictures' Create the report

v.reports["tdr from circuit"].traces = ["0(A176:zdiff)"]
v.parameters["erl"].design_name = "Circuit1" Close the report

v.create_compliance_report() Close Electronics Desktop

d.release_desktop(close_projects=True, close_on_exit=True)
|
```

```

v.project_file = cir.project_file
v.reports["insertion losses"].design_name = "LNA"
v.reports["return losses"].design_name = "LNA"
v.reports["common mode return losses"].design_name = "LNA"
v.reports["tdr from circuit"].design_name = "TDR"
v.reports["eye1"].design_name = "AMI"
v.reports["eye3"].design_name = "AMI"
v.parameters["erl"].design_name = "LNA"
v.specs_folder = os.path.join(workdir, 'readme_pictures')
```

## Define trace names

Change the trace name with projects and users. Reuse the compliance template and update traces accordingly.

```

v.reports["insertion losses"].traces = insertion

v.reports["return losses"].traces = return_diff

v.reports["common mode return losses"].traces = return_commm

v.reports["eye1"].traces = eye_curve_tx
v.reports["eye3"].traces = eye_curve_tx
v.reports["tdr from circuit"].traces = tdr_probe_name
v.parameters["erl"].trace_pins = [
 "X1.A5.PCIe_Gen4_RX1_P", "X1.A6.PCIe_Gen4_RX1_N", "U1.AR25.PCIe_Gen4_RX1_P", "U1.
 ↵AP25.PCIe_Gen4_RX1_N",
 [7, 8, 18, 17]]
```

## Generate PDF report

Generate the reports and produce a PDF report.



Figure 4. Plot frequency for trace  $\text{dB}(S(U1_{\text{RX}0},X1_{\text{RX}0}))$

Check Zone	Trace Name	Criteria	Pass/Fail limit value	Worst simulated value	X at worst value	Test Result
Zone 1	$\text{dB}(S(U1_{\text{RX}0},X1_{\text{RX}0}))$	Lower Limit	-4.2	-3.87648	14.95	PASS

Table Pass Fail Criteria on insertion losses  $\text{dB}(S(U1_{\text{RX}0},X1_{\text{RX}0}))$

3.5 common mode return losses

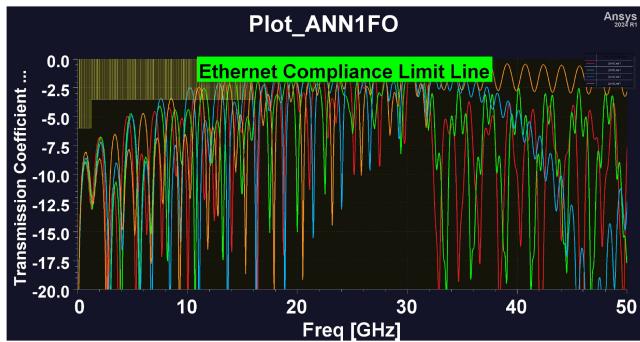


Figure 8. Plot frequency for common mode return losses

Check Zone	Trace Name	Criteria	Pass/Fail limit value	Worst simulated value	X at worst value	Test Result
Zone 1	$\text{dB}(S(\text{COMMON\_X1\_RX0},\text{COM\_MON\_X1\_RX0}))$	Upper Limit	-6	-8.90468	0.7	PASS
Zone 2	$\text{dB}(S(\text{COMMON\_X1\_RX0},\text{COM\_MON\_X1\_RX0}))$	Upper Limit	-3.5	-6.7524	2.0	PASS
Zone 3	$\text{dB}(S(\text{COMMON\_X1\_RX0},\text{COM\_MON\_X1\_RX0}))$	Upper Limit	-3.5	-1.76737	15.7	FAIL

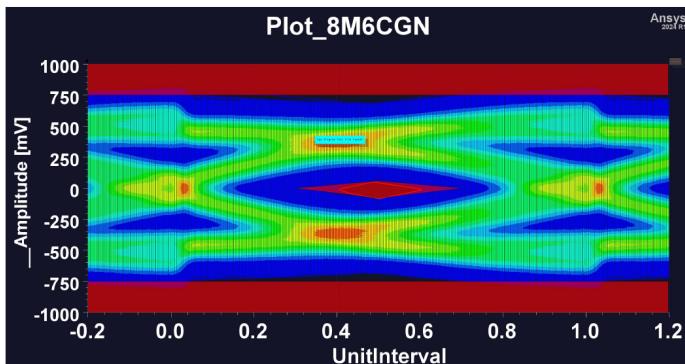


Figure 10. Plot statistical eye for trace b\_input\_67

Pass Fail Criteria	Test Result
Eye Mask Violation:	FAIL on 1423 points.
Upper/Lower Mask Violation:	FAIL

Table Pass Fail Criteria on eye1\_b\_input\_67

EyeLevelZero	-376.3920
EyeLevelOne	369.2000
EyeAmplitude	745.5920
EyeHeight	164.4694
EyeSignalToNoise	3.8491 (SI)
EyeOpeningFactor	0.7402 (SI)
EyeWidth	0.4347
EyeJitterP2P	0.8483
EyeJitterRMS	0.0942
MinEyeWidth	0.1776
MinEyeHeight	91.7058

Table Eye Measurements on eye1\_b\_input\_67

```
v.create_compliance_report()
d.release_desktop(True, True)
```

```
True
```

**Total running time of the script:** (6 minutes 47.353 seconds)

### 3.12.11 EMIT examples

These examples use PyAEDT to show some end-to-end workflows for EMIT. This includes schematic generation, setup, and postprocessing.

#### EMIT: Classify interference type

This example shows how you can use PyAEDT to load an existing AEDT project with an EMIT design and analyze the results to classify the worst-case interference.

## Perform required imports

Perform required imports.

```
import sys
from pyaedt.emit_core.emit_constants import InterfererType, ResultType, TxRxMode
from pyaedt import Emit
import pyaedt
import os
import pyaedt.generic.constants as consts
import subprocess
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"

Check to see which Python libraries have been installed
reqs = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])
installed_packages = [r.decode().split('==')[0] for r in reqs.split()]

Install required packages if they are not installed
def install(package):
 subprocess.check_call([sys.executable, '-m', 'pip', 'install', package])

Install plotly library (if needed) to display legend and scenario matrix results
(internet connection needed)
required_packages = ['plotly']
for package in required_packages:
 if package not in installed_packages:
 install(package)

Import plotly library
import plotly.graph_objects as go

Define colors for tables
table_colors = {"green": '#7d73ca', "yellow": '#d359a2', "orange": '#ff6361', "red": '#ffa600', "white": '#ffffff'}
header_color = 'grey'

Check for if emit version is compatible
if aedt_version <= "2023.1":
 print("Warning: this example requires AEDT 2023.2 or later.")
 sys.exit()
```

## Launch AEDT with EMIT

Launch AEDT with EMIT. The Desktop class initializes AEDT and starts it on the specified version and in the specified graphical mode.

```
non_graphical = False
new_thread = True
desktop = pyaedt.launch_desktop(aedt_version, non_graphical=non_graphical, new_desktop_
 ↪session=new_thread)

path_to_desktop_project = pyaedt.downloads.download_file("emit", "interference.aedtz")
emitapp = Emit(non_graphical=False, new_desktop_session=False, projectname=path_to_
 ↪desktop_project)

Get all the radios in the project
~~~~~
Get lists of all transmitters and receivers in the project.
rev = emitapp.results.analyze()
tx_interferer = InterfererType().TRANSMITTERS
rx_radios = rev.get_receiver_names()
tx_radios = rev.get_interferer_names(tx_interferer)
domain = emitapp.results.interaction_domain()

if tx_radios is None or rx_radios is None:
 print("No receivers or transmitters are in the design.")
 sys.exit()
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: ↪
 subprocess 8480 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 ↪py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 ↪log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Classify the interference

Iterate over all the transmitters and receivers and compute the power at the input to each receiver due to each of the transmitters. Computes which, if any, type of interference occurred.

```
power_matrix = []
all_colors = []
all_colors, power_matrix = rev.interference_type_classification(domain, use_filter=False,
 ↪ filter_list=[])
```

## Save project and close AEDT

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

```
emitapp.save_project()
emitapp.release_desktop()
```

```
True
```

## Create a scenario matrix view

Create a scenario matrix view with the transmitters defined across the top and receivers down the left-most column. The power at the input to each receiver is shown in each cell of the matrix and color-coded based on the interference type.

```
def create_scenario_view(emis, colors, tx_radios, rx_radios):
 """Create a scenario matrix-like table with the higher received
 power for each Tx-Rx radio combination. The colors
 used for the scenario matrix view are based on the interference type."""

 all_colors = []
 for color in colors:
 col = []
 for cell in color:
 col.append(table_colors[cell])
 all_colors.append(col)

 fig = go.Figure(data=[go.Table(
 header=dict(
 values=['Tx/Rx', '{}'.format(tx_radios[0]), '{}'.
 format(tx_radios[1])],
 line_color='darkslategray',
 fill_color='grey',
 align=['left', 'center'],
 font=dict(color='white', size=16)
),
 cells=dict(
 values=[
 rx_radios,
 emis[0],
 emis[1]],
 line_color='darkslategray',
 fill_color=['white', all_colors[0], all_colors[1]],
 align=['left', 'center'],
 height=25,
 font=dict(
 color=['darkslategray', 'black'],
 size=15)
)
]))
 fig.update_layout(
```

(continues on next page)

(continued from previous page)

```

 title=dict(
 text='Interference Type Classification',
 font=dict(color='darkslategray', size=20),
 x=0.5
),
 width=600
)
fig.show()

```

## Generate a legend

Define the interference types and colors used to display the results of the analysis.

```

def create_legend_table():
 """Create a table showing the interference types."""
 classifications = ['In-band/In-band', 'Out-of-band/In-band',
 'In-band/Out-of-band', 'Out-of-band/Out-of-band']
 fig = go.Figure(data=[go.Table(
 header=dict(
 values=['Interference Type (Source/Victim)'],
 line_color='darkslategray',
 fill_color=header_color,
 align=['center'],
 font=dict(color='white', size=16)
),
 cells=dict(
 values=[classifications],
 line_color='darkslategray',
 fill_color=[[table_colors['red'], table_colors['orange'], table_colors[
 'yellow'], table_colors['green']]],
 align=['center'],
 height=25,
 font=dict(
 color=['darkslategray', 'black'],
 size=15
)
)
)])
 fig.update_layout(
 title=dict(
 text='Interference Type Classification',
 font=dict(color='darkslategray', size=20),
 x=0.5
),
 width=600
)
 fig.show()

if os.getenv("PYAEDT_DOC_GENERATION", "False") != "1":
 # Create a scenario view for all the interference types
 create_scenario_view(power_matrix, all_colors, tx_radios, rx_radios)

```

(continues on next page)

(continued from previous page)

```
Create a legend for the interference types
create_legend_table()
```

**Total running time of the script:** (8 minutes 52.998 seconds)

## EMIT: Compute receiver protection levels

This example shows how you can use PyAEDT to open an AEDT project with an EMIT design and analyze the results to determine if the received power at the input to each receiver exceeds the specified protection levels.

### Perform required imports

Perform required imports.

sphinx\_gallery\_thumbnail\_path = Resources/emit\_protection\_levels.png

```
import os
import sys
import subprocess
import pyaedt
from pyaedt import Emit
from pyaedt.emit_core.emit_constants import TxRxMode, ResultType, InterfererType

Check to see which Python libraries have been installed
reqs = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])
installed_packages = [r.decode().split('==')[0] for r in reqs.split()]

Install required packages if they are not installed
def install(package):
 subprocess.check_call([sys.executable, '-m', 'pip', 'install', package])

Install any missing libraries
required_packages = ['plotly']
for package in required_packages:
 if package not in installed_packages:
 install(package)

Import required modules
import plotly.graph_objects as go
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. "PYAEDT\_NON\_GRAPHICAL" is needed to generate documentation only. You can set `non_graphical` either to `True` or `False`. The `new_thread` Boolean variable defines whether to create a new instance of AEDT or try to connect to existing instance of it if one is available.

```
non_graphical = False
new_thread = True
```

## Launch AEDT with EMIT

Launch AEDT with EMIT. The `Desktop` class initializes AEDT and starts it on the specified version and in the specified graphical mode.

```
if aedt_version <= "2023.1":
 print("Warning: this example requires AEDT 2023.2 or later.")
 sys.exit()

d = pyaedt.launch_desktop(aedt_version, non_graphical, new_thread)
emitapp = Emit(pyaedt.generate_unique_project_name())
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 8096 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 _py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 _log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Specify the protection levels

The protection levels are specified in dBm. If the damage threshold is exceeded, permanent damage to the receiver front end may occur. Exceeding the overload threshold severely densensitizes the receiver. Exceeding the intermod threshold can drive the victim receiver into non-linear operation, where it operates as a mixer. Exceeding the desense threshold reduces the signal-to-noise ratio and can reduce the maximum range, maximum bandwidth, and/or the overall link quality.

```
header_color = 'grey'
damage_threshold = 30
overload_threshold = -4
intermod_threshold = -30
desense_threshold = -104
```

(continues on next page)

(continued from previous page)

```
protection_levels = [damage_threshold, overload_threshold, intermod_threshold, desense_
 ↪threshold]
```

## Create and connect EMIT components

Set up the scenario with radios connected to antennas.

```
bluetooth, blue_ant = emitapp.modeler.components.create_radio_antenna("Bluetooth Low_Energy (LE)", "Bluetooth")
gps, gps_ant = emitapp.modeler.components.create_radio_antenna("GPS Receiver", "GPS")
wifi, wifi_ant = emitapp.modeler.components.create_radio_antenna("WiFi - 802.11-2012",
 ↪"WiFi")
```

## Configure the radios

Enable the HR-DSSS bands for the Wi-Fi radio and set the power level for all transmit bands to -20 dBm.

```
bands = wifi.bands()
for band in bands:
 if "HR-DSSS" in band.node_name:
 if "Ch 1-13" in band.node_name:
 band.enabled=True
 band.set_band_power_level(-20)

Reduce the bluetooth transmit power
bands = bluetooth.bands()
for band in bands:
 band.set_band_power_level(-20)

def get_radio_node(radio_name):
 """Get the radio node that matches the
 given radio name.
 Arguments:
 radio_name: String name of the radio.
 Returns: Instance of the radio.
 """
 if gps.name == radio_name:
 radio = gps
 elif bluetooth.name == radio_name:
 radio = bluetooth
 else:
 radio = wifi
 return radio

bands = gps.bands()
for band in bands:
 for child in band.children:
 if "L2 P(Y)" in band.node_name:
 band.enabled=True
```

(continues on next page)

(continued from previous page)

```
else:
 band.enabled=False
```

## Load the results set

Create a results revision and load it for analysis.

```
rev = emitapp.results.analyze()
```

## Generate a legend

Define the thresholds and colors used to display the results of the protection level analysis.

```
def create_legend_table():
 """Create a table showing the defined protection levels."""
 protectionLevels = ['>{} dBm'.format(damage_threshold), '>{} dBm'.format(overload_
threshold),
 '>{} dBm'.format(intermod_threshold), '>{} dBm'.format(desense_threshold)]
 fig = go.Figure(data=[go.Table(
 header=dict(
 values=['Interference', 'Power Level Threshold'],
 line_color='darkslategray',
 fill_color=header_color,
 align=['left', 'center'],
 font=dict(color='white', size=16)
),
 cells=dict(
 values=[['Damage', 'Overload', 'Intermodulation', 'Clear'], protectionLevels],
 line_color='darkslategray',
 fill_color=['white', ['red', 'orange', 'yellow', 'green']],
 align = ['left', 'center'],
 font = dict(
 color = ['darkslategray', 'black'],
 size = 15)
)
)])
 fig.update_layout(
 title=dict(
 text='Protection Levels (dBm)',
 font=dict(color='darkslategray', size=20),
 x = 0.5
),
 width = 600
)
 fig.show()
```

## Create a scenario matrix view

Create a scenario matrix view with the transmitters defined across the top and receivers down the left-most column. The power at the input to each receiver is shown in each cell of the matrix and color-coded based on the protection level thresholds defined.

```
def create_scenario_view(emis, colors, tx_radios, rx_radios):
 """Create a scenario matrix-like table with the higher received
 power for each Tx-Rx radio combination. The colors
 used for the scenario matrix view are based on the highest
 protection level that the received power exceeds."""
 fig = go.Figure(data=[go.Table(
 header=dict(
 values=['Tx/Rx', '{}'.format(tx_radios[0]), '{}'.
 ↪format(tx_radios[1])],
 line_color='darkslategray',
 fill_color=header_color,
 align=['left', 'center'],
 font=dict(color='white', size=16)
),
 cells=dict(
 values=[
 rx_radios,
 emis[0],
 emis[1]],
 line_color='darkslategray',
 fill_color=['white', colors[0], colors[1]],
 align = ['left', 'center'],
 font = dict(
 color = ['darkslategray', 'black'],
 size = 15)
)
)])
 fig.update_layout(
 title=dict(
 text='Protection Levels (dBm)',
 font=dict(color='darkslategray', size=20),
 x = 0.5
),
 width = 600
)
 fig.show()
```

## Get all the radios in the project

Get lists of all transmitters and receivers in the project.

```
if os.getenv("PYAEDT_DOC_GENERATION", "False") != "1":
 rev = emitapp.results.current_revision
 rx_radios = rev.get_receiver_names()
 tx_radios = rev.get_interferer_names(InterfererType.TRANSMITTERS)
 domain = emitapp.results.interaction_domain()
```

## Classify the results

Iterate over all the transmitters and receivers and compute the power at the input to each receiver due to each of the transmitters. Computes which, if any, protection levels are exceeded by these power levels.

```
if os.getenv("PYAEDT_DOC_GENERATION", "False") != "1":
 power_matrix=[]
 all_colors=[]

 all_colors, power_matrix = rev.protection_level_classification(domain, global_levels_=
→= protection_levels)

 # Create a scenario matrix-like view for the protection levels
 create_scenario_view(power_matrix, all_colors, tx_radios, rx_radios)

 # Create a legend for the protection levels
 create_legend_table()
```

## Save project and close AEDT

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

```
emitapp.save_project()
emitapp.release_desktop(close_projects=True, close_desktop=True)
```

```
True
```

**Total running time of the script:** (0 minutes 38.408 seconds)

## EMIT: antenna

This example shows how you can use PyAEDT to create a project in EMIT for the simulation of an antenna.

## Perform required inputs

Perform required imports.

```
sphinx_gallery_thumbnail_path = Resources/emit_simple_cosite.png
```

```
import os
import pyaedt
from pyaedt.emit_core.emit_constants import TxRxMode, ResultType
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`. The `NewThread` Boolean variable defines whether to create a new instance of AEDT or try to connect to existing instance of it if one is available.

```
non_graphical = False
NewThread = True
```

## Launch AEDT with EMIT

Launch AEDT with EMIT. The `Desktop` class initializes AEDT and starts it on the specified version and in the specified graphical mode.

```
d = pyaedt.launch_desktop(aedt_version, non_graphical, NewThread)
aedtapp = pyaedt.Emit(pyaedt.generate_unique_project_name())
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning: subprocess 5264 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create and connect EMIT components

Create three radios and connect an antenna to each one.

```
rad1 = aedtapp.modeler.components.create_component("New Radio")
ant1 = aedtapp.modeler.components.create_component("Antenna")
if rad1 and ant1:
 ant1.move_and_connect_to(rad1)

Convenience method to create a radio and antenna connected together
rad2, ant2 = aedtapp.modeler.components.create_radio_antenna("GPS Receiver")
rad3, ant3 = aedtapp.modeler.components.create_radio_antenna("Bluetooth Low Energy (LE)",
 ↵ "Bluetooth")
```

## Define coupling among RF systems

Define the coupling among the RF systems. This portion of the EMIT API is not yet implemented.

## Run EMIT simulation

Run the EMIT simulation.

This part of the example requires Ansys AEDT 2023 R2.

```
if aedt_version > "2023.1" and os.getenv("PYAEDT_DOC_GENERATION", "False") != "1":
 rev = aedtapp.results.analyze()
 rx_bands = rev.get_band_names(rad2.name, TxRxMode.RX)
 tx_bands = rev.get_band_names(rad3.name, TxRxMode.TX)
 domain = aedtapp.results.interaction_domain()
 domain.set_receiver(rad2.name, rx_bands[0], -1)
 domain.set_interferer(rad3.name, tx_bands[0])
 interaction = rev.run(domain)
 worst = interaction.get_worst_instance(ResultType.EMI)
 if worst.has_valid_values():
 emi = worst.get_value(ResultType.EMI)
 print("Worst case interference is: {} dB".format(emi))
```

## Save project and close AEDT

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

```
aedtapp.save_project()
aedtapp.release_desktop(close_projects=True, close_desktop=True)
```

```
True
```

**Total running time of the script:** (0 minutes 36.490 seconds)

## EMIT: HFSS to EMIT coupling

This example shows how you can use PyAEDT to open an AEDT project with an HFSS design, create an EMIT design in the project, and link the HFSS design as a coupling link in the EMIT design.

### Perform required imports

Perform required imports.

```
sphinx_gallery_thumbnail_path = Resources/emit_hfss.png
```

```
import os

Import required modules
import pyaedt
from pyaedt.generic.filesystem import Scratch
from pyaedt.emit_core.emit_constants import TxRxMode, ResultType
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Set non-graphical mode

Set non-graphical mode. You can set `non_graphical` either to `True` or `False`. The Boolean parameter `new_thread` defines whether to create a new instance of AEDT or try to connect to an existing instance of it.

The following code uses AEDT 2023 R2.

```
non_graphical = False
NewThread = True
scratch_path = pyaedt.generate_unique_folder_name()
```

### Launch AEDT with EMIT

Launch AEDT with EMIT. The `Desktop` class initializes AEDT and starts it on the specified version and in the specified graphical mode.

```
d = pyaedt.launch_desktop(aedt_version, non_graphical, NewThread)

temp_folder = os.path.join(scratch_path, ("EmitHFSSEExample"))
if not os.path.exists(temp_folder):
 os.mkdir(temp_folder)

example_name = "Cell Phone RFI Desense"
example_aedt = example_name + ".aedt"
example_results = example_name + ".aedtresults"
```

(continues on next page)

(continued from previous page)

```
example_lock = example_aedt + ".lock"
example_pdf_file = example_name + " Example.pdf"

example_dir = os.path.join(d.install_path, "Examples\\EMIT")
example_project = os.path.join(example_dir, example_aedt)
example_results_folder = os.path.join(example_dir, example_results)
example_pdf = os.path.join(example_dir, example_pdf_file)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 3676 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

If the Cell Phone RFT Defense example is not in the installation directory, exit from this example.

```
if not os.path.exists(example_project):
 msg = """
 Cell phone RFT Desense example file is not in the
 Examples/EMIT directory under the EDT installation. You cannot run this example.
 """
 print(msg)
 d.release_desktop(True, True)
 exit()

my_project = os.path.join(temp_folder, example_aedt)
my_results_folder = os.path.join(temp_folder, example_results)
my_project_lock = os.path.join(temp_folder, example_lock)
my_project_pdf = os.path.join(temp_folder, example_pdf_file)

if os.path.exists(my_project):
 os.remove(my_project)

if os.path.exists(my_project_lock):
 os.remove(my_project_lock)

with Scratch(scratch_path) as local_scratch:
 local_scratch.copyfile(example_project, my_project)
 local_scratch.copyfolder(example_results_folder, my_results_folder)
 if os.path.exists(example_pdf):
 local_scratch.copyfile(example_pdf, my_project_pdf)

aedtapp = pyaedt.Emit(my_project)
```

## Create and connect EMIT components

Create two radios with antennas connected to each one.

```
rad1, ant1 = aedtapp.modeler.components.create_radio_antenna("Bluetooth Low Energy (LE)")
rad2, ant2 = aedtapp.modeler.components.create_radio_antenna("Bluetooth Low Energy (LE)")
```

## Define coupling among RF systems

Define coupling among the RF systems.

```
for link in aedtapp.couplings.linkable_design_names:
 aedtapp.couplings.add_link(link)

for link in aedtapp.couplings.coupling_names:
 aedtapp.couplings.update_link(link)
```

## Run EMIT simulation

Run the EMIT simulation.

This part of the example requires Ansys AEDT 2023 R2.

```
if aedt_version > "2023.1":
 rev = aedtapp.results.analyze()
 rx_bands = rev.get_band_names(rad1.name, TxRxMode.RX)
 tx_bands = rev.get_band_names(rad2.name, TxRxMode.TX)
 domain = aedtapp.results.interaction_domain()
 domain.set_receiver(rad1.name, rx_bands[0], -1)
 domain.set_interfererer(rad2.name, tx_bands[0])
 interaction = rev.run(domain)
 worst = interaction.get_worst_instance(ResultType.EMI)
 if worst.has_valid_values():
 emi = worst.get_value(ResultType.EMI)
 print("Worst case interference is: {} dB".format(emi))
```

Worst case interference is: 90.0 dB

## Save project and close AEDT

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

```
aedtapp.save_project()
aedtapp.release_desktop(close_projects=True, close_desktop=True)
```

True

**Total running time of the script:** (0 minutes 48.547 seconds)

## EMIT: Classify interference type GUI

This example uses a GUI to open an AEDT project with an EMIT design and analyze the results to classify the worst-case interference.

### Perform required imports

Perform required imports.

```
import sys
from pyaedt.emit_core.emit_constants import InterfererType, ResultType, TxRxMode
from pyaedt import Emit
from pyaedt import get_pyaedt_app
import pyaedt
import os
import subprocess
import pyaedt.generic.constants as consts

Check that emit is a compatible version
aedt_version = "2024.1"
if aedt_version < "2023.2":
 print("Must have v2023.2 or later")
 sys.exit()

Check to see which Python libraries have been installed
reqs = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])
installed_packages = [r.decode().split('==')[0] for r in reqs.split()]

Install required packages if they are not installed
def install(package):
 subprocess.check_call([sys.executable, '-m', 'pip', 'install', package])

Install required libraries for GUI and Excel exporting (internet connection needed)
required_packages = ['PySide6', 'openpyxl']
for package in required_packages:
 if package not in installed_packages:
 install(package)

Import PySide6 and openpyxl libraries
from PySide6 import QtWidgets, QtUiTools, QtGui, QtCore
from openpyxl.styles import PatternFill
import openpyxl

Uncomment if there are Qt plugin errors
import PySide6
dirname = os.path.dirname(PySide6.__file__)
plugin_path = os.path.join(dirname, 'plugins', 'platforms')
os.environ['QT_QPA_PLATFORM_PLUGIN_PATH'] = plugin_path

Launch EMIT
non_graphical = False
new_thread = True
```

(continues on next page)

(continued from previous page)

```

desktop = pyaedt.launch_desktop(aedt_version, non_graphical, new_thread)

Add emitapi to system path
emit_path = os.path.join(desktop.install_path, "Delcross")
sys.path.insert(0, emit_path)
import EmitApiPython
api = EmitApiPython.EmitApi()

Define .ui file for GUI
ui_file = pyaedt.downloads.download_file("emit", "interference_gui.ui")
Ui_MainWindow, _ = QtUiTools.loadUiType(ui_file)

class DoubleDelegate(QtWidgets.QStyledItemDelegate):
 def __init__(self, decimals, values, max_power, min_power):
 super().__init__()
 self.decimals = decimals
 self.values = values
 self.max_power = max_power
 self.min_power = min_power

 def createEditor(self, parent, option, index):
 editor = super().createEditor(parent, option, index)
 if isinstance(editor, QtWidgets.QLineEdit):
 validator = QtGui.QDoubleValidator(parent)
 num_rows = len(self.values)
 cur_row = index.row()
 if cur_row == 0:
 min_val = self.values[1]
 max_val = self.max_power
 elif cur_row == num_rows - 1:
 min_val = self.min_power
 max_val = self.values[cur_row-1]
 else:
 min_val = self.values[cur_row + 1]
 max_val = self.values[cur_row - 1]
 validator.setRange(min_val, max_val, self.decimals)
 validator.setNotation(QtGui.QDoubleValidator.Notation.StandardNotation)
 editor.setValidator(validator)
 return editor

 def update_values(self, values):
 self.values = values

class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
 def __init__(self):
 super(MainWindow, self).__init__()
 self.emitapp = None
 self.populating_dropdown = False
 self.setupUi(self)
 self.setup_widgets()

#####

```

(continues on next page)

(continued from previous page)

```

Setup widgets
~~~~~
Define all widgets from the UI file, connect the widgets to functions, define
table colors, and format table settings.

def setup_widgets(self):
 # Widget definitions for file selection/tab management
 self.file_select_btn = self.findChild(QtWidgets.QToolButton, "file_select_btn")
 self.file_path_box = self.findChild(QtWidgets.QLineEdit, "file_path_box")
 self.design_name_dropdown = self.findChild(QtWidgets.QComboBox, "design_name_"
→dropdown")
 self.design_name_dropdown.setEnabled(True)
 self.tab_widget = self.findChild(QtWidgets.QTabWidget, "tab_widget")

 # Widget definitions for protection level classification
 self.protection_results_btn = self.findChild(QtWidgets.QPushButton, "protection_"
→results_btn")
 self.protection_matrix = self.findChild(QtWidgets.QTableWidget, "protection_"
→matrix")
 self.protection_legend_table = self.findChild(QtWidgets.QTableWidget,
→"protection_legend_table")

 self.damage_check = self.findChild(QtWidgets.QCheckBox, "damage_check")
 self.overload_check = self.findChild(QtWidgets.QCheckBox, "overload_check")
 self.intermodulation_check = self.findChild(QtWidgets.QCheckBox,
→"intermodulation_check")
 self.desensitization_check = self.findChild(QtWidgets.QCheckBox,
→"desensitization_check")
 self.protection_export_btn = self.findChild(QtWidgets.QPushButton, "protection_"
→export_btn")
 self.radio_specific_levels = self.findChild(QtWidgets.QCheckBox, "radio_specific_"
→levels")
 self.radio_dropdown = self.findChild(QtWidgets.QComboBox, "radio_dropdown")
 self.protection_save_img_btn = self.findChild(QtWidgets.QPushButton, 'protection_'
→save_img_btn')

 # warning label
 self.warning_label = self.findChild(QtWidgets.QLabel, "warnings")
 myFont = QtGui.QFont()
 myFont.setBold(True)
 self.warning_label.setFont(myFont)
 self.warning_label setHidden(True)
 self.design_name_dropdown.currentIndexChanged.connect(self.design_dropdown_
→changed)

 # Setup for protection level buttons and table
 self.protection_results_btn.setEnabled(False)
 self.protection_export_btn.setEnabled(False)
 self.protection_save_img_btn.setEnabled(False)
 self.file_select_btn.clicked.connect(self.open_file_dialog)
 self.protection_export_btn.clicked.connect(self.save_results_excel)
 self.protection_results_btn.clicked.connect(self.protection_results)

```

(continues on next page)

(continued from previous page)

```

self.protection_legend_table.resizeRowsToContents()
self.protection_legend_table.resizeColumnsToContents()
self.damage_check.stateChanged.connect(self.protection_results)
self.overload_check.stateChanged.connect(self.protection_results)
self.intermodulation_check.stateChanged.connect(self.protection_results)
self.desensitization_check.stateChanged.connect(self.protection_results)
self.protection_legend_table.setEditTriggers(QtWidgets.QTableWidget.
 DoubleClicked)
 self.global_protection_level = True
 self.protection_levels = {}
 values = [float(self.protection_legend_table.item(row, 0).text()) for row in
range(self.protection_legend_table.rowCount())]
 self.protection_levels['Global'] = values
 self.changing = False
 self.radio_dropdown.currentIndexChanged.connect(self.radio_dropdown_changed)
 self.protection_legend_table.itemChanged.connect(self.table_changed)
 self.protection_save_img_btn.clicked.connect(self.save_image)

 # Widget definitions for interference type
 self.interference_results_btn = self.findChild(QtWidgets.QPushButton,
"interference_results_btn")
 self.interference_matrix = self.findChild(QtWidgets.QTableWidget, "interference_
matrix")
 self.interference_legend_table = self.findChild(QtWidgets.QTableWidget,
"interference_legend_table")

 # set the items read only
 for i in range(0, self.interference_legend_table.rowCount()):
 item = self.interference_legend_table.item(i, 0)
 item.setFlags(QtCore.Qt.ItemIsEnabled)
 self.interference_legend_table.setItem(i, 0, item)

 self.in_in_check = self.findChild(QtWidgets.QCheckBox, "in_in_check")
 self.in_out_check = self.findChild(QtWidgets.QCheckBox, "in_out_check")
 self.out_in_check = self.findChild(QtWidgets.QCheckBox, "out_in_check")
 self.out_out_check = self.findChild(QtWidgets.QCheckBox, "out_out_check")
 self.interference_export_btn = self.findChild(QtWidgets.QPushButton,
"interference_export_btn")
 self.interference_save_img_btn = self.findChild(QtWidgets.QPushButton,
"interference_save_img_btn")

 # Setup for interference type buttons and table
 self.interference_results_btn.setEnabled(False)
 self.interference_export_btn.setEnabled(False)
 self.interference_save_img_btn.setEnabled(False)
 self.interference_export_btn.clicked.connect(self.save_results_excel)
 self.interference_results_btn.clicked.connect(self.interference_results)
 self.interference_legend_table.resizeRowsToContents()
 self.interference_legend_table.resizeColumnsToContents()
 self.in_in_check.stateChanged.connect(self.interference_results)
 self.in_out_check.stateChanged.connect(self.interference_results)
 self.out_in_check.stateChanged.connect(self.interference_results)

```

(continues on next page)

(continued from previous page)

```

self.out_out_check.stateChanged.connect(self.interference_results)
self.radio_specific_levels.stateChanged.connect(self.radio_specific)
self.interference_save_img_btn.clicked.connect(self.save_image)

Color definition dictionary and previous project/design names
self.color_dict = {"green": [QtGui.QColor(125, 115, 202), '#7d73ca'],
 "yellow": [QtGui.QColor(211, 89, 162), '#d359a2'],
 "orange": [QtGui.QColor(255, 99, 97), '#ff6361'],
 "red": [QtGui.QColor(255, 166, 0), '#ffa600'],
 "white": [QtGui.QColor("white"), '#ffffff']}
self.previous_design = ''
self.previous_project = ''

Set the legend tables to stretch resize mode
header = self.protection_legend_table.horizontalHeader()
v_header = self.protection_legend_table.verticalHeader()

header.setSectionResizeMode(0, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(0, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(1, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(2, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(3, QtWidgets.QHeaderView.ResizeMode.Stretch)

header = self.interference_legend_table.horizontalHeader()
v_header = self.interference_legend_table.verticalHeader()

header.setSectionResizeMode(0, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(0, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(1, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(2, QtWidgets.QHeaderView.ResizeMode.Stretch)
v_header.setSectionResizeMode(3, QtWidgets.QHeaderView.ResizeMode.Stretch)

Input validation for protection level legend table
self.delegate = DoubleDelegate(decimals=2, values=values,
 max_power=1000, min_power=-200)
self.protection_legend_table.setItemDelegateForColumn(0, self.delegate)
self.open_file_dialog()

#####
Open file dialog and select project
#
~~~~~
Open the file dialog for project selection and populate the design dropdown
with all EMIT designs in the project.

def open_file_dialog(self):
 fname, _filter = QtWidgets.QFileDialog.getOpenFileName(self, "Select EMIT Project",
 "", "", "Ansys Electronics Desktop Files (*.aedt)",)
 if fname:
 self.file_path_box.setText(fname)

 # Close previous project and open specified one
if self.emitapp is not None:

```

(continues on next page)

(continued from previous page)

```

 self.emitapp.close_project()
 self.emitapp = None
 desktop_proj = desktop.load_project(self.file_path_box.text())

 # check for an empty project (i.e. no designs)
 if isinstance(desktop_proj, bool):
 self.file_path_box.setText("")
 msg = QtWidgets.QMessageBox()
 msg.setWindowTitle("Error: Project missing designs.")
 msg.setText(
 "The selected project has no designs. Projects must have at least "
 "one EMIT design. See AEDT log for more information.")
 x = msg.exec()
 return

 # Check if project is already open
 if desktop_proj.lock_file == None:
 msg = QtWidgets.QMessageBox()
 msg.setWindowTitle("Error: Project already open")
 msg.setText("Project is locked. Close or remove the lock before proceeding. See AEDT log for more information.")
 x = msg.exec()
 return

 # Populate design dropdown with all design names
 designs = desktop_proj.design_list
 emit_designs = []
 self.populating_dropdown = True
 self.design_name_dropdown.clear()
 self.populating_dropdown = False
 for d in designs:
 design_type = desktop.design_type(desktop_proj.project_name, d)
 if design_type == "EMIT":
 emit_designs.append(d)

 # add warning if no EMIT design
 # Note: this should never happen since loading a project without an EMIT
design
 # should add a blank EMIT design
 self.warning_label.setHidden(True)
 if len(emit_designs) == 0:
 self.warning_label.setText("Warning: The project must contain at least one EMIT design.")
 self.warning_label.setHidden(False)
 return

 self.populating_dropdown = True
 self.design_name_dropdown.addItems(emit_designs)
 self.populating_dropdown = False
 self.emitapp = get_pyaedt_app(desktop_proj.project_name, emit_designs[0])
 self.design_name_dropdown.setCurrentIndex(0)

```

(continues on next page)

(continued from previous page)

```

check for at least 2 radios
radios = self.emitapp.modeler.components.get_radios()
self.warning_label setHidden(True)
if len(radios) < 2:
 self.warning_label.setText("Warning: The selected design must contain at_
least two radios.")
 self.warning_label setHidden(False)

if self.radio_specific_levels.isEnabled():
 self.radio_specific_levels.setChecked(False)
 self.radio_dropdown.clear()
 self.radio_dropdown.setEnabled(False)
 self.protection_levels = {}
 values = [float(self.protection_legend_table.item(row, 0).text()) for_
row in range(self.protection_legend_table.rowCount())]
 self.protection_levels['Global'] = values

 self.radio_specific_levels.setEnabled(True)
 self.protection_results_btn.setEnabled(True)
 self.interference_results_btn.setEnabled(True)

#####
Change design selection
~~~~~
Refresh the warning messages when the selected design changes

def design_dropdown_changed(self):
 if self.populating_dropdown:
 # don't load design's on initial project load
 return
 design_name = self.design_name_dropdown.currentText()
 self.emitapp = get_pyaedt_app(self.emitapp.project_name, design_name)
 # check for at least 2 radios
 radios = self.emitapp.modeler.components.get_radios()
 self.warning_label setHidden(True)
 if len(radios) < 2:
 self.warning_label.setText("Warning: The selected design must contain at_
least two radios.")
 self.warning_label setHidden(False)

 # clear the table if the design is changed
 self.clear_table()

#####
Enable radio specific protection levels
~~~~~
Activate radio selection dropdown and initialize dictionary to store protection_
levels
when the radio-specific level dropdown is checked.

def radio_specific(self):
 self.radio_dropdown.setEnabled(self.radio_specific_levels.isChecked())

```

(continues on next page)

(continued from previous page)

```

self.radio_dropdown.clear()
if self.radio_dropdown.isEnabled():
 self.emitapp.set_active_design(self.design_name_dropdown.currentText())
 radios = self.emitapp.modeler.components.get_radios()
 values = [float(self.protection_legend_table.item(row, 0).text()) for row in
 range(self.protection_legend_table.rowCount())]
 for radio in radios:
 if radios[radio].has_rx_channels():
 self.protection_levels[radio] = values
 self.radio_dropdown.addItem(radio)
else:
 self.radio_dropdown.clear()
 values = [float(self.protection_legend_table.item(row, 0).text()) for row in
 range(self.protection_legend_table.rowCount())]
 self.protection_levels['Global'] = values
 self.global_protection_level = not self.radio_specific_levels.isChecked()

#####
Update legend table
~~~~~
Update shown legend table values when the radio dropdown value changes.

def radio_dropdown_changed(self):
 if self.radio_dropdown.isEnabled():
 self.changing = True
 for row in range(self.protection_legend_table.rowCount()):
 item = self.protection_legend_table.item(row, 0)
 item.setText(str(self.protection_levels[self.radio_dropdown.
currentText()][row]))
 self.changing = False
 # update the validator so min/max for each row is properly set
 values = [float(self.protection_legend_table.item(row, 0).text()) for row in
 range(self.protection_legend_table.rowCount())]
 self.delegate.update_values(values)

#####
Save legend table values
~~~~~
Save inputted radio protection level threshold values every time one is changed
in the legend table.

def table_changed(self):
 if self.changing == False:
 values = [float(self.protection_legend_table.item(row, 0).text()) for row in
 range(self.protection_legend_table.rowCount())]
 if self.radio_dropdown.currentText() == '':
 index = 'Global'
 else:
 index = self.radio_dropdown.currentText()
 self.protection_levels[index] = values
 self.delegate.update_values(values)

```

(continues on next page)

(continued from previous page)

```

#####
Save scenario matrix to as PNG file
#
~~~~~
Save the scenario matrix table as a PNG file.

def save_image(self):
 if self.tab_widget.currentIndex() == 0:
 table = self.protection_matrix
 else:
 table = self.interference_matrix

 fname, _filter = QtWidgets.QFileDialog.getSaveFileName(self, "Save Scenario_"
 ↵Matrix", "Scenario Matrix", "png (*.png)")
 if fname:
 image = QtGui.QImage(table.size(), QtGui.QImage.Format_ARGB32)
 table.render(image)
 image.save(fname)

#####
Save scenario matrix to Excel file
#
~~~~~
Write the scenario matrix results to an Excel file with color coding.

def save_results_excel(self):
 defaultName = ""
 if self.tab_widget.currentIndex() == 0:
 table = self.protection_matrix
 defaultName = "Protection Level Classification"
 else:
 table = self.interference_matrix
 defaultName = "Interference Type Classification"

 fname, _filter = QtWidgets.QFileDialog.getSaveFileName(self, "Save Scenario_"
 ↵Matrix", defaultName, "xlsx (*.xlsx)")

 if fname:
 workbook = openpyxl.Workbook()
 worksheet = workbook.active
 header = self.tx_radios[:]
 header.insert(0, "Tx/Rx")
 worksheet.append(header)
 for row in range(2, table.rowCount() + 2):
 worksheet.cell(row = row, column = 1, value = str(self.rx_radios[row - 2]))
 for col in range(2, table.columnCount() + 2):
 text = str(table.item(row - 2, col - 2).text())
 worksheet.cell(row = row, column = col, value = text)
 cell = worksheet.cell(row, col)
 cell.fill = PatternFill(start_color = self.color_dict[self.all_
 ↵colors[col - 2][row - 2]][1][1],
 end_color = self.color_dict[self.all_
 ↵colors[col - 2][row - 2]][1][1],
 fill_type = "solid"))

```

(continues on next page)

(continued from previous page)

```

workbook.save(fname)

#####
Run interference type simulation
~~~~~
Run interference type simulation and classify results.

def interference_results(self):
 # Initialize filter check marks and expected filter results
 self.interference_checks = [self.in_in_check.isChecked(), self.out_in_check.
 ↪isChecked(),
 self.in_out_check.isChecked(), self.out_out_check.
 ↪isChecked()]

 self.interference_filters = ["TxFundamental:In-band", ["TxHarmonic/Spurious:In-
 ↪band", "Intermod:In-band", "Broadband:In-band"],
 "TxFundamental:Out-of-band", ["TxHarmonic/
 ↪Spurious:Out-of-band", "Intermod:Out-of-band", "Broadband:Out-of-band"]]

 # Create list of problem types to analyze according to inputted filters
 filter = [i for (i,v) in zip(self.interference_filters, self.interference_
 ↪checks) if v]

 if self.file_path_box.text() != "" and self.design_name_dropdown.currentText() !=
 ↪"":
 if self.previous_design != self.design_name_dropdown.currentText() or self.
 ↪previous_project != self.file_path_box.text():
 self.previous_design = self.design_name_dropdown.currentText()
 self.previous_project = self.file_path_box.text()
 self.emitapp.set_active_design(self.design_name_dropdown.currentText())

 # Check if file is read-only
 if self.emitapp.save_project() == False:
 msg = QtWidgets.QMessageBox()
 msg.setWindowTitle("Writing Error")
 msg.setText("An error occurred while writing to the file. Is it_
 ↪readonly? Disk full? See AEDT log for more information.")
 x = msg.exec()
 return

 # Get results and radios
 self.rev = self.emitapp.results.analyze()
 self.tx_interferer = InterfererType().TRANSMITTERS
 self.rx_radios = self.rev.get_receiver_names()
 self.tx_radios = self.rev.get_interferer_names(self.tx_interferer)

 # Check if design is valid
 if self.tx_radios is None or self.rx_radios is None:
 return

 # Classify the interference
 # ~~~~~

```

(continues on next page)

(continued from previous page)

```

Iterate over all the transmitters and receivers and compute the power
at the input to each receiver due to each of the transmitters. Compute
which, if any, type of interference occurred.
domain = self.emitapp.results.interaction_domain()
self.all_colors, self.power_matrix = self.rev.interference_type_
→classification(domain, use_filter = True, filter_list = filter)

Save project and plot results on table widget
self.emitapp.save_project()
self.populate_table()

#####
Run protection level simulation
#
Run protection level simulation and classify results according to inputted
threshold levels.

def protection_results(self):
 # Initialize filter check marks and expected filter results
 self.protection_checks = [self.damage_check.isChecked(), self.overload_check.
→isChecked(),
 self.intermodulation_check.isChecked(), self.
→desensitization_check.isChecked()]

 self.protection_filters = ['damage', 'overload', 'intermodulation',
→'desensitization']

 filter = [i for (i,v) in zip(self.protection_filters, self.protection_checks) if_
→v]

 if self.file_path_box.text() != "" and self.design_name_dropdown.currentText() !=
→"":
 if self.previous_design != self.design_name_dropdown.currentText() or self.
→previous_project != self.file_path_box.text():
 self.previous_design = self.design_name_dropdown.currentText()
 self.previous_project = self.file_path_box.text()
 self.emitapp.set_active_design(self.design_name_dropdown.currentText())

 # Check if file is read-only
 if self.emitapp.save_project() == False:
 msg = QtWidgets.QMessageBox()
 msg.setWindowTitle("Writing Error")
 msg.setText("An error occurred while writing to the file. Is it_
→readonly? Disk full? See AEDT log for more information.")
 x = msg.exec()
 return

 # Get results and design radios
 self.tx_interferer = InterfererType().TRANSMITTERS
 self.rev = self.emitapp.results.analyze()
 self.rx_radios = self.rev.get_receiver_names()
 self.tx_radios = self.rev.get_interferer_names(self.tx_interferer)

```

(continues on next page)

(continued from previous page)

```

Check if there are radios in the design
if self.tx_radios is None or self.rx_radios is None:
 return

domain = self.emitapp.results.interaction_domain()
self.all_colors, self.power_matrix = self.rev.protection_level_
classification(domain,
 self.
 global_protection_level,
 self.
 protection_levels['Global'],
 self.
 protection_levels, use_filter = True,
 filter_.
 list = filter)

self.populate_table()

#####
Populate the scenario matrix
#
~~~~~
Create a scenario matrix view with the transmitters defined across the top
and receivers down the left-most column.

def populate_table(self):
 if self.tab_widget.currentIndex() == 0:
 table = self.protection_matrix
 button = self.protection_export_btn
 img_btn = self.protection_save_img_btn
 else:
 table = self.interference_matrix
 button = self.interference_export_btn
 img_btn = self.interference_save_img_btn

 num_cols = len(self.all_colors)
 num_rows = len(self.all_colors[0])
 table.setColumnCount(num_cols)
 table.setRowCount(num_rows)
 table.setVerticalHeaderLabels(self.rx_radios)
 table.setHorizontalHeaderLabels(self.tx_radios)

 for col in range(num_cols):
 for row in range(num_rows):
 item = QtWidgets.QTableWidgetItem(str(self.power_matrix[col][row]))
 table.setItem(row, col, item)
 cell = table.item(row, col)
 cell.setBackground(self.color_dict[self.all_colors[col][row]][0])

button.setEnabled(True)
img_btn.setEnabled(True)

```

(continues on next page)

(continued from previous page)

```

def clear_table(self):
 # get the table/buttons based on current tab
 if self.tab_widget.currentIndex() == 0:
 table = self.protection_matrix
 button = self.protection_export_btn
 img_btn = self.protection_save_img_btn
 else:
 table = self.interference_matrix
 button = self.interference_export_btn
 img_btn = self.interference_save_img_btn

 # disable export options
 button.setEnabled(False)
 img_btn.setEnabled(False)

 # clear the table
 table.setColumnCount(0)
 table.setRowCount(0)

#####
GUI closing event
#
Close AEDT if the GUI is closed.
def closeEvent(self, event):
 msg = QtWidgets.QMessageBox()
 msg.setWindowTitle("Closing GUI")
 msg.setText("Closing AEDT. Wait for the GUI to close on its own.")
 x = msg.exec()
 if self.emitapp:
 self.emitapp.close_project()
 self.emitapp.close_desktop()
 else:
 desktop.release_desktop(True, True)

```

```

C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 11236 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val

```

## Run GUI

Launch the GUI. If you want to run the GUI, uncomment the `window.show()` and `app.exec_()` method calls.

```
if __name__ == '__main__':
 app = QtWidgets.QApplication([])
 window = MainWindow()
 window.show()
 app.exec_()
else:
 desktop.release_desktop(True, True)
```

**Total running time of the script:** (0 minutes 22.297 seconds)

## 3.12.12 Twin Builder examples

These examples use PyAEDT to show some end-to-end workflows for Twin Builder. This includes schematic generation, setup, and postprocessing.

### Twin Builder: RC circuit design analysis

This example shows how you can use PyAEDT to create a Twin Builder design and run a Twin Builder time-domain simulation.

#### Perform required imports

Perform required imports.

```
import pyaedt
```

#### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

#### Select version and set launch options

Select the Twin Builder version and set the launch options. The following code launches Twin Builder 2023 R2 in graphical mode.

You can change the Boolean parameter `non_graphical` to `True` to launch Twin Builder in non-graphical mode. You can also change the Boolean parameter `new_thread` to `False` to launch Twin Builder in an existing AEDT session if one is running.

```
non_graphical = False
new_thread = True
```

## Launch Twin Builder

Launch Twin Builder using an implicit declaration and add a new design with a default setup.

```
tb = pyaedt.TwinBuilder(projectname=pyaedt.generate_unique_project_name(),
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=new_thread
)
tb.modeler.schematic_units = "mil"
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
 subprocess 2304 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create components for RC circuit

Create components for an RC circuit driven by a pulse voltage source. Create components, such as a voltage source, resistor, and capacitor.

```
source = tb.modeler.schematic.create_voltage_source("E1", "EPULSE", 10, 10, [0, 0])
resistor = tb.modeler.schematic.create_resistor("R1", 10000, [1000, 1000], 90)
capacitor = tb.modeler.schematic.create_capacitor("C1", 1e-6, [2000, 0])
```

## Create ground

Create a ground, which is needed for an analog analysis.

```
gnd = tb.modeler.components.create_gnd([0, -1000])
```

## Connect components

Connects components with pins.

```
source.pins[1].connect_to_component(resistor.pins[0])
resistor.pins[1].connect_to_component(capacitor.pins[0])
capacitor.pins[1].connect_to_component(source.pins[0])
source.pins[0].connect_to_component(gnd.pins[0])
```

```
(True, <pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object at
 0x0000025922346E30>, <pyaedt.modeler.circuits.object3dcircuit.CircuitComponent object
 at 0x0000025922346E90>)
```

## Parametrize transient setup

Parametrize the default transient setup by setting the end time.

```
tb.set_end_time("300ms")
```

```
True
```

## Solve transient setup

Solve the transient setup.

```
tb.analyze_setup("TR")
```

```
True
```

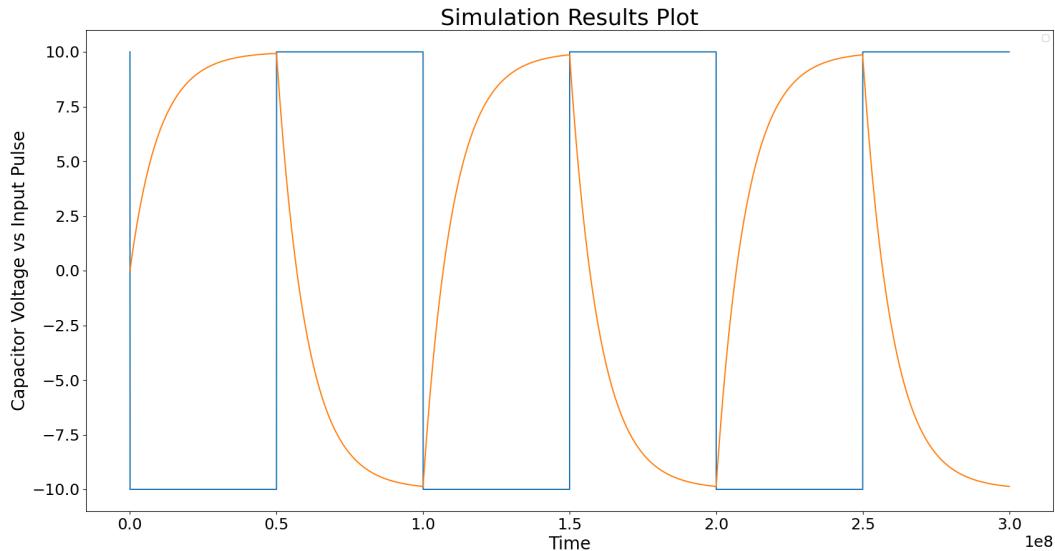
## Get report data and plot using Matplotlib

Get report data and plot it using Matplotlib. The following code gets and plots the values for the voltage on the pulse voltage source and the values for the voltage on the capacitor in the RC circuit.

```
E_Value = "E1.V"
C_Value = "C1.V"

x = tb.post.get_solution_data([E_Value, C_Value], "TR", "Time")
x.plot([E_Value, C_Value], x_label="Time", y_label="Capacitor Voltage vs Input Pulse")

tb.save_project()
```



```
No artists with labels found to put in legend. Note that artists whose label start with _
an underscore are ignored when legend() is called with no argument.
```

```
True
```

## Close Twin Builder

After the simulation completes, you can close Twin Builder or release it. All methods provide for saving the project before closing.

```
tb.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 54.020 seconds)

## Twin Builder: wiring a rectifier with a capacitor filter

This example shows how you can use PyAEDT to create a Twin Builder design and run a Twin Builder time-domain simulation.

### Perform required imports

Perform required imports.

```
import math
import matplotlib.pyplot as plt
import pyaedt
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Select version and set launch options

Select the Twin Builder version and set the launch options. The following code launches Twin Builder in graphical mode.

You can change the Boolean parameter `non_graphical` to `True` to launch Twin Builder in non-graphical mode. You can also change the Boolean parameter `new_thread` to `False` to launch Twin Builder in an existing AEDT session if one is running.

```
non_graphical = False
new_thread = True
```

## Launch Twin Builder

Launch Twin Builder using an implicit declaration and add a new design with a default setup.

```
tb = pyaedt.TwinBuilder(projectname=pyaedt.generate_unique_project_name(),
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=new_thread
)
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 5688 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 -py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 -log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create components for bridge rectifier

Create components for a bridge rectifier with a capacitor filter.

```
Define the grid distance for ease in calculations.

G = 0.00254

Create an AC sinusoidal source component.

source = tb.modeler.schematic.create_voltage_source("V_AC", "ESINE", 100, 50, [-1 * G, 0])

Create the four diodes of the bridge rectifier.

diode1 = tb.modeler.schematic.create_diode(location=[10 * G, 6 * G], angle=270)
diode2 = tb.modeler.schematic.create_diode(location=[20 * G, 6 * G], angle=270)
diode3 = tb.modeler.schematic.create_diode(location=[10 * G, -4 * G], angle=270)
diode4 = tb.modeler.schematic.create_diode(location=[20 * G, -4 * G], angle=270)

Create a capacitor filter.

capacitor = tb.modeler.schematic.create_capacitor(value=1e-6, location=[29 * G, -10 * G])

Create a load resistor.

resistor = tb.modeler.schematic.create_resistor(value=100000, location=[39 * G, -10 * G])

Create a ground.

gnd = tb.modeler.components.create_gnd(location=[5 * G, -16 * G])
```

## Connect components

Connect components with wires.

```
Wire the diode bridge.

tb.modeler.schematic.create_wire(points=[diode1.pins[0].location, diode3.pins[0].
 ↪location])
tb.modeler.schematic.create_wire(points=[diode2.pins[1].location, diode4.pins[1].
 ↪location])
tb.modeler.schematic.create_wire(points=[diode1.pins[1].location, diode2.pins[0].
 ↪location])
tb.modeler.schematic.create_wire(points=[diode3.pins[1].location, diode4.pins[0].
 ↪location])

Wire the AC source.

tb.modeler.schematic.create_wire(points=[source.pins[1].location, [0, 10 * G], [15 * G, ↩
 ↪10 * G], [15 * G, 5 * G]])
tb.modeler.schematic.create_wire(points=[source.pins[0].location, [0, -10 * G], [15 * G, ↩
 ↪-10 * G], [15 * G, -5 * G]])

Wire the filter capacitor and load resistor.

tb.modeler.schematic.create_wire(points=[resistor.pins[0].location, [40 * G, 0], [22 * G,
 ↪0]])
tb.modeler.schematic.create_wire(points=[capacitor.pins[0].location, [30 * G, 0]])

Wire the ground.

tb.modeler.schematic.create_wire(points=[resistor.pins[1].location, [40 * G, -15 * G], ↩
 ↪gnd.pins[0].location])
tb.modeler.schematic.create_wire(points=[capacitor.pins[1].location, [30 * G, -15 * G]])
tb.modeler.schematic.create_wire(points=[gnd.pins[0].location, [5 * G, 0], [8 * G, 0]])

Zoom to fit the schematic
tb.modeler.zoom_to_fit()
```

## Parametrize transient setup

Parametrize the default transient setup by setting the end time.

```
tb.set_end_time("100ms")
```

```
True
```

## Solve transient setup

Solve the transient setup.

```
tb.analyze_setup("TR")
```

```
True
```

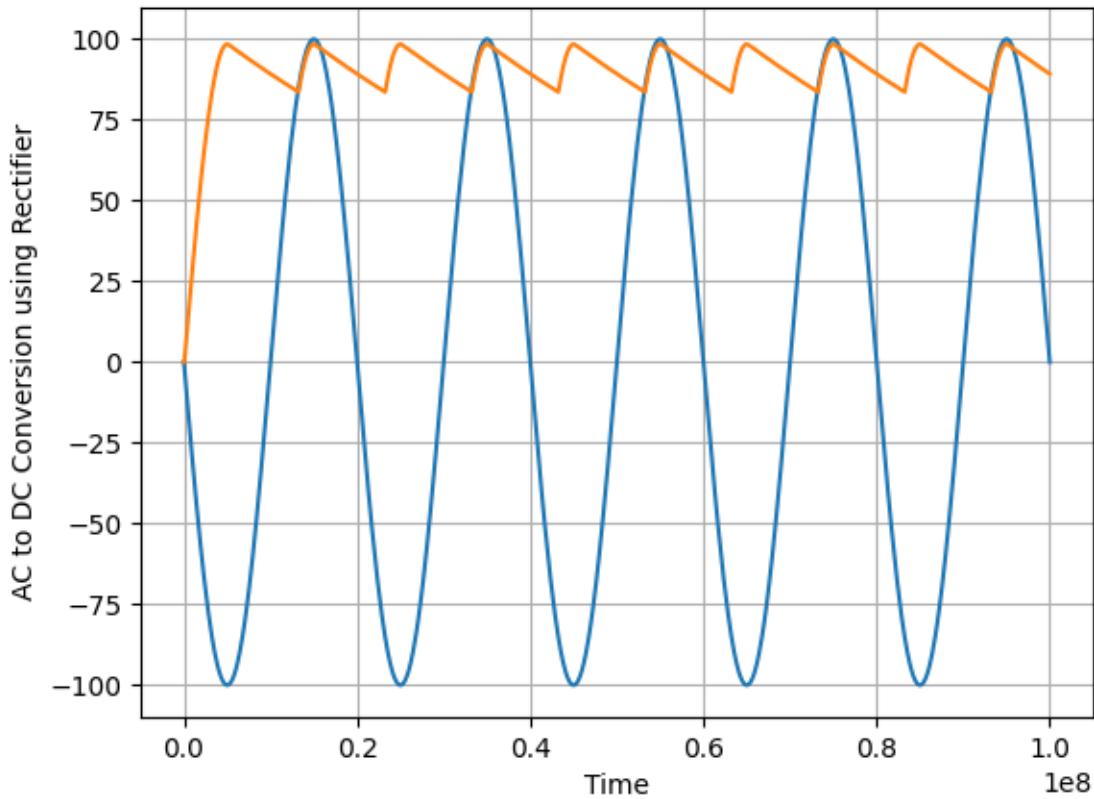
## Get report data and plot using Matplotlib

Get report data and plot it using Matplotlib. The following code gets and plots the values for the voltage on the pulse voltage source and the values for the voltage on the capacitor in the RC circuit.

```
E_Value = "V_AC.V"
x = tb.post.get_solution_data(E_Value, "TR", "Time")
plt.plot(x.intrinsics["Time"], x.data_real(E_Value))

R_Value = "R1.V"
x = tb.post.get_solution_data(R_Value, "TR", "Time")
plt.plot(x.intrinsics["Time"], x.data_real(R_Value))

plt.grid()
plt.xlabel("Time")
plt.ylabel("AC to DC Conversion using Rectifier")
plt.show()
```



## Close Twin Builder

After the simulation is completed, you can close Twin Builder or release it. All methods provide for saving the project before closing.

```
tb.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 54.449 seconds)

## Twin Builder: dynamic ROM creation and simulation (2023 R2 beta)

This example shows how you can use PyAEDT to create a dynamic ROM in Twin Builder and run a Twin Builder time-domain simulation.

---

**Note:** This example uses functionality only available in Twin Builder 2023 R2 and later. For 2023 R2, the build date must be 8/7/2022 or later.

---

## Perform required imports

Perform required imports.

```
import os
import shutil
import matplotlib.pyplot as plt
from pyaedt import TwinBuilder
from pyaedt import generate_unique_project_name
from pyaedt import generate_unique_folder_name
from pyaedt import downloads
from pyaedt import settings
```

## Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

## Select version and set launch options

Select the Twin Builder version and set launch options. The following code launches Twin Builder in graphical mode.

You can change the Boolean parameter `non_graphical` to `True` to launch Twin Builder in non-graphical mode. You can also change the Boolean parameter `new_thread` to `False` to launch Twin Builder in an existing AEDT session if one is running.

```
non_graphical = False
new_thread = True
```

## Set up input data

Define needed file name

```
source_snapshot_data_zipfilename = "Ex1_Mechanical_DynamicRom.zip"
source_build_conf_file = "dynarom_build.conf"

Download data from example_data repository
temp_folder = generate_unique_folder_name()
source_data_folder = downloads.download_twin_builder_data(source_snapshot_data_
 ↴zipfilename, True, temp_folder)
source_data_folder = downloads.download_twin_builder_data(source_build_conf_file, True, ↴
 ↴temp_folder)

Toggle these for local testing
source_data_folder = "D:\\Scratch\\TempDyn"

data_folder = os.path.join(source_data_folder, "Ex03")

Unzip training data and config file
```

(continues on next page)

(continued from previous page)

```
downloads.unzip(os.path.join(source_data_folder, source_snapshot_data_zipfilename), data_
 ↪_folder)
shutil.copyfile(os.path.join(source_data_folder, source_build_conf_file),
 os.path.join(data_folder, source_build_conf_file))
```

```
'D:\\Temp\\pyaedt_prj_USB\\twin_builder\\Ex03\\dynarom_build.conf'
```

## Launch Twin Builder and build ROM component

Launch Twin Builder using an implicit declaration and add a new design with a default setup for building the dynamic ROM component.

```
tb = TwinBuilder(projectname=generate_unique_project_name(),
 specified_version=aedt_version,
 non_graphical=non_graphical,
 new_desktop_session=new_thread)

Switch the current desktop configuration and the schematic environment to "Twin Builder".
The Dynamic ROM feature is only available with a twin builder license.
This and the restoring section at the end are not needed if the desktop is already
→configured as "Twin Builder".
current_desktop_config = tb._odesktop.GetDesktopConfiguration()
current_schematic_environment = tb._odesktop.GetSchematicEnvironment()
tb._odesktop.SetDesktopConfiguration("Twin Builder")
tb._odesktop.SetSchematicEnvironment(1)

Get the dynamic ROM builder object
rom_manager = tb._odesign.GetROMManager()
dynamic_rom_builder = rom_manager.GetDynamicROMBuilder()

Build the dynamic ROM with specified configuration file
conf_file_path = os.path.join(data_folder, source_build_conf_file)
dynamic_rom_builder.Build(conf_file_path.replace('\\\\', '/'))

Test if ROM was created successfully
dynamic_rom_path = os.path.join(data_folder, 'DynamicRom.dyn')
if os.path.exists(dynamic_rom_path):
 tb._odesign.AddMessage("Info", "path exists: {}".format(dynamic_rom_path.replace('\\\\',
 ↪', '/'), ""))
else:
 tb._odesign.AddMessage("Info", "path does not exist: {}".format(dynamic_rom_path), "")

Create the ROM component definition in Twin Builder
rom_manager.CreateROMComponent(dynamic_rom_path.replace('\\\\', '/'), 'dynarom')
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:
→subprocess 5616 is still running
 _warn("subprocess %s is still running" % self.pid,
```

(continues on next page)

(continued from previous page)

```
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create schematic

Place components to create a schematic.

```
Define the grid distance for ease in calculations
G = 0.00254

Place a dynamic ROM component
rom1 = tb.modeler.schematic.create_component("ROM1", "", "dynarom", [36 * G, 28 * G])

Place two excitation sources
source1 = tb.modeler.schematic.create_periodic_waveform_source(None, "PULSE", 190, 0.002,
 "300deg", 210, 0,
 [20 * G, 29 * G])
source2 = tb.modeler.schematic.create_periodic_waveform_source(None, "PULSE", 190, 0.002,
 "300deg", 210, 0,
 [20 * G, 25 * G])

Connect components with wires
tb.modeler.schematic.create_wire([[22 * G, 29 * G], [33 * G, 29 * G]])
tb.modeler.schematic.create_wire([[22 * G, 25 * G], [30 * G, 25 * G], [30 * G, 28 * G],
 [33 * G, 28 * G]])

Zoom to fit the schematic
tb.modeler.zoom_to_fit()
```

## Parametrize transient setup

Parametrize the default transient setup by setting the end time.

```
tb.set_end_time("1000s")
tb.set_hmin("1s")
tb.set_hmax("1s")
```

True

## Solve transient setup

Solve the transient setup.

```
tb.analyze_setup("TR")
```

```
True
```

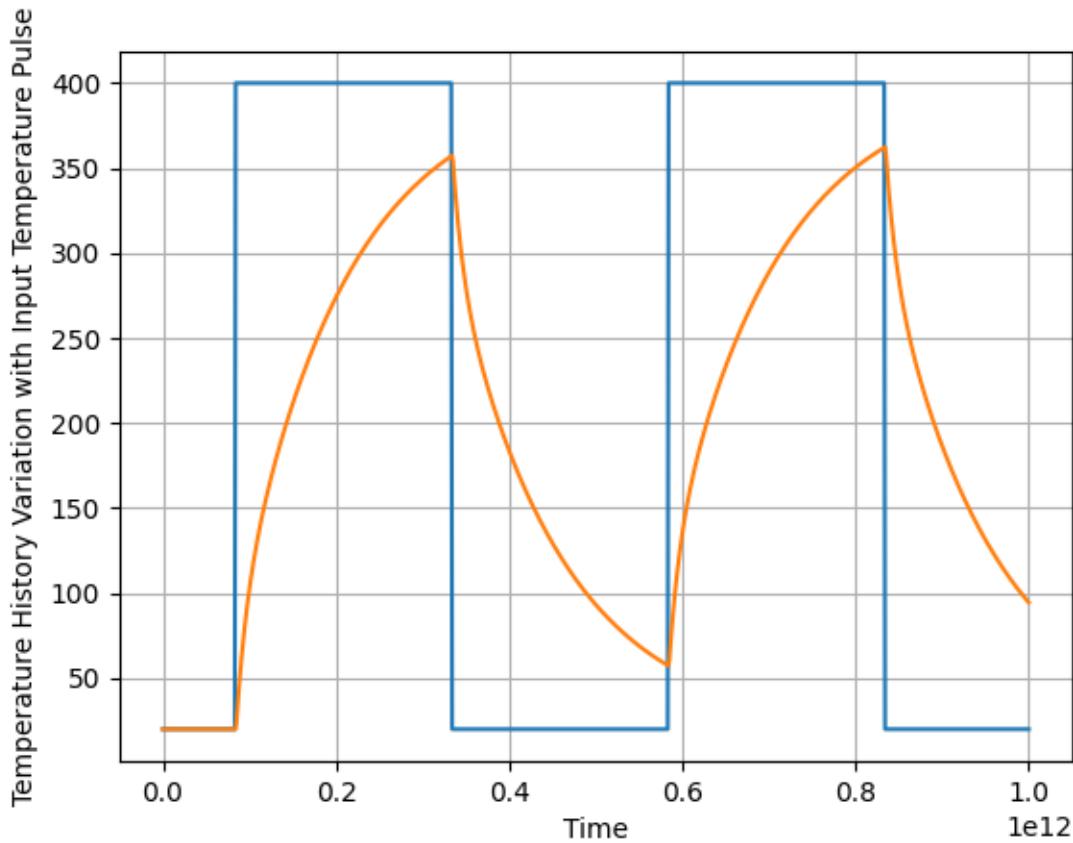
## Get report data and plot using Matplotlib

Get report data and plot it using Matplotlib. The following code gets and plots the values for the voltage on the pulse voltage source and the values for the output of the dynamic ROM.

```
input_excitation = "PULSE1.VAL"
x = tb.post.get_solution_data(input_excitation, "TR", "Time")
plt.plot(x.intrinsics["Time"], x.data_real(input_excitation))

output_temperature = "ROM1.Temperature_history"
x = tb.post.get_solution_data(output_temperature, "TR", "Time")
plt.plot(x.intrinsics["Time"], x.data_real(output_temperature))

plt.grid()
plt.xlabel("Time")
plt.ylabel("Temperature History Variation with Input Temperature Pulse")
plt.show()
```



## Close Twin Builder

After the simulation is completed, you can close Twin Builder or release it. All methods provide for saving the project before closing.

```
Clean up the downloaded data
shutil.rmtree(source_data_folder)

Restore earlier desktop configuration and schematic environment
tb._odesktop.SetDesktopConfiguration(current_desktop_config)
tb._odesktop.SetSchematicEnvironment(current_schematic_environment)

tb.release_desktop()
```

```
True
```

**Total running time of the script:** (1 minutes 10.815 seconds)

## Twin Builder: static ROM creation and simulation (2023 R2 beta)

This example shows how you can use PyAEDT to create a static ROM in Twin Builder and run a Twin Builder time-domain simulation.

---

**Note:** This example uses functionality only available in Twin Builder 2023 R2 and later. For 2023 R2, the build date must be 8/7/2022 or later.

---

### Perform required imports

Perform required imports.

```
import os
import math
import shutil
import matplotlib.pyplot as plt
from pyaedt import TwinBuilder
from pyaedt import generate_unique_project_name
from pyaedt import generate_unique_folder_name
from pyaedt import downloads
from pyaedt import settings
```

### Set AEDT version

Set AEDT version.

```
aedt_version = "2024.1"
```

### Select version and set launch options

Select the Twin Builder version and set launch options. The following code launches Twin Builder 2023 R2 in graphical mode.

You can change the Boolean parameter `non_graphical` to `True` to launch Twin Builder in non-graphical mode. You can also change the Boolean parameter `new_thread` to `False` to launch Twin Builder in an existing AEDT session if one is running.

```
non_graphical = False
new_thread = True
```

## Set up input data

Define needed file name

```
source_snapshot_data_zipfilename = "Ex1_Fluent_StaticRom.zip"
source_build_conf_file = "SROMbuild.conf"
source_props_conf_file = "SROM_props.conf"

Download data from example_data repository
source_data_folder = downloads.download_twin_builder_data(source_snapshot_data_
 ↴zipfilename, True)
source_data_folder = downloads.download_twin_builder_data(source_build_conf_file, True)
source_data_folder = downloads.download_twin_builder_data(source_props_conf_file, True)

Uncomment the following line for local testing
source_data_folder = "D:\\\\Scratch\\\\TempStatic"

data_folder = os.path.join(source_data_folder, "Ex04")

Unzip training data and config file
downloads.unzip(os.path.join(source_data_folder, source_snapshot_data_zipfilename), data_
 ↴folder)
shutil.copyfile(os.path.join(source_data_folder, source_build_conf_file),
 os.path.join(data_folder, source_build_conf_file))
shutil.copyfile(os.path.join(source_data_folder, source_props_conf_file),
 os.path.join(data_folder, source_props_conf_file))
```

'D:\\\\Temp\\\\PyAEDTEXamples\\\\twin\_builder\\\\Ex04\\\\SROM\_props.conf'

## Launch Twin Builder and build ROM component

Launch Twin Builder using an implicit declaration and add a new design with a default setup for building the static ROM component.

```
tb = TwinBuilder(projectname=generate_unique_project_name(), specified_version=aedt_
 ↴version,
 non_graphical=non_graphical, new_desktop_session=new_thread)

Switch the current desktop configuration and the schematic environment to "Twin Builder
 .
The Static ROM feature is only available with a twin builder license.
This and the restoring section at the end are not needed if the desktop is already
 configured as "Twin Builder".
current_desktop_config = tb._odesktop.GetDesktopConfiguration()
current_schematic_environment = tb._odesktop.GetSchematicEnvironment()
tb._odesktop.SetDesktopConfiguration("Twin Builder")
tb._odesktop.SetSchematicEnvironment(1)

Get the static ROM builder object
rom_manager = tb._odesign.GetROMManager()
static_rom_builder = rom_manager.GetStaticROMBuilder()
```

(continues on next page)

(continued from previous page)

```
Build the static ROM with specified configuration file
confpath = os.path.join(data_folder, source_build_conf_file)
static_rom_builder.Build(confpath.replace('\\', '/'))

Test if ROM was created successfully
static_rom_path = os.path.join(data_folder, 'StaticRom.rom')
if os.path.exists(static_rom_path):
 tb.logger.info("Built intermediate rom file successfully at: %s", static_rom_path)
else:
 tb.logger.error("Intermediate rom file not found at: %s", static_rom_path)

Create the ROM component definition in Twin Builder
rom_manager.CreateROMComponent(static_rom_path.replace('\\', '/'), 'staticrom')
```

```
C:\actions-runner_work_tool\Python\3.10.9\x64\lib\subprocess.py:1072: ResourceWarning:_
 subprocess 8116 is still running
 _warn("subprocess %s is still running" % self.pid,
C:\actions-runner_work\pyaedt\pyaedt\.venv\lib\site-packages\pyaedt\generic\settings.
 py:383: ResourceWarning: unclosed file <_io.TextIOWrapper name='D:\\Temp\\pyaedt_ansys.
 log' mode='a' encoding='cp1252'>
 self._logger = val
```

## Create schematic

Place components to create a schematic.

```
Define the grid distance for ease in calculations
G = 0.00254

Place a dynamic ROM component
rom1 = tb.modeler.schematic.create_component("ROM1", "", "staticrom", [40 * G, 25 * G])

Place two excitation sources
source1 = tb.modeler.schematic.create_periodic_waveform_source(None, "SINE", 2.5, 0.01, 0, 7.5, 0, [20 * G, 29 * G])
source2 = tb.modeler.schematic.create_periodic_waveform_source(None, "SINE", 50, 0.02, 0, 450, 0, [20 * G, 25 * G])

Connect components with wires

tb.modeler.schematic.create_wire([[22 * G, 29 * G], [33 * G, 29 * G]])
tb.modeler.schematic.create_wire([[22 * G, 25 * G], [30 * G, 25 * G], [30 * G, 28 * G], [33 * G, 28 * G]])

Enable storage of views

rom1.set_property("store_snapshots", 1)
rom1.set_property("view1_storage_period", "10s")
rom1.set_property("view2_storage_period", "10s")
```

(continues on next page)

(continued from previous page)

```
Zoom to fit the schematic
tb.modeler.zoom_to_fit()
```

## Parametrize transient setup

Parametrize the default transient setup by setting the end time.

```
tb.set_end_time("300s")
tb.set_hmin("1s")
tb.set_hmax("1s")
```

```
True
```

## Solve transient setup

Solve the transient setup. Skipping in case of documentation build.

```
if os.getenv("PYAEDT_DOC_GENERATION", "False") != "1":
 tb.analyze_setup("TR")
```

## Get report data and plot using Matplotlib

Get report data and plot it using Matplotlib. The following code gets and plots the values for the voltage on the pulse voltage source and the values for the output of the dynamic ROM.

```
if os.getenv("PYAEDT_DOC_GENERATION", "False") != "1":
 e_value = "ROM1.outfield_mode_1"
 x = tb.post.get_solution_data(e_value, "TR", "Time")
 x.plot()
 e_value = "ROM1.outfield_mode_2"
 x = tb.post.get_solution_data(e_value, "TR", "Time")
 x.plot()
 e_value = "SINE1.VAL"
 x = tb.post.get_solution_data(e_value, "TR", "Time")
 x.plot()
 e_value = "SINE2.VAL"
 x = tb.post.get_solution_data(e_value, "TR", "Time")
 x.plot()
```

## Close Twin Builder

After the simulation is completed, you can close Twin Builder or release it. All methods provide for saving the project before closing.

```
Clean up the downloaded data
shutil.rmtree(source_data_folder)

Restore earlier desktop configuration and schematic environment
tb._odesktop.SetDesktopConfiguration(current_desktop_config)
tb._odesktop.SetSchematicEnvironment(current_schematic_environment)

tb.release_desktop()
```

```
True
```

**Total running time of the script:** (0 minutes 43.695 seconds)

## PYTHON MODULE INDEX

p

pyaedt.generic.constants, ??