Table of Contents

1. Introduction

WIPL-D offers powerful, comprehensible, and easy to use scripting library. This scripting library allows users to access and modify symbols lists, to open and run WIPL-D suites, and to get hold of simulation results. The scripting language that has been used is Python 3.10. WIPL-D does not offer its own scripting editor, so an external Python interpreter should be used. We recommend Visual Studio Code, Anaconda, PyCharm, etc.

The WIPL-D package provides three different modules: interface, symbols, and results.

2. Installing WIPL-D python library

WIPL-D python library is distributed as a wheel (.whl) file. It is a standard format installation of Python distributions and contains all the files and metadata required for installation.

Please follow the next steps to install WIPL-D library:

- Download and unpack WIPL-D Python library from:

  http://server.wipl-d.com/download/wiplpy/

- Open Windows PowerShell in Administrator mode
- Install .whl file. For example, if you have downloaded wiplpy-1.0.0-py3-none-any.whl to the folder C:\Downloads\. Use the command below to install the whl package file.

  pip install C:\Downloads\wiplpy-1.0.0-py3-none-any.whl

Solutions of some problems:

- If users face any problem running the command in PowerShell regarding any 'Permission', they should close and reopen the PowerShell as Administrator.
- Sometimes, in the case of Python 3.x the user needs to use pip3 instead of pip, so if the user gets any error regarding the pip (except: 'Pip' is not recognized as an external or internal command, for this purpose the user needs to add the location of pip in the path inside system variable, or simply reinstall Python or Modify it and while doing so check the option ADD to Path).
- If pip or pip.exe is not recognized, check whether it is in the Python scripts directory, but the path of the script is not in the system variable. In that case, you should simply add the Python scripts path to the system variable PATH.

3. Interface Module (WiplInterface.py)

The interface module allows opening and running WIPL-D suites. It is possible to run WIPL-D Pro CAD or WIPL-D Pro suites. In addition, this module provides access to projects and project symbols.

3.1. Initialize WIPL-D Suite

Class InitializeWIPLDSuite(WIPLDInstallationDirectory, SuiteName="wipldprocad") – base class that provides methods for opening and running WIPL-D suites. It takes up to two parameters: *WIPLDInstallationDirectory* and *SuiteName*.

Parameters:

WIPLDInstallationDirectory – path to the WIPL-D installation directory.

SuiteName – name of the WIPL-D Software suite to be opened or run. Supported inputs are "**wipldprocad**" or "**wipldpro**". **The default value is** "**wipldprocad**".

The following methods are available:

Run(PathToProject) – opens, runs and closes WIPL-D suite for the specified project path. In order to run WIPL-D projects remotely on Windows or Linux server, the users should configure the server parameters in WIPL-D suite and set localization type to Run Remote Localization.

Parameters:

PathToProject – path to the project without extension.

Start(PathToProject) – starts WIPL-D suite and opens project.

Parameters:

PathToProject – path to the project without extension.

Close() – closes WIPL-D suite.

Example 1: Running WIPL-D Pro CAD suite

```
import wiplpy.WiplInterface
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
PathToProject="C:\\Microwave Pro v6.2\\3DEM\WCAD\\Examples\\Quick
Tour\\patch_antenna"
CAD=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory)
CAD.Run(PathToProject)
```

Example 2: Running WIPL-D Pro suite

```
import wiplpy.WiplInterface
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
PathToProject="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task2\\Intro_Task2"
PRO=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory,"w
ipldpro")
PRO.Run(PathToProject)
```

Example 3: Running WIPL-D Pro CAD suite remotely on a Linux server.

The first step is to configure Remote server. For more details, please refer to "Remote Run - User's Manual" (PDF) and the chapter "Configuring the Remote Server Accounts".

Then, set the machine where a simulation will be run. Go to Simulation tab in WIPL-D Pro CAD and select Run Localization. Select the Remote radio button, check the Run option on Default Server check box, and then click the Set Default Server button.

These steps are only necessary if WIPL-D project is run remotely for the first time.

Type the following code in python interpreter:

```
import wiplpy.WiplInterface
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
```

```
PathToProject="C:\\Microwave Pro v6.2\\3DEM\WCAD\\Examples\\Quick
Tour\\patch_antenna"
CAD=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory)
CAD.Run(PathToProject)
```

After a simulation is finalized on a remote server, simulation results are automatically downloaded to the client machine, and the project is deleted from the server.

3.2. Projects

Class WProject(PathToProject, SuiteName="wipldprocad") – this class implements *WSymbols* class. In the future, this class will also implement *WProjectSettings* class, *WGeometry* class, etc.

Parameters:

PathToProject – path to the project without extension

SuiteName – name of the WIPL-D Software to be opened, connected to, or run. Supported inputs are: "**wipldprocad**" or "**wipldpro**". **The default value is "wipldprocad".**

The following methods are available:

GetProjectName() – returns the project name without extension.

GetProjectPath() – returns the location of the project.

Example 1: Accessing the Symbols from WProject class

```
import wiplpy.WiplInterface
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
PathToProject="C:\\Microwave Pro
v6.2\\3DEM\\WCAD\\Training\\Models\\Intro 2 of 3 - Horn
Antenna\\Horn_full"
Project=wiplpy.WiplInterface.WProject(PathToProject)
```

```
Project.Symbols.PrintSymbols()
```

Results:

```
{'Name': 'A', 'Value': [60.0], 'Expression': 'None'}
{'Name': 'B', 'Value': [30.0], 'Expression': 'None'}
{'Name': 'Ahorn', 'Value': [120.0], 'Expression': 'None'}
{'Name': 'Bhorn', 'Value': [100.0], 'Expression': 'None'}
{'Name': 'Lwg', 'Value': [50.0], 'Expression': 'None'}
{'Name': 'Lhorn', 'Value': [50.0], 'Expression': 'None'}
```

4. Symbols (WSymbols.py)

In addition to the interface module that can provide access to project symbols, project symbols can also be accessed using a separate symbols module. No running instance of WIPL-D Suite is required for symbols access.

Class WSymbols(PathToSmb) – base symbols class.

Parameters:

PahtToSmb – path to the appropriate symbols file with extension (.wsmb for WIPL-D Pro CAD projects or .smb for WIPL-D Pro projects).

The following methods are available:

AddSymbolByName(symbname,symbexpression) – adds new symbol with symbol value or expression.

Parameters:

symbname – name of the symbol.

symbexpression – can be a numerical value or an expression.

SetSymbolByName(symbname,symbvalue) – changes symbol value by symbol name (changes symbol value in _Symbols dictionary).

Parameters:

symbname – name of the symbol.

symbvalue – new numerical value of the symbol.

GetSymbolByName(symbname) – returns symbol value for given symbol name (if there is no symbol with the given name, method returns minimum value that float can represent).

Parameters:

symbname – name of the symbol.

GetSymbolsCount() – returns the number of symbols in the list.

GetSymbolsName() – returns the list of all symbol names.

GetSymbolsValue() – returns the list of all symbol values.

GetSymbolsExpression() – returns list of all symbol expressions.

GetLastError() – returns the last error message if method returns non-zero value.

PrintSymbols() – prints all the symbols as a list of dictionary.

Example 1: Add symbols p1=1, p2=0.5 and p3=p1+p2 to the symbol list in WIPL-D Pro and print all symbols.

```
import wiplpy.WSymbols
PathToSMB="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task2\\Intro_Task2.smb"
SymbolsList=wiplpy.WSymbols.GetSymbols(PathToSMB)
SymbolsList.AddSymbolByName("p1",1)
SymbolsList.AddSymbolByName("p2",0.5)
SymbolsList.AddSymbolByName("p3","p1+p2")
SymbolsList.PrintSymbols()
```

Results:

```
{'Name':'B', 'Value': [60.0], 'Expression': 'None'}
{'Name':'A', 'Value': [30.0], 'Expression': 'None'}
{'Name':'Bhorn', 'Value': [150.0], 'Expression': 'None'}
{'Name':'Ahorn', 'Value': [120.0], 'Expression': 'None'}
{'Name':'Lhorn', 'Value': [50.0], 'Expression': 'None'}
{'Name':'Lwg', 'Value': [50.0], 'Expression': 'None'}
{'Name':'Hwire', 'Value': [24.0], 'Expression': '0.8*A'}
{'Name':'Rwire', 'Value': [0.8], 'Expression': 'Hwire/30'}
{'Name':'x2', 'Value': [25.0], 'Expression': 'Lwg/2'}
{'Name':'y2', 'Value': [30.0], 'Expression': 'B/2'}
{'Name':'z1', 'Value': [45.0], 'Expression': '(Ahorn-A)/2'}
{'Name':'z2', 'Value': [75.0], 'Expression': 'A+z1'}
{'Name':'x3', 'Value': [75.0], 'Expression': 'x2+Lhorn'}
{'Name':'y3', 'Value': [75.0], 'Expression': 'y2+(Bhorn- B)/2'}
{'Name': 'p1', 'Value': [1.0], 'Expression': 'None'}
```

```
{'Name': 'p2', 'Value': [0.5], 'Expression': 'None'}
{'Name': 'p3', 'Value': [1.5], 'Expression': 'p1+p2'}
```

Example 2: Get symbol “A”, change its value to 100 and print the new symbol value.

```
import wiplpy.WSymbols
PathToSMB="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task2\\Intro_Task2.smb"
SymbolsList=wiplpy.WSymbols.GetSymbols(PathToSMB)
Symb1=SymbolsList.GetSymbolByName("A")
print("The value of symbol A: ",Symb1)
SymbolsList.SetSymbolByName("A",100.0)
Symb1=SymbolsList.GetSymbolByName("A")
print("The new value of symbol A: ",Symb1)
```

Results:

```
The value of symbol A:[30.0]
The new value of symbol A:[100.0]
```

5. Results Module (WResults.py)

This module enables access to WIPL-D output results. It allows one-dimensional access to results. Two-dimensional results can be obtained using loop and one-dimensional results. No running instance of WIPL-D suite is required for results access.

5.1. Far Field

WFarField InitializeFFResults(NFFilePath) – this method provides an interface to access far field results.

Class WFarField(PathToProject) – this class reads far field output results.

Parameters:

PathToProject – path to the project without the extension.

The following methods are available:

GetFFSettings(self) – returns the far field (radiation) settings and does so in order: {excitation, frequency, number of phi angles, number of theta angles}. Settings are listed for each excitation and for each frequency.

GetNumberOfExcitations() – returns the number of excitations (waves or generators).

GetExcitation() – returns the list of excitations.

GetNumberOfFrequencies() – returns the number of frequency points.

GetFrequencies() – returns the list of frequency points.

GetNumberOfPhiPoints(igen=1) – returns the number of phi points. By default, it returns the number of phi points for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetPhiPoints(igen=1) – returns the list of phi points. By default, it returns phi points for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetNumberOfThetaPoints(igen=1) – returns the number of theta points. By default, it returns the number of theta points for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetThetaPoints(igen=1) – returns the list of theta points. By default, it returns theta points for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetXLabel() – returns the label of X-axis.

GetYLabel() – returns the label of Y-axis as combination of selected component and format.

GetXUnit() – returns the unit of X-axis.

GetYUnit() – returns the unit of Y-axis.

GetData(XaxisLabel,Cuts) – returns list of tuples for selected *XaxisLabel* and given *Cuts*.

Parameters:

XaxisLabel – can be set to: "**Phi**", "**Theta**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**Phi**": phi-angle value,

"**Theta**": theta-angle value,

"**Frequency**": frequency value,

"**Excitation**": index of excitation,

}

Please see the examples for more details.

GetXData(XaxisLabel) – returns X-axis list of doubles for selected *XaxisLabel*.

Parameters:

XaxisLabel – can be set to: "**Phi**", "**Theta**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

GetYData(ComponentName,FormatName,XaxisLabel,Cuts) – returns the list of doubles for selected *XaxisLabel*, given *Cuts*, *ComponentName* and *FormatName*.

Parameters:

ComponentName – can be set to: "**Total**", "**Phi-component**", "**Theta-component**", "**RC**", "**LC**", "**MaxLin**", "**MinLin**", "**Copolar**", "**Crosspolar**".

FormatName – differs for antenna and scatterer operation mode.

For antenna operation mode, it can be set to: "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**", "**Gain**", "**GaindB**", "**Ellipticity**", "**EllipticitydB**", "**PolarAngle**".

For scatterer operation mode, it can be set to: "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**", "**RCS**", "**RCSdB**", "**Ellipticity**", "**EllipticitydB**", "**PolarAngle**". RCS is calculated per lambda squared.

XaxisLabel – can be set to: "**Phi**", "**Theta**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**Phi**": phi-angle value,

"**Theta**": theta-angle value,

"**Frequency**": frequency value,

"**Excitation**": index of excitation,

}

GetLastError() – gets the last error message if method returns none.

Example 1: The following code runs WIPL-D Pro and returns total gain in dB at each frequency for phi=0.0 deg, theta=0.0 deg and excitation=1, so frequency is x-axis, while phi, theta, and excitation are cuts.

```
import wiplpy.WResults
import wiplpy.WiplInterface
ProjectPath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
# runs WIPL-D Pro
pro=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory,"w
ipldpro")
pro.Run(ProjectPath)
# gets the results after simulation
ff=wiplpy.WResults.InitializeFFResults(ProjectPath)
cuts={"Phi":0.0,
       "Theta":0.0,
       "Excitation":1
}
XaxisLabel="Frequency"
Component="Total"
Format="GaindB"
resx=ff.GetXData(XaxisLabel)
resy=ff.GetYData(Component,Format,XaxisLabel,cuts)
print("X-axis in "+ff.GetXUnit()+":")
print(resx)
print("Y-axis in "+ff.GetYUnit()+":")
print(resy)
```

Results:

```
X-axis in GHz:
[2.0, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5, 2.55, 2.6,
2.65, 2.7, 2.75, 2.8]
Y-axis in dB:
[9.3019035744063, 9.3278623694337, 9.4152101593585, 9.59326983772109,
9.85261236844315, 10.1659250926954, 10.4955104432321, 10.8088338760301,
```

```
11.1089264483652, 11.4608062371566, 11.9517437243678, 12.4410728126998,
12.1439249471145, 10.864757081776, 8.36679088859799, 4.79903593910454,
-2.02258456398004]
```

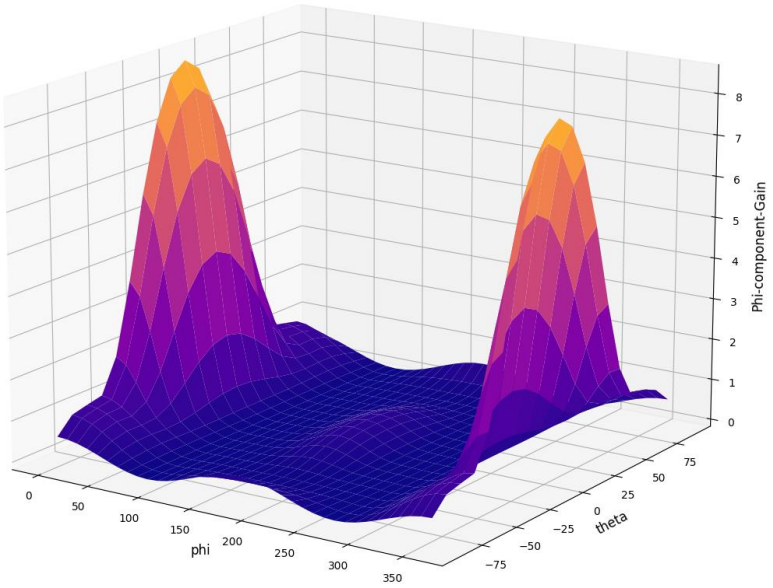
Example 2: This example plots 3D gain of Phi-component of far field at first frequency. For each theta cut (0, ntheta-1) in the loop, *GetYData* returns the list of doubles along phi axis. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
import numpy as np
import matplotlib.pyplot as plt
FFFilePath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
ff=wiplpy.WResults.InitializeFFResults(FFFilePath)
ntheta=ff.GetNumberOfThetaPoints()
XaxisLabel="Phi"
Component="Phi-component"
Format="Gain"
res_3D=[]
for itheta in range(0,ntheta):
  cuts={
     "Frequency":ff.GetFrequencies()[0],
     "Excitation": 1,
     "Theta":ff.GetThetaPoints()[itheta]
  }
  res_3D.append(list(ff.GetYData(Component,Format,XaxisLabel,cuts)))

phi=np.array(ff.GetPhiPoints())
theta=np.array(ff.GetThetaPoints())
X,Y=np.meshgrid(phi,theta)
Z=np.array(res_3D)
font={'size'   : 12}
fig=plt.figure()
ax=plt.axes(projection='3d')
ax.plot_surface(X,Y,Z, rstride=1, cstride=1,
                cmap='plasma', edgecolor='none',vmin=0,vmax=10)
ax.set_xlabel('phi',**font)
```

```
ax.set_ylabel('theta',**font)
ax.set_zlabel('Phi-component-Gain',**font)
```

Results:



Example 3: The next example returns the blocks of far field data at different frequency points. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
FFFilePath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
ff=wiplpy.WResults.InitializeFFResults(FFFilePath)
nphi=ff.GetNumberOfPhiPoints()
ntheta=ff.GetNumberOfThetaPoints()
cuts={ "Phi":ff.GetPhiPoints()[nphi-1],
       "Theta":ff.GetThetaPoints()[ntheta-1],
       "Excitation":1
}
XaxisLabel="Frequency"
resx=ff.GetXData(XaxisLabel)
```

```
res=ff.GetData(XaxisLabel,cuts)
print(res)
```

A singe block of far field is given as a tuple in the following format:
“Phi, Theta, Ephi_real, Ephi_imag, Etheta_real, Etheta_imag, Gain, GaindB”, where:
Phi, Theta - space angles
Ephi_real: real part of electrical field phi component
Ephi_imag: imaginary part of electrical field phi component
Etheta_real: real part of electrical field theta component
Etheta_imag: imaginary part of electrical field theta component

5.2. Near Field

WNearField InitializeNFResults(NFFilePath) – this method provides an interface to access near field results.

Class WNearField(PathToProject) – this class reads near field output results.

Parameters:

PathToProject – path to the project without the extension.

The following methods are available:

GetNFSettings() – returns the near field settings and does so in order: {excitation, frequency, number of X points, number of Y points, number of Z points}. Settings are listed for each excitation and for each frequency.

GetNumberOfExcitations() – returns the number of excitations (waves or generators).

GetExcitation() – returns the list of excitations.

GetNumberOfFrequencies() – returns the number of frequency points.

GetFrequencies() – returns the list of frequency points.

GetNumberOfXPoints(igen=1) – returns the number of X coordinates. By default, it returns the number of X coordinates for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetXPoints(igen=1) – returns the list of X coordinates. By default, it returns X coordinates for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetNumberOfYPoints(igen=1) – returns the number of Y coordinates. By default, it returns the number of Y coordinates for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetYPoints(igen=1) – returns the list of Y coordinates. By default, it returns Y coordinates for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetNumberOfZPoints(igen=1) – returns the number of Z coordinates. By default, it returns the number of Z coordinates for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetZPoints(igen=1) – returns the list of Z coordinates. By default, it returns Z coordinates for the first excitation.

Parameters:

igen – ordinal number of the excitation.

GetXLabel() – returns the label of X-axis.

GetYLabel() – returns the label of Y-axis as combination of selected component and format.

GetXUnit() – returns the unit of X-axis.

GetYUnit() – returns the unit of Y-axis.

GetData(XaxisLabel,Cuts) – returns list of tuples for selected *XaxisLabel* and given *Cuts*.

Parameters:

XaxisLabel – can be set to: "**X**", "**Y**", "**Z**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**X**": x-coordinate value

"**Y**": y-coordinate value

"**Z**": z-coordinate value

"**Frequency**": frequency value

"**Excitation**": index of excitation

}

Please see the examples for more details.

GetXData(XaxisLabel) – returns X-axis values as list of doubles for selected *XaxisLabel*

Parameters:

XaxisLabel – can be set to: "**X**", "**Y**", "**Z**", " **Frequency** ", " **Excitation**". An axis is a variable along which the results can be obtained.

GetYData(ComponentName,FormatName,XaxisLabel,Cuts) – returns the list of doubles for selected *XaxisLabel*, given *Cuts*, *ComponentName* and *FormatName*

Parameters:

ComponentName- can be set to: "**Etotal**", "**Ex**", "**Ey**", "**Ez**", "**Htotal**", "**Hx**", "**Hy**", "**Hz**", "**Ptotal**", "**Px**", "**Py**", "**Pz**", or "**SAR**".

FormatName- can be set to: "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**" or "**dBu**"

XaxisLabel – can be set to: "**X**", "**Y**", "**Z**", " **Frequency** ", " **Excitation**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**X**": x-coordinate value,

"**Y**": y-coordinate value,

"**Z**": z-coordinate value,

"**Frequency**": frequency value,

"**Excitation**": index of excitation,

}

GetLastError() – gets the last error message if method returns none.

Example 1: The following code returns the magnitude of total electric field at each frequency for X=-0.099 m, Y=-0.049 m, Z=0.0 m and for the first generator, so frequency is x-axis, while X, Y, Z and Excitation are cuts. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
NFFilePath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
nf=wiplpy.WResults.InitializeNFResults(NFFilePath)
cuts={ "X":-0.099,
       "Y":-0.049,
       "Z":0.0,
       "Excitation":1
}
XaxisLabel="Frequency"
Component="Etotal"
Format="Mag"
resx=nf.GetXData(XaxisLabel)
resy=nf.GetYData(Component,Format,XaxisLabel,cuts)
print("X-axis in "+nf.GetXUnit()+":")
print(resx)
print("Y-axis in "+nf.GetYUnit()+":")
print(resy)
```

Results:

```
X-axis in GHz:
[2.0, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5,……]
Y-axis in V/m:
[1.2704016826924827, 1.0684094738573944, 1.0717432590335862,
1.3523359654762264, 1.890060863448369,……]
```

Example 2: The next code gets the magniude of Ex component of near field along X-coordinate for Y=-0.049 m, Z=0.0 m, at the last frequency point and for the first excitation. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
NFFilePath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
nf=wiplpy.WResults.InitializeNFResults(NFFilePath)
nfreq=nf.GetNumberOfFrequencies()
cuts={ "Y":-0.049,
       "Z":0.0,
       "Frequency":nf.GetFrequencies()[nfreq-1],
       "Excitation":1
}
XaxisLabel="X"
Component="Ex"
Format="Mag"
resx=nf.GetXData(XaxisLabel)
resy=nf.GetYData(Component,Format,XaxisLabel,cuts)
print("X-axis in "+nf.GetXUnit()+":")
print(resx)
print("Y-axis in "+nf.GetYUnit()+":")
print(resy)
```

Results:

```
X-axis in MHz:
[-0.099,-0.097,-0.095,-0.093,-0.091,-0.089,-0.087,-0.085,-0.083,……]
Y-axis in V/m:
[0.8031708451103209,0.8215755749516609,0.8405627872933193,0.8601549555
789877, 0.8803753303295189, 0.9012479148094541, 0.9227974301801674,……]
```

Example 3: This example returns the blocks of near field data at different frequency points. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
NFFilePath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
nf=wiplpy.WResults.InitializeNFResults(NFFilePath)
nx=nf.GetNumberOfXPoints()
ny=nf.GetNumberOfYPoints()
nz=nf.GetNumberOfZPoints()
cuts={ "X":nf.GetXPoints()[nx-1],
       "Y":nf.GetYPoints()[ny-1],
       "Z":nf.GetZPoints()[nz-1],
       "Excitation":1
}
XaxisLabel="Frequency"
resx=nf.GetXData(XaxisLabel)
res=nf.GetData(XaxisLabel,cuts)
print(res)
```

A singe block of near field is given as a tuple in the following format:
“x, y, z, Ex-real, Ex-imag, Ey-real, Ey-imag, Ez-real, Ez-imag, Hx-real, Hx-imag, Hy-real, Hy-imag, Hz-real, Hz-imag”,
where:
x, y, z: coordinates
Ex-real: real part of electrical field x component
Ex-imag: imaginary part of electrical field x component
Ey-real: real part of electrical field y component
Ey-imag: imaginary part of electrical field y component
Ez-real: real part of electrical field z component
Ez-imag: imaginary part of electrical field z component
Hx-real: real part of magnetic field x component
Hx-imag: imaginary part of magnetic field x component
Hy-real: real part of magnetic field y component
Hy-imag: imaginary part of magnetic field y component
Hz-real: real part of magnetic field z component
Hz-imag: imaginary part of magnetic field z component

5.3. YZS - parameters

WYZS InitializeYZSResults(YZSFilePath) – this method provides an interface to access YZS results.

Class WYZS(PathToProject) – this class reads YZS output results.

Parameters:

PathToProject – path to the project without the extension.

The following methods are available:

GetYZSSettings(self) – returns the YZS settings and does so in order: {frequency, i (first port parameter), j (second port parameter)}. Settings are listed for each frequency.

GetNumberOfFrequencies() – returns the number of frequency points.

GetFrequencies() – returns the number of frequency points.

GetNumberOfRows() – returns the number of rows (i – first port parameter) in the circuit parameter matrix.

GetRows() – returns the list of rows (i – first port parameter) in the circuit parameter matrix.

GetNumberOfColumns() – returns the number of columns (j – second port parameter) in the circuit parameter matrix.

GetColumns() – returns the list of columns (j – second port parameter) in the circuit parameter matrix.

GetXLabel() – returns the label of X-axis.

GetYLabel() – returns the label of Y-axis as combination of selected component and format.

GetXUnit() – returns the unit of X-axis.

GetYUnit() – returns the unit of Y-axis.

GetData(XaxisLabel,Cuts) – returns list of tuples for selected *XaxisLabel* and given *Cuts*.

Parameters:

XaxisLabel – can be set to: "**Frequency**", "**i**", "**j**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**i**": first port parameter,

"**j**": second port parameter,

"**Frequency**": frequency value

}

GetXData(XaxisLabel) – returns X-axis values as list of doubles for selected *XaxisLabel.*

Parameters:

XaxisLabel – can be set to: "**Frequency**", "**i**", "**j**". An axis is a variable along which the results can be obtained.

GetYData(ComponentName,FormatName,XaxisLabel,Cuts) – returns the list of doubles for selected *XaxisLabel*, given *Cuts*, *ComponentName* and *FormatName.*

Parameters:

ParameterName – can be set to: "**Admittance**", "**Impedance**", "**Sparameter**".

FormatName – can be set to: "**RI**", "**Re**", "**Im**", "**Mag**", "**MagdB**", "**Phase**", "**VSWR**".

XaxisLabel – can be set to: "**Frequency**", "**i**", "**j**". An axis is a variable along which the results can be obtained.

Cuts={"**i**": first port parameter,

"**j**": second port parameter,

"**Frequency**": frequency value

}

GetLastError() – gets the last error message if method returns none.

Example 1: The first example returns *s*11 parameters in dB at different frequency points, so *XaxisLabel* is frequency, while i=1 and j=1 are cuts. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
YZSFilePath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Eval
Task6\\Slot-Fed_Microstrip_Patch"
yzs=wiplpy.WResults.InitializeYZSResults(YZSFilePath)
cuts={"i":1,
      "j":1,
}
XaxisLabel="Frequency"
Component="Sparameter"
Format="MagdB"
resx=yzs.GetXData(XaxisLabel)
res=yzs.GetYData(Component,Format,XaxisLabel,cuts)
print("X-axis in: "+yzs.GetXUnit())
print(resx)
print("Y-axis in: "+yzs.GetYUnit())
print(res)
```

Results:

```
X-axis in: GHz
[8.7,  8.86666666666667,  9.03333333333333,  9.2,  9.36666666666667,
9.53333333333333, 9.7]
Y-axis in: dB
[-1.6765501060674,-2.2691807321912,-3.4796614176146,-6.51421270590,
-19.094504492666, -9.617978281373, -4.002216754721]
```

Example 2: This code runs WIPL-D Pro and returns VSWR at different frequency points.

```
import wiplpy.WResults
import wiplpy.WiplInterface
```

```
ProjectPath="C:\\Microwave Pro v6.2\\3DEM\\Training\\Models\\Intro
Task1\\Intro_Task1"
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
# runs WIPL-D Pro
pro=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory,"w
ipldpro")
pro.Run(ProjectPath)
# gets the results after simulation
yzs=wiplpy.WResults.InitializeYZSResults(ProjectPath)
cuts={ "i":1,
       "j":1,
}
XaxisLabel="Frequency"
Component="Sparameter"
Format="VSWR"
resx=yzs.GetXData(XaxisLabel)
res=yzs.GetYData(Component,Format,XaxisLabel,cuts)
print("X-axis in: "+yzs.GetXUnit())
print(resx)
print("Y-axis in: "+yzs.GetYUnit())
print(res)
```

Results:

```
X-axis in: GHz
[2.0, 2.05, 2.1, 2.15, 2.2, 2.25, 2.3, 2.35, 2.4, 2.45, 2.5, 2.55, 2.6,
2.65, 2.7, 2.75, 2.8]
Y-axis in: U
[10.383668659778,   6.39885673223,   4.358280185707,   3.1512560396157,
2.332496083724,   1.7283482724395,   1.2841683793807,   1.0516905383541,
1.2212948688781,   1.227441909586,   1.1608235564434,   1.9643597325994,
2.9749126971224,   1.3231222881629,   8.166327331174,   65.7062461540,
37.77790346230]
```

Example 3: This code runs WIPL-D Pro CAD and *s*21 parameters in dB at different frequency points, so *XaxisLabel* is frequency, while i=2 and j=1 are cuts.

```
import wiplpy.WResults
import wiplpy.WiplInterface
```

```
ProjectPath="C:\\Microwave Pro
v6.2\\3DEM\\WCAD\\Training\\Models\\Evaluation 3 of 6 - 3-Port
Wilkinson Divider\\Wilkinson"
WIPLDInstallDirectory="C:\Microwave Pro v6.2"
# runs WIPL-D Pro CAD
pro=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory,"w
ipldprocad")
pro.Run(ProjectPath)
# gets the results after simulation
yzs=wiplpy.WResults.InitializeYZSResults(ProjectPath)
cuts={ "i":2,
       "j":1,
}
XaxisLabel="Frequency"
Component="Sparameter"
Format="MagdB"
resx=yzs.GetXData(XaxisLabel)
res=yzs.GetYData(Component,Format,XaxisLabel,cuts)
print("X-axis in: "+yzs.GetXUnit())
print(resx)
print("Y-axis in: "+yzs.GetYUnit())
print(res)
```

Results:

```
X-axis in: GHz
[22.0, 22.5, 23.0, 23.5, 24.0, 24.5, 25.0, 25.5, 26.0]
Y-axis in: dB
[-3.5143792407887,-3.51668859895,-3.5138184327278,-3.50705735586643,-
3.4994145510442,-3.493048343711,-3.48298911931769, -3.4682967494383, -
3.4596281351260]
```

5.4. Currents

WCurrent InitializeCUResults(CUFilePath,Current="Electric") – this method provides an interface to access electric or magnetic currents. **The default value for *Current* is "Electric". "Magnetic" is the alternative input.**

Class WCurrent(PathToProject,Current) – this class reads electric or magnetic currents.

Parameters:

PathToProject – path to the project without the extension.

Current – can be "**Electric**" or "**Magnetic**". **The default value is** "**Electric**".

The following methods are available:

GetCUSettings(self) – returns the currents settings and does so in order: {element, element number, excitation, frequency, np, ns}. Settings are listed for each excitation and for each frequency.

GetNumberOfExcitations() – returns the number of excitations (waves or generators).

GetExcitation() – returns the list of excitations (waves or generators).

GetNumberOfFrequencies() – returns the number of frequency points.

GetFrequencies() – returns the list of frequency points.

GetNumberOfElements(Frequency,Excitation) – returns the number of elements (wires + plates) for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetElements(Frequency,Excitation) – returns list of elements (wires + plates) for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetNumberOfWires(Frequency,Excitation) – returns the number of wires for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetWires(Frequency,Excitation) – returns the list of wires for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetNumberOfStrips(Frequency,Excitation) – returns the number of strips (WIPL-D 2D solver) for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetStrips(Frequency,Excitation) – returns the list of strips (WIPL-D 2D solver) for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetNumberOfPlates(Frequency,Excitation) – returns the number of plates for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetPlates(Frequency,Excitation) – returns the list of plates for given frequency and excitation.

Parameters:

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetNumberOfPCoordinates(Elem,ElemNumber,Frequency,Excitation) – returns the number of P coordinates for given element, frequency, and excitation.

Parameters:

Elem – "**Wire**", "**Plate**" or "**Strip**" (2D).

ElemNumber – ordinal number of the element.

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetPCoordinates(Elem,ElemNumber, Frequency,Excitation) – returns the list of P coordinates for given element, frequency, and excitation.

Parameters:

Elem – "**Wire**", "**Plate**" or "**Strip**" (2D).

ElemNumber – ordinal number of the element.

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetNumberOfSCoordinates(Elem,ElemNumber,Frequency,Excitation) – returns the number of S coordinates for given element, frequency, and excitation.

Parameters:

Elem – "**Wire**", "**Plate**" or "**Strip**" (2D).

ElemNumber – ordinal number of the element.

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetSCoordinates(Elem,ElemNumber, Frequency,Excitation) – returns the list of S coordinates for given element, frequency, and excitation.

Parameters:

Elem – "**Wire**", "**Plate**" or "**Strip**" (2D).

ElemNumber – ordinal number of the element.

Frequency – frequency of interest.

Excitation – ordinal number of the excitation of interest.

GetXLabel() – returns the label of X-axis.

GetYLabel() – returns the label of Y-axis as combination of component and format.

GetXUnit() – returns the unit of X-axis.

GetYUnit() – returns the unit of Y-axis.

GetData(XaxisLabel,Cuts) – returns list of tuples for selected *XaxisLabel* and given *Cuts*.

XaxisLabel – can be set to: "**P**", "**S**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**P**": p-coordinate value

"**S**": s-coordinate value

"**Frequency**": frequency value

"**Excitation**": index of excitation

}

Please see the examples for more details.

GetXData(XaxisLabel,Elements) – returns X-axis list of doubles for selected *XaxisLabel*

Parameters:

XaxisLabel – can be set to: "**P**", "**S**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

Elements – can be "**AllElements**" or "**Wire number**" or "**Plate number**". If *"Elements"* is set to "**AllElements**", the method returns the results for all wires and plates. If *"Elements" is* **"Wire 1"**, the method returns the results for the first wire.

GetYData(ComponentName,FormatName,XaxisLabel,Cuts,Elements) – returns the list of doubles for selected *XaxisLabel*, given *Cuts*, *ComponentName*, *FormatName and Elements*.

Parameters:

ComponentName – can be set to: "**I**", "**J**", "**Jp**", "**Js**".

FormatName – can be set to: "**RI**", "**Re**", "**Im**", "**Mag**", "**MagdB**", "**Phase**".

XaxisLabel – can be set to: "**P**", "**S**", "**Frequency**", "**Excitation**". An axis is a variable along which the results can be obtained.

Cuts – represents the fixed values for which the results are obtained. A variable that is selected to be *XaxisLabel* cannot be *Cuts* at the same time. *Cuts* is defined as dictionary with the following data:

Cuts={"**P**": p-coordinate value

"**S**": s-coordinate value

"**Frequency**": frequency value

"**Excitation**": index of excitation

}

Elements – can be "**AllElements**" or "**Wire number**" or "**Plate number**". If *"Elements"* is set to "**AllElements**", the method returns results for all wires and plates. If *"Elements" is* **"Plate 1"**, the method returns the results for the first plate.

GetLastError() – gets the last error message if method returns none.

Example 1: This code runs WIPL-D Pro and returns magnitude of total density of surface currents (J) for a single plate (plate number is 24), along P-coordinate for S=-1, Excitation=1 and Frequency=250 MHz.

```
import wiplpy.WResults
import wiplpy.WiplInterface

ProjectPath="C:\\Microwave Pro v6.2\\3DEM\\Tutorial\\Demo223"
WIPLDInstallDirectory="C:\Microwave Pro v6.2"

# runs WIPL-D Pro
pro=wiplpy.WiplInterface.InitializeWIPLDSuite(WIPLDInstallDirectory,"w
ipldpro")
pro.Run(ProjectPath)

# gets the results after simulation
cu=wiplpy.WResults.InitializeCUResults(ProjectPath)
cuts={ "S":-1,
```

```
       "Excitation":1,
       "Frequency": cu.GetFrequencies()[0],
}
XaxisLabel="P"
Component="J"
Format="Mag"
Element="Plate 24"
resx=cu.GetXData(XaxisLabel,Element)
res=cu.GetYData(Component,Format,XaxisLabel,cuts,Element)
print("X-axis in: "+cu.GetXUnit())
print(resx)
print("Y-axis in: "+cu.GetYUnit())
print(res)
```

Results:

```
X-axis in: U
[-1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0.0, 0.1,
0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
Y-axis in: A/m
[{'Element':     'Plate24',     'Results':     [0.001264994652746435,
0.0011715528718554602,  0.0010872732248887551,  0.0010119017265207122,
0.0009450750418675938,  0.0008863095406020984,  0.0008350009019815313,
0.0007904381104293755,  0.0007518337029886083,  0.0007183692710614943,
0.0006892525030317601,  0.0006637805453109519,  0.0006414045915306693,
0.0006217918664054711,  0.0006048822972566464,  0.0005909366869874243,
0.0005805699942502693,  0.0005747574015038555,  0.0005747947532920678,
0.000582195217997782, 0.0005985194716298451]}]
```

Example 2: The next code returns P-component of surface currents (Jp) in dB for all elements (plate 23 and plate 24), along P-coordinate for S=-1, Excitation=1 and Frequency=250 MHz. In case the corresponding project doesn't already have the results, simulate it first in WIPL-D Pro and then execute the code.

```
import wiplpy.WResults
import wiplpy.WiplInterface
ProjectPath="C:\\Microwave Pro v6.2\\3DEM\\Tutorial\\Demo223"
cu=wiplpy.WResults.InitializeCUResults(ProjectPath)
cuts={ "S":-1,
       "Excitation":1,
       "Frequency": cu.GetFrequencies()[0],
}
```

```
XaxisLabel="P"
Component="Jp"
Format="MagdB"
Element="AllElements"
resx=cu.GetXData(XaxisLabel,Element)
res=cu.GetYData(Component,Format,XaxisLabel,cuts,Element)
print("X-axis in: "+cu.GetXUnit())
print(resx)
print("Y-axis in: "+cu.GetYUnit())
print(res)
```

Results:

```
X-axis in: U
[-1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3, -0.2, -0.1, 0.0, 0.1,
0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
Y-axis in: dB
[{'Element': 'Plate23', 'Results': [-54.23238887246, -54.92770494721,
-55.65389321910, -56.41129014627, -57.198698542417, -58.01225959424877,
-58.843695613965,-59.677745710100,-60.48886928021, -61.23809786707395,
-61.87255567477,-62.331897036434,-62.564459108690,-62.54780532567,
-62.2994086786, -61.86815559121, -61.314170456828, -60.69159404174912,
-60.040990510631, -59.389217016036, -58.75241876474]},
{'Element': 'Plate24', 'Results': [-58.75241876, -59.49030038616733,
-60.215497730709, -60.921467485854, -61.60203367364, -62.252198203404,
-62.86903128281, -63.452465590692, -64.00579745249576, -64.53574884141,
-65.05204773129, -65.56658021350, -66.09218352349, -66.64103387450,
-67.2223084831, -67.83834189897, -68.47788190709, -69.10475453558,
-69.64246129981, -69.96530206842, -69.92472474541]}]
```

5.5. Axis

Class _Axis(ResultType) – this class sets independent axes. An axis is a variable along which data can be obtained. A list of available axes depends on output results (near field, far field, YZS, current).

Possible axes for different output results are given below:

Far Field - "**Phi**", "**Theta**", "**Frequency**" or "**Excitation**",

Near Field - "**X**", "**Y**", "**Z**", "**Frequency**" or "**Excitation**",

YZS - "**Frequency**", "**i**" or "**j**",

Currents - "**P**", "**S**", "**Frequency**" or "**Excitation**".

The default value for independed axes are:

Far Field - "**Phi**",

Near Field - "**X**",

YZS - "**Frequency**",

Currents - "**P**".

Parameters:

ResultType – type of the output results (near field, far field, YZS, current).

The following methods are available:

PrintAllAxes() – prints all possible axes for the selected output results.

SetAxis(XaxisLabelNew) – sets new independent axis by passing *XaxisLabelNew* parameter.

Parameters:

X*axisLabelNew* – depends on the output results and can be:

Far Field - "**Phi**", "**Theta**", "**Frequency**" or "**Excitation**",

Near Field - "**X**", "**Y**", "**Z**", "**Frequency**" or "**Excitation**",

YZS - "**Frequency**", "**i**" or "**j**",

Currents - "**P**", "**S**", "**Frequency**" or "**Excitation**".

GetAxis() – returns the selected *XaxisAxis*.

GetAllAxes() – returns the list of all posible axes for selected output results.

5.6. Cut

Class _Cut(ResultType,**fixed_values) – this class sets the fixed values for which the results are obtained. The cuts depend on the selected independent axis and the output results. A variable that is selected to be axis cannot be Cut at the same time.

If the fixed value is selected as the axis, its value will not be considered. Cut is defined as dictionary with the following data:

Far Field:

Cuts={"**Phi**": phi-angle value,
"**Theta**": theta-angle value,
"**Frequency**": frequency value,
"**Excitation**": index of excitation,
}

Near Field:

Cuts={"**X**": x-coordinate value,
"**Y**": y-coordinate value,
"**Z**": z-coordinate value,
"**Frequency**": frequency value,
"**Excitation**": index of excitation,
}

YZS:

Cuts={"**i**": first port parameter,
"**j**": second port parameter,
"**Frequency**": frequency value
}

Currents:

Cuts={"**P**": p-coordinate value
"**S**": s-coordinate value
"**Frequency**": frequency value
"**Excitation**": index of excitation
}

Parameters:

ResultType – type of the output results (near field, far field, YZS, current).

***fixed_values* – arbitrary number of arguments. This way the function will receive a dictionary of arguments that represent fixed values (cuts).

5.7. Component

Class _Component(ResultType) – this class sets the components for selected results.

Possible components for different output results are given below:

Far Field - "**Total**", "**Phi-component**", "**Theta-component**", "**RC**", "**LC**", "**MaxLin**", "**MinLin**", "**Copolar**", "**Crosspolar**".

Near Field - "**Etotal**", "**Ex**", "**Ey**", "**Ez**", "**Htotal**", "**Hx**", "**Hy**", "**Hz**", "**Ptotal**", "**Px**", "**Py**", "**Pz**", or "**SAR**".

YZS - "**Admittance**", "**Impedance**", "**Sparameter**".

Current - "**I**", "**J**", "**Jp**", "**Js**".

Parameters:

ResultType – type of the output results (near field, far field, YZS, current).

The following methods are available:

PrintAllComponentss() – prints all available components for selected output results.

SetComponent(CompNameNew) – sets the component by passing *CompNameNew*

Parameters:

CompNameNew – depends on output results an can be:

Far Field - "**Total**", "**Phi-component**", "**Theta-component**", "**RC**", "**LC**", "**MaxLin**", "**MinLin**", "**Copolar**", "**Crosspolar**".

Near Field - "**Etotal**", "**Ex**", "**Ey**", "**Ez**", "**Htotal**", "**Hx**", "**Hy**", "**Hz**", "**Ptotal**", "**Px**", "**Py**", "**Pz**", or "**SAR**".

YZS - "**Admittance**", "**Impedance**", "**Sparameter**".

Currents - "**I**", "**J**", "**Jp**", "**Js**".

GetComponent() – returns selected component.

GetAllComponents() – returns all available components for selected output results.

5.8. Format

Class _Format(ResultType,Scatterer=False) – this class sets the format in which selected parameter is calculated.

Parameters:

ResultType – type of the output results (near field, far field, YZS, current).

Scatterer – scatterer operation mode. Supported inputs are "**True**" or "**False**". **The default value is** "**False**".

Possible formats for different output results are given below:

Far Field - "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**", "**Gain**", "**GaindB**", "**Ellipticity**", "**EllipticitydB**", "**PolarAngle**","**RCS**", "**RCSdB**".

Near Field - "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**" or "**dBu**".

YZS - "**RI**", "**Re**", "**Im**", "**Mag**", "**MagdB**", "**Phase**", "**VSWR**".

Currents - "**RI**", "**Re**", "**Im**", "**Mag**", "**MagdB**", "**Phase**".

The following methods are available:

PrintAllFormats() – prints all possible formats for selected output results and parameters.

SetFormat(FormatNameNew) – sets the format by passing *FormatNameNew*.

Parameters:

FormatNameNew can be:

Far Field - "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**", "**Gain**", "**GaindB**", "**Ellipticity**", "**EllipticitydB**", "**PolarAngle**","**RCS**", "**RCSdB**".

Near Field - "**RI**", "**Re**", "**Im**", "**Mag**", "**Phase**" or "**dBu**".

YZS - "**RI**", "**Re**", "**Im**", "**Mag**", "**MagdB**", "**Phase**", "**VSWR**".

Currents - "**RI**", "**Re**", "**Im**", "**Mag**", "**MagdB**", "**Phase**".

GetFormat() – returns selected format.

GetFormats() – returns all available formats for selected output results.