Pyawr Utilities

Contents

Element xy position

Element Enable/Disable

Element Nodes Dictionary

Parameters List

Parameters Dictionary

Value

String Value
Expression List

Equations Dictionary

Add Equation

Remove Equation

Set Equation Object

Equations Expression

Equation Name

Equation Enable/Disable

Equation Value

Equation Variable Type

Equation xy Position
System Diagram Names List

System Diagram Dictionary

Rename System Diagram

Copy System Diagram

Add System Diagram

Remove System Diagram

Default System Frequencies

Default System Options

Set System Diagram Object

System Diagram Name

Add Wire

Remove Wire

Wire Segments Dictionary

Element Name List

Element Dictionary

Add Element

Remove Element

Parameter Name List

Parameter Dictionary

Modify Parameter Value

Element xy position

Element Enable/Disable

Element Nodes Dictionary
Expression List

Equations Dictionary

Add Equation

Remove Equation

Set Equation Object

Equation Expression

Equation Name

Equation Enable/Disable

Equation Value

Equation Variable Type

Equation xy Position

System Diagram Use Default Frequencies

System Diagram Frequencies

Set Grid Visibility

System Diagram Use Default Options

System Diagram Options
Output Equations List

Output Equations Dictionary

Add Output Equation Document

Remove Output Equation Document

Set Output Equation Object

Output Equation Document Name

Equations
Graph Names List

Graphs Dictionary

Add Graph

Rename Graph

Copy Graph

Remove Graph

Set Graph Object

Graph Name

Graph Type

Freeze/Unfreeze Traces

Measurement Name List

Measurement Dictionary

Add Measurement

Remove Measurement

Set Measurement Object

Measurement Name

Measurement Source Document

Enable/Disable

Modify Measurement

Measurement Axis

Number of Traces

Read Trace Data

Read SWPVAR Parameters
Axis Name List

Axis Dictionary

Set Axis Object

Axis Name

Auto Scale

Min Scale

Max Scale

Grid Step
Marker Name List

Marker Dictionary

Add Marker

Remove Marker

Set Marker Object

Marker Name

Marker Measurement

Marker Sweep Value

Marker to Max

Marker to Min

Marker Trace Index

Read Marker Value
Data File Names List

Date Files Dictionary

Add Data File

Remove Data File

Rename Data File

Copy Data File

Import Data File

Set Data File Object

Data File Name

Date File Type

Export Data File
Global Definitions Documents List

Global Definitions Documents Dictionary

Add Global Definitions Document

Remove Global Definitions Document

Set Global Definition Document Object

Elements and Equations
Optimization Variables Dictionary

Optimization Print Variables

Optimization Type List

Optimization Type

Optimization Type Properties

Optimization Maximum Iterations

Optimization Show All Iterations

Optimization Stop At Minimum Error

Optimization Stop On Simulation Error

Optimization Cancel On Stop Request

Optimization Log To File

Optimization Read Best Cost

Optimization Read Current Cost

Optimization Start/Stop

Optimization Round Variables

Parameter/Equation Optimization Enable/Disable

Parameter/Equation Optimization Constraint Enable/Disable

Parameter/Equation Optimization Lower/Upper Constraint

Parameter/Equation Optimization Constraint Step Size
Optimization Goals Dictionary

Add Optimization Goal

Remove Optimization Goal

Set Optimization Goal Object

Optimization Goal Name

Optimization Goal Measurement Name

Optimization Goal Source Document Name

Optimization Goal Type

Optimization Goal Enable/Disable

Optimization Goal X Range

Optimization Goal Y Range

Optimization Goal Weight

Optimization Goal L Factor

Optimization Goal Cost
Optimization Log File Selection

Optimization Log File Extract

Optimization Log File Parameters

Optimization Log File Goals

Optimization Log File Variables

Optimization Log File Cost Array
Yield Variables Dictionary

Print Yield Variables Dictionary

Yield Analysis Type List

Yield Analysis Type

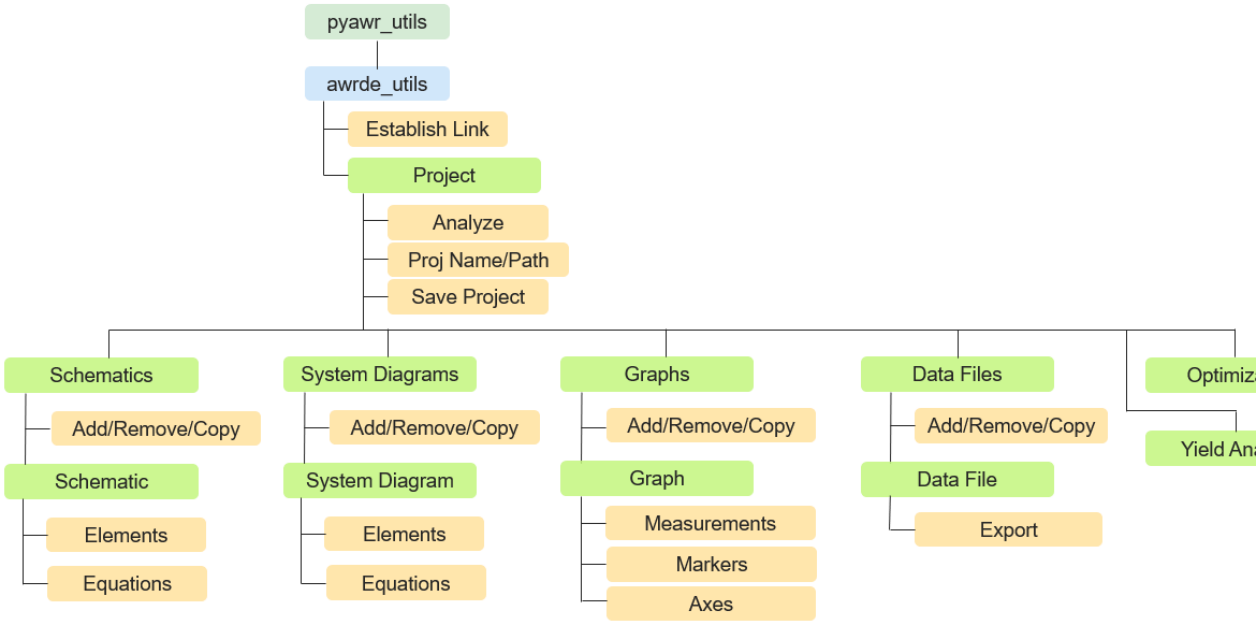
Yield Anlaysis Type Properties

Overview

Introduction

pyawr_utils is Python module that provides a higher level interface to the AWR Design Environment (AWRDE) API. Python code used for interfacing with AWRDE uses the command syntax found in the document: *AWR API Scripting Guide*. Although the API is written for SAX Basic, which is compatible with Visual Basic for Applications (VBA), *AWR Scripting in Python: Using the AWRDE API Scripting Guide* gives instructions on how to interpret the VBA commands into Python syntax. This exhaustive set of commands can make scripting AWRDE in Python a laborious task. **pyawr_utils** encapsulates the lower level API commands into a higher level set of commands thus creating a simpler API interface for some of the more common scripting tasks. The intent is to provide a simpler interface to the AWRDE API so that the user can speed up the time it takes to develop a script.

pyawr_utils is organized as shown:



Python classes provide entry points for the major AWRDE groupings: *Schematics, System Diagrams* , *Graphs* and *Data Files.* These major class groups fall under the overall class *Project,* which illustrates the *Project* class is the single entry point for all of the **pyawr_utils** operations with one exception: establishing a link between Python and AWRDE. The groupings all follow a common theme where classes that are plural (*Schematics, System Diagrams, Graphs, and Data Files*) operate on the collection of items. For example under *Graphs* operations such as adding graphs, removing graphs, copying graphs exist. Classes that are singular (*Schematic, System Diagram, and Graph*) have operations particular to an individual document. For example under *Graph*, operations on a single graph exist.

pyawr_utils Limitations

Only a fraction of the API commands are available through **pyawr_utils**. The intent of **pyawr_utils** is to provide a higher level set of commands for some of the more common activities. For commands not found in pyawr_utils then directly accessing the AWRDE API will be required. This link provides details on direct API command access.

Installation

Requirements

Python 3.9 or later and **pyawr** must be installed. See the instruction in *AWR Scripting in Python: Getting Started and Installation* or *AWR Scripting in Python: Using Anaconda or Microconda in Your Python Workflow*

An additional module is **numpy**. From a Windows command window type

```
pip install numpy
```

pyawr_utils Installation

From a Windows command window type

```
pip install pyawr-utils
```

Project

Establish Link Between Python and AWRDE

Before using **pyawr_utils**, AWRDE must be running with the desired project opened. In the Python code the file **awde_utils** must be imported from the **pyawr_utils** package as shown:

```
from pyawr_utils import awrde_utils
```

There are three different ways of creating a link between Python and AWRDE dependent on how AWRDE is configured. An object variable **awrde** is created in each case. This object variable will be used in subsequent calls to **pyawr_utils.**

1 Single AWRDE version installed

When only one version of AWRDE is installed on the computer, the simplest form is to use the establish_link call without pass parameters as shown:

```
from pyawr_utils import awrde_utils
awrde = awrde_utils.establish_link()
```

2 Multiple AWRDE versions installed

If multiple versions of AWRDE are installed on the computer, then the version_str parameter will need to be passed as shown here

```
from pyawr_utils import awrde_utils
awrde = awrde_utils.establish_link(version_str='16.0')
```

Even for minor revisions, still use the format 'XX.0' for the version string. For instance, AWRDE version 16.02 would still use '16.0'.

3 Class ID

Class ID (CLSID) is a unique identification number associated with each opened session of AWRDE. So if multiple sessions of AWRDE are opened simultaneously, CLSID allows Python to link to the desired AWRDE session. The CLSID number is obtained using VBA scripting in the AWRDE. For information on using VBA within AWRDE see this link. Shown here is the VBA code required to print out the CLSID number:

```
Sub Main
        Debug.Print MWOffice.InstanceCLSID
End Sub
```

In Python, use this syntax:

```
from pyawr_utils import awrde_utils
awrde = awrde_utils.establish_link(clsid_str='5BF3163E-6734-4FB4-891E-FD9E3D4A2CFA')
```

Project Class

All methods and properties besides the *establish_link* method are under class *Project.* Accessing class *Project* uses the syntax shown

```
Project = awrde_utils.Project(awrde, bypass_initalization=False)
```

Parameters

awrde: object returned from the establish_link function

bypass_initalization: boolen, optional. Default=False.

False: initializes all schematics, system diagrams, graphs, global definitions documents and output equation documents.

Initialization involves creating dictionaries of elements, element parameters, equations

True: byasses initialization. For larger projects this option can speed up code execution

Returns

Project: object for the class Project

Initialize On Demand

If the Project Class parameter bypass_initalization is set to True, the initialize_on_demand command can selectively initialize various components

```
Project.initialize_on_demand(which_initialization='all')
```

Parameters

which_initialization: string, optional. Default = 'all'

valid commands: 'all', 'schematics', 'system diagrams', 'graphs', 'data files', 'global definitions', 'optimization', 'output equations'

'all' : initialize all the following items

'schematics': initializes all circuit schematics. Builds up dictionaries of elements, element parameters, and equations

'system diagrams': initializes all system diagram. Builds up dictionaries of elements, element parameters, and equations

'global definitions': initializes all global definitions documents. Builds up dictionaries of elements, element parameters, and equations

'data files': initializes all data files. Builds dictionary of data file names

'graphs': initialize all graphs. Builds dictionaries of graphs, and measurements

'optimization': initializes optimization related items. Build dictionary of optimization variables, goals and optimizer types

'output equations': initializes output equation documents

Project Name

Returns project name as a string for the currently opened project.

```
project_name = Project.project_name
```

Project Path

Returns the file location as a string for the currently opened project.

```
project_path = Project.project_path
```

Save Project

Save the project command.

```
Project.save_project()
```

Save Project As

Project Save As command.

```
Project.save_project_as(Dir_n_File='Full Path and File Name')
```

Parameters

Dir_n_File: string. The full path and file name of the project

Save Project As Verion

Save project as a specific version

```
Project.save_project_as_version(Dir_n_File='Full Path and File Name', version_num=16)
```

Parameters

Dir_n_File: string. The full path and file name of the project

version_num: integer. Major release version number. For instance, AWRDE 16.03 would use version_num=16

Open Project

Open a project.

```
Project.open_project(Dir_n_File='Full Path and File Name')
```

Parameters

Dir_n_File: string. The full path and file name of the project

Close Project

Close the current project. Does not save, so make sure Save Project or Save Project As command has been initiated prior to issuing the Close Project commant

```
Project.close_project()
```

Simulate Analyze

Equivalent to the AWRDE command Simulate > Analyze

```
measurement_done = Project.simulate_analyze(ping_interval=1, max_time=120)
```

Parameters

ping_interval: float, optional. Wait time in seconds between checking for simulation status. Default is 1 second.

max_time: float, optional. Maximum simulation time in seconds before exiting simulation. Default is 120 seconds

Returns

measurement_done: boolean. Returns True if simulation completed successfully

The simulate_analyze method starts a circuit, EM or VSS frequency simulation. This method waits until either the simulation has completed or the max_time value has been achieved. During the simulation the simulation status is checked every ping_interval seconds. Once the simulation is either complete or max_time has been achieved, the boolean measurement_done is returned.

Run/Stop System Simulator

Equivalent to the AWRDE command Simulate > Run/Stop System Simulators. This starts a VSS time domain simulation

```
measurement_done = Project.simulate_run_system_simulator(ping_interval=1, max_time=120)
```

Parameters

ping_interval: float, optional. Wait time in seconds between checking for simulation status. Default is 1 second.

max_time: float, optional. Maximum simulation time in seconds before exiting simulation. Default is 120 seconds

Returns

measurement_done: boolean. Returns True if simulation completed successfully

The simulate_analyze method starts a VSS time domain simulation. This method waits until either the simulation has completed or the max_time value has been achieved. During the simulation the simulation status is checked every ping_interval seconds. Once the simulation is either complete or max_time has been achieved, the boolean measurement_done is returned. If the system diagram option *Use Simulation Stop Time* is not checked, then the simulation will keep running after exiting this method.

To stop a running time domain simulation, use the stop_system_simulator command:

```
Project.stop_system_simulator()
```

Default Project Frequencies

Reading and setting project level frequencies found in Options > Project Options > Frequencies tab

Reading default project frequencies:

```
freq_array = Project.project_frequencies
```

Returns

freq_array: numpy ndarray An array of default project frequencies in Hz

Setting default project frequencies

```
freq_array = np.linspace(0.5, 2, 21)                                        #numpy command to create a
vector
Project.set_project_frequencies(project_freq_ay=freq_array, units_str='GHz')
```

Parameters

project_freq_ay: numpy ndarray An array of default project frequencies

units_str: string, optional. Units for the values in freq_array. Valid strings: 'Hz', 'kHz', 'MHz', 'GHz' Default is 'GHz'

Environment Options

Reading and setting options in Options > Environment Options

Read Environment Options

```
env_options_dict = Project.environment_options_dict
for option_name, option_value in env_options_dict.items():
    print(option_name,' : ', option_value)
#end for
```

Returns

env_options_dict: dictionary. Dictionary keys are the option names and dictionary values are the option values

The above code returns:

```
ShowTopLevelOnly  :  True
OptionNodesInProject  :  False
CollapseProjectTreeOnOpen  :  False
EvironmentOptionsInProject  :  False
SaveRevisions  :  False
AutoSaveProject  :  False
AutoSaveInterval  :  15
RevisionCount  :  1
SaveBeforeSimulating  :  False
ShowSplashScreen  :  False
RunScriptingEnv  :  False
DrawDisabledGoals  :  True
ShowDisabledMeasurementsInLegends  :  False
EqnEditAutoComplete  :  True
EnableGPUFor2DLayout  :  False
EnableGPUFor3DLayout  :  False
DiagramTextColor  :  0
DiagramWireColor  :  255
DiagramNodeColor  :  65280
DiagramDisabledColor  :  11842740
SchematicBackColor  :  16777215
```

```
SchematicGridColor  :  0
ElementNormalColor  :  16711680
ElementLayoutColor  :  16711860
SchematicSourceColor  :  180
SchematicMeterColor  :  25600
SystemBackColor  :  16777215
SystemGridColor  :  0
SystemElementNormalColor  :  120
SystemSourceColor  :  25855
LayoutBackColor  :  16777215
LayoutGridColor  :  0
PlainTextColor  :  8388608
SchemShapeColor  :  0
TuneParamColor  :  16711680
TuneOptParamColor  :  16711680
OptParamColor  :  0
ReadOnlyParamColor  :  65280
GraphBackColor  :  16777215
NetRoutedColor  :  16711860
OutputEqnColor  :  5201441
GlobalDefBackColor  :  16777215
GlobalDefGridColor  :  0
OutputEqnBackColor  :  16777215
OutputEqnGridColor  :  0
HideSymbolElementNodes  :  False
SymbolLinesBack  :  False
BoldElementNamesBold  :  False
BoldParameterNames  :  False
SymbolLineThickness  :  0
DependentParametersInBaseUnits  :  False
SchemElementFontSize  :  10
SchemEquationFontSize  :  10
SchemElementFontName  :  Arial
SchemEquationFontName  :  Arial
LayoutSnapTogetherMode  :  0
LayoutNoRotateOnSnap  :  False
LayoutScalePrintouts  :  False
LayoutPrintScaleFactor  :  1.0
LayoutPrintMaxLayervisible  :  True
LayoutDefaultFaceAlign  :  0
LayoutEnablePinAreaDef  :  True
LayoutProjectTopLevelCellsOnly  :  True
ShowDerivedLPF  :  True
LayoutSnapToAdjacentDef  :  True
SysDependentParametersInBaseUnits  :  True
EnableElementAlignmentGuides  :  True
MouseZoomModifierKey  :  2
MouseVertScrollModifierKey  :  0
MouseHorizScrollModifierKey  :  1
MouseWheelTiltHorizScroll  :  False
MouseMiddleButtonPan  :  True
MouseRotate3DModifierKey  :  3
MouseTilt3DModifierKey  :  4
AllowParameterFrameRotation  :  True
```

Setting environment options. The following shows how to modify a single environment option and then write the environment options to AWRDE

```
env_options_dict = Project.environment_options_dict                                 #Read environment options
env_options_dict['ShowTopLevelOnly'] = True                                         #Modify an option
Project.set_environment_options_dict(environment_options_dict=env_options_dict)     #Write environment options
```

Parameters

environment_options_dict: dictionary. Dictionary keys are the option names and dictionary values are the option values

Default Project Options

Reading and setting project options in Options > Project Options. Exception is frequencies under the Frequencies tab, for that refer to this link.

Reading project options:

```
proj_options_dict = Project.project_options_dict
for option_name, option_value in proj_options_dict.items():
    print(option_name,' : ', option_value)
#end for
```

Returns

proj_options_dict: dictionary. Dictionary keys are the option names and dictionary values are the option values

The above code returns:

```
HideParameters  :  False
HideParameterUnits  :  False
HideParameterIfEmpty  :  False
HideParameterVarName  :  False
LeftJustifyParameters  :  False
BoldParameterFont  :  False
HideElementNames  :  False
BoldElementFont  :  False
LeftJustifyElementName  :  True
HideSecondaryParameters  :  True
SymbolLineThickness  :  0
BlackAndWhite  :  False
DependentParametersInBaseUnits  :  False
SignificantParameterDigits  :  4
ParameterZeroThreshold  :  1e-30
SchematicCleanUpWires  :  True
ParametersMultiColumn  :  False
SignificantAnnotationDigits  :  3
NumberInterpolationPoints  :  200
InterpolationWindowSize  :  10
InterpolationSplineOrder  :  3
InterpolationMethod  :  2
InterpolationCoordinates  :  0
EnableInterpolation  :  False
ConsiderPassiveForNoise  :  False
LayoutGridSpacing  :  25400
LayoutDatabaseUnits  :  2540
LayoutAngleSnapDiv  :  45
LayoutSnapTogetherMode  :  2
LayoutNoRotateOnSnap  :  False
LayoutPointsPerCircle  :  36
LayoutConnectionLineType  :  2
LayDefaultClosestFace  :  False
LayFixedCellOrigin  :  True
LayFixedSubCktOrigin  :  True
LayCellOriginOnGrid  :  True
LayLockSubCktHierUpdate  :  False
LayExcludePcell  :  False
LayAutoSnapParamSubCircuits  :  True
InstanceExportOptions  :  0
ExportMergeShapes  :  False
ExportSubcircuitsAsInstances  :  True
LayoutCellExportOptions  :  0
LayCellLimitNameLen  :  False
LayExportCellFaces  :  False
LayCellRemoveIllegalChars  :  False
LayExportWriteParamMap  :  False
LayExportPromptForLPF  :  False
LayExportUseParentLPFForEM  :  False
LimitVerticesPerPolygon  :  8123
ExportUnionShapes  :  False
LayCellsExportAsInstances  :
LayCellsNotExportAsInstances  :
PathWidth  :  1270000
```

```
PathMiterOffset  :  635000
PathEndType  :  0
PathMiterType  :  0
DrawPathAsPolygon  :  1
LayPathDefProcessLineType  :  0
LayPathDefaultUseProcessLayers  :  False
PathCurvesUseFixedRadius  :  False
LayInetMinViaSize  :  True
LayInetViaSizeOpts  :  0
LayInetDefRouteBendStyle  :  0
LayInetDefRouteBendAmount  :  1.0
LayRouteViaMode  :  0
LayRouteBendAmountRel  :  True
LayDefRouteConnectionModel  :  0
LayRouteCurveFixedRadius  :  False
LayRouteDefaultSemiAutoVia  :  False
RouteMinSpacing  :  10000
RoutePathWidth  :  10000
RouteLevel  :  2
LayoutRouteLineType  :  0
RulerFontFace  :  Arial
RulerFontHeight  :  25000
RulerTickHeight  :  15000
RulerDisplayPrecision  :  1
RulerShowUnits  :  False
RulerSpacing  :  30000
RulerTickLocation  :  1
RulerDefaultGap  :  5000
LayoutRulerShowLabels  :  True
LayoutRulerShowMinorTicks  :  True
DimensionLineArrowSize  :  20000
DimensionLineTextFont  :  Arial
DimensionLineFontHeight  :  25000
DimensionLineSidesOffset  :  5000
DimensionLineDisplayPrecision  :  1
DimensionLineShowUnits  :  False
DimensionLineShowTolerance  :  False
DimensionLineToleranceLength  :  0
DimensionLineTextLocation  :  1
DimensionLineArrowLocation  :  0
LayoutTextFont  :  Arial
LayoutTextHeight  :  100000
LayoutTextBold  :  False
LayoutTextItalic  :  False
LayoutTextDrawAsPolygons  :  True
CellStretcherArrowHeight  :  30000
CellStretcherDefaultMultiplier  :  1.0
CellStretcherDefaultOffset  :  0.0
CellStretcherUpperLimit  :  500000000
CellStretcherLowerLimit  :  -500000000
CellStretcherIsBounded  :  True
CellStretcherDefaultParameter  :  L
SysDependentParametersInBaseUnits  :  True
DefBooleanVersion  :  0
DefBooleanCornerStyle  :  0
YldOp_NumbCompBins  :  18
YldOp_NumbMeasBins  :  21
YldOp_SaveSamples  :  1
YldOp_SaveMeasOpt  :  2
YldOp_LimitYieldTraces  :  0
YldOp_MinYieldTraces  :  100
YldOp_MaxYieldTraces  :  1000
AWR_JobSchedulerJobOSPriority  :  0
AWR_JobSchedulerJobRemotePreference  :  0
AWR_JobSchedulerJobRemoteServerSelection  :  Default
AWR_JobSchedulerJobPerfPreference  :  2
AWR_JobSchedulerJobMemCapPreference  :  2
AWR_JobSchedulerMaxRemoteComputers  :  0
AWR_JobSchedulerMaxJobsPerComputer  :  1
AWR_OS_PercentImprove4AsyncUpdate  :  10
AWR_AsyncMWOServerLogging  :  0
```

```
AWR_SaveProjAfterMWOServerSucces  :  1
RT_TestVoiding  :  0
RT_ExclusiveOrforLayComp  :  0
```

Setting project options. The following shows how to modify a single project option and then write the project options to AWRDE

```
proj_options_dict = Project.project_options_dict                          #Read project options
proj_options_dict['LayoutDatabaseUnits'] = 2550                           #Modify an option
Project.set_project_options_dict(project_options_dict=proj_options_dict)  #Write project options to AWRDE
```

Parameters

project_options_dict: dictionary. Dictionary keys are the option names and dictionary values are the option values

Project Show Files/Directories

Returns a dictionary of project files and directories. The files and directories can be obtained from Main Menu > Help > Show Files/Directories

```
show_files_dirs_dict = Project.show_files_directories_dict       #Read the dictionary
for name, path in show_files_dirs_dict.items():                  #Print the names and directory paths
    print(name,'    ',path)
#end for
```

Returns

show_files_dirs_dict: dictionary. Dictionary keys are the file or directory names, dictionary values are the paths

Example output of the above code:

```
Temporary      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\temp\awrF5E1.tmp
CommonIni      C:\ProgramData\AWR\Design Environment\17.0\mwoffice.ini
UserIni      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\user.ini
Customizations      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\customizations.xml
DataCache      C:\ProgramData\AWR\Design Environment\17.0\dedata.cache
License      C:\user_name_awr\MWO_Files\license_files\awrd_17.lic
Executable      C:\Program Files (x86)\AWR\AWRDE\17\MWOffice.exe
EMSightCache      C:\ProgramData\AWR\Design Environment\17.0\emsight.cache
MWOfficeExe      C:\Program Files (x86)\AWR\AWRDE\17\mwoffice.exe
Project      C:
\user_name_awr\MWO_Files\Testing_n_KB\v17_Features\PointerHybridOptimizer\Sandbox\Lumped_Distributed_LPF.emp
GlobalScript      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\global.mws
AppDir      C:\Program Files (x86)\AWR\AWRDE\17
AppData      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0
AppDataUser      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0
AppDataCommon      C:\ProgramData\AWR\Design Environment\17.0
TempFile      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\temp
AppDataLog      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\logs
AppDataLibraryCache      C:\ProgramData\AWR\Design Environment\17.0\libcache
Examples      C:\Program Files (x86)\AWR\AWRDE\17\examples
Libraries      C:\Program Files (x86)\AWR\AWRDE\17\library
EmModels      C:\Program Files (x86)\AWR\AWRDE\17\em_models
Models      C:\Program Files (x86)\AWR\AWRDE\17\models
Cells      C:\Program Files (x86)\AWR\AWRDE\17\cells
Symbols      C:\Program Files (x86)\AWR\AWRDE\17\symbols
Signals      C:\Program Files (x86)\AWR\AWRDE\17\signals
Textures      C:\Program Files (x86)\AWR\AWRDE\17\textures
Data      C:\Program Files (x86)\AWR\AWRDE\17\data
HSpice      C:\Program Files (x86)\AWR\AWRDE\17\analog
TestResults      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\testresults
EmModelsUser      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\em_models
Logs      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\logs
LibraryCache      C:\ProgramData\AWR\Design Environment\17.0\libcache
Analog      C:\Program Files (x86)\AWR\AWRDE\17\analog
Projects      C:\Users\user_name\Documents\AWR Projects
Scripts      C:\Program Files (x86)\AWR\AWRDE\17\scripts
ScriptsUser      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\scripts
CurrentProject      C:\user_name_awr\MWO_Files\Testing_n_KB\v17_Features\PointerHybridOptimizer\Sandbox\
Documents      C:\Users\user_name\Documents
DesignKits      C:\Program Files (x86)\AWR\AWRDE\17
Measurements      C:\Program Files (x86)\AWR\AWRDE\17\measurements
XmlUser      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\xml
WizardsUser      C:\Users\user_name\AppData\Local\AWR\Design Environment\17.0\wizards
Wizards      C:\Program Files (x86)\AWR\AWRDE\17\wizards
DataSet      \
VersionControl      C:\Users\user_name\AppData\Local\AWR\Design Environment\All Versions\vcs
OpenAccess      \
```

Delete All Data Sets

Removes all data sets from the project

```
Project.delete_all_datasets
```

Status Messages List

Returns list of messages that are present in the Status Window

```
StatusMsg_list = self.Project.status_messages_list
```

Returns

StatusMsg_list: list

Each row in the list is a status message

In each row:

Index 0: "Information" | "Warning" | "Error"

Index 1: Status Group Name

Index 2: Status text

Clear Status Messages

Delete all messages in the Status Window

```
Project.clear_status_messages
```

Schematics

Circuit Schematics Names List

Returns a Python list of all the circuit schematic names in the project

```
circuit_schematics_names_list = Project.circuit_schematics_name_list
```

Returns

circuit_schematics_names_list: list.

Circuit Schematics Dictionary

The circuit schematics dictionary method creates a python dictionary of schematic objects.

```
circuit_schematics_dict = Project.circuit_schematics_dict
```

Returns

circuit_schematics_dict: dictionary. Dictionary keys are the circuit schematic names and the dictionary values are schematic objects.

Add Circuit Schematic

```
circuit_schematic_name = 'test_add_circuit_schematic'
new_circuit_schematics_dict = Project.add_circuit_schematic(circuit_schematic_name)
```

Parameter

circuit_schematic_name: string. Name of the desired circuit schematic to be added.

Returns

new_circuit_schematic_dict: Returns a new circuit schematics dictionary with the new circuit schematic dictionary key and schematic object added.

Remove Circuit Schematic

```
circuit_schematic_name = 'test_remove_circuit_schematic'
removed_circuit_schematics_dict = Project.remove_circuit_schematic(circuit_schematic_name)
```

Parameter

circuit_schematic_name: string. Name of the desired circuit schematic to be removed.

Returns

new_circuit_schematic_dict: Returns a new circuit schematics dictionary with the removed circuit schematic dictionary key and schematic object deleted.

Copy Circuit Schematic

```
circuit_schematic_name = 'existing_schematic'
new_circuit_schematics_name = 'copy_of_existing_schematic'
new_circuit_schematic_dict_copy = Project.copy_circuit_schematic(circuit_schematic_name,
new_circuit_schematic_name)
```

Parameter

circuit_schematic_name: string. Name of the desired circuit schematic to be copied.

new_circuit_schematic_name: string. Name of the new schematic copied from an existing schematic.

Returns

new_circuit_schematic_dict: Returns a new circuit schematics dictionary with the new circuit schematic dictionary key and schematic object added.

Rename Circuit Schematic

```
existing_circuit_schematic_name = 'existing_schematic'
circuit_schematic_renamed = 'test_rename_circuit_schematic'
circuit_schematic_dicst_renamed = Project.rename_circuit_schematic(existing_circuit_schematic_name,
circuit_schematic_renamed)
```

Parameter

existing_circuit_schematic_name: string. Name of the desired circuit schematic to be copied.

new_circuit_schematic_name: string. Name of the new schematic copied from an existing schematic.

Returns

new_circuit_schematic_dict: Returns a new circuit schematics dictionary with the new circuit schematic dictionary key and schematic object added.

Schematic

Circuit Schematic Name

Returns the circuit schematic name

```
circuit_schematic_name = 'test_name_circuit_schematic'
circuit_schematic_name_string = new_circuit_schematics_dict[circuit_schematic_name].circuit_schematic_name
```

Returns

circuit_schematic_name_string: string. Name of the circuit schematic

Use Project Frequencies

Checks to see if the circuit schematic is using the project frequencies and toggles the setting on and off.

```
circuit_schematic_name = 'test_toggle_circuit_schematic_frequencies'
circuit_schematic_bool = circuit_schematics_dict[circuit_schematic_name].use_project_frequencies

circuit_schematics_dict[circuit_schematic_name].use_project_frequencies = False
```

Parameters

Set the use_project_frequencies to True or False to toggle use project frequencies in the schematic options

Returns

circuit_schematic_bool: bool. If called, returns True or False depending on if the schematic has the use project frequencies toggled on or off in the schematic options.

Get Frequency Range

Gets the frequency values of the circuit chematic.

```
circuit_schematic_name = 'test_circuit_schematic_frequency_range_set'
circuit_schematic_frequency_range = new_circuit_schematics_dict[circuit_schematic_name].frequency_range
```

Parameters

None

Returns

circuit_schematic_frequency_range: np.ndarray. Returns a numpy array containing the frequencies in base units.

Set Frequency Range

Sets the frequency range of the schematic. Note this also toggles off use project frequencies in the schematic options

```
circuit_schematic_name = 'test_circuit_schematic_frequency_range_set'
new_circuit_schematic_frequency_range = np.array([1, 2, 3, 4, 5])
frequency_unit = 'GHz'
new_circuit_schematics_dict[circuit_schematic_name].set_frequency_range(new_circuit_schematic_frequency_range,
frequency_unit)
```

Parameters

new_circuit_schematic_frequency_range: np.ndarray. Numpy array of frequency values unscaled

frequency_unit: string. Units desired for frequency values contained in frequency range numpy array.

Returns

None

Set Grid Visibility

Turns on or off the schematic grid visibility

```
circuit_schematic.set_grid_visible(visibility_boolean)
```

Parmeters

visibility_boolean: Boolean

Schematic Add Wire

Add a wire in the selected schematic

```
circuit_schematic.add_wire(start_xy=(100,0), end_xy=(200,0))
```

Parameters

start_xy: tuple. x,y values for wire start point

end_xy: tuple. x,y values for wire end point

Schematic Remove Wire

Remove a wire from the selected schematic

```
circuit_schematic.remove_wire(start_xy=(100,0), end_xy=(200,0))
```

Parameters

start_xy: tuple. x,y values for wire start point

end_xy: tuple. x,y values for wire end point

Schematic Wire Segments Dictionary

Returns a dictionary of wire segments x,y positions for all the wires in the selected schematic

```
wire_segments_dict = circuit_schematic.wire_segments_dict
```

Returns

wire_sements_dict: dictionary

keys are the wire segment indices

values are a tuple of wire segment x,y positions: (start x, start y, end x, end y)

Elements

Circuit Schematic Element Names List

Returns a list of all the element names in the selected circuit schematic

```
elements_list = circuit_schematic.element_names_list
```

Returns

elements_list: list. Each item in the list is an element name and ID. Format is: 'Element Name.ID'

Circuit Schematic Elements Dictionary

The elements dictionary method creates a python dictionary of element objects.

```
elements_dict = circuit_schematics_dict['Circuit Schematic Name'].elements_dict
```

Parameters

None

Returns

elements_dict: dictionary. Dictionary keys are the element names and the dictionary values are element objects.

Circuit Schematic Add Element

Add an element to the selected schematic

```
circuit_schematic.add_element(element_name='Name', x_pos, y_pos, rotation=0, flipped=False)
```

Parameters

element_name: string

x_pos, y_pos: float. The x and y location for the equation

rotation: float Angle in degrees. Default=0

flipped: boolean. Default=False

Circuit Schematic Remove Element

Remove an element from the currently selected schematic

```
circuit_schematic.remove_element(self, element_name_n_id='Name.ID')
```

Parameters

element_name_n_id: string. Must be in form 'Name.ID'

Circuit Schematic Element xy Position

The x and y coordinates associated with an element can be read and set as shown:

```
elements_dict = circuit_schematics_dict['Circuit Schematic Name'].elements_dict   #Get elements dictionary
element = elements_dict['Name.ID']                                                #Set element object variable
xy_position = element.xy_position                                                 #Read element's x,y position
element.set_xy_position(xy_position_tuple=(-900,-2200))                           #Set element's x,y position
```

xy_position is read back as a tuple: (x position, y position). x_positon and y_position are type float. When setting the xy position, the pass parameter is also a tuple.

Schematic Element Enable/Disable

Enables/Disables selected element. Read the enabled state of the selected element

```
elements_dict = circuit_schematics_dict['Circuit Schematic Name'].elements_dict   #Get elements dictionary
element = elements_dict['Name.ID']                                                #Set element object variable
element.element_enabled = True                                                    #Enable element
element.element_enabled = False                                                   #Enable disable
enabled_state = element.element_enabled                                           #Read enabled state
```

Schematic Element Nodes Dictionary

Returns dictionary of x,y positions of the element nodes

```
elements_dict = circuit_schematics_dict['Circuit Schematic Name'].elements_dict   #Get elements dictionary
element = elements_dict['Name.ID']                                                #Set element object variable
nodes_dict = element.element_nodes_dict                                           #Get dictionary of node xy
positions
```

Returns

nodes_dict: dictionary

keys are the node indices

values are a tuple of the node x,y position values

Parameters

Parameter Names List

Returns list of names of the parameters for the selected element

```
elements_dict = circuit_schematics_dict['Circuit Schematic Name'].elements_dict   #Get elements dictionary
ele = elements_dict['Element Name']
parameter_names_list = ele.parameter_names_list
```

Parameters Dictionary

The parameters dictionary method creates a python dictionary of parameter objects.

```
elements_dict = circuit_schematics_dict['Circuit Schematic Name'].elements_dict['Element Name'].parameters_dict
```

Parameters

None

Returns

parameters_dict: dictionary. Dictionary keys are the parameter names and the dictionary values are parameter objects.

Parameter

Value

The parameters dictionary method creates a python dictionary of parameter objects.

```
parameter_value = circuit_schematics_dict['Circuit Schematic Name'].elements_dict['Element Name'].
parameters_dict['Parameter Name'].value

circuit_schematics_dict['Circuit Schematic Name'].elements_dict['Element Name'].parameters_dict['Parameter
Name'].value = 1e-12
```

Parameters

parameter_value: float. Setting parameter value as base units.

Returns

parameter_value: float. Calling value method returns the value of the parameter in base units.

Value String

The parameters dictionary method creates a python dictionary of parameter objects.

```
parameter_value = circuit_schematics_dict['Circuit Schematic Name'].elements_dict['Element Name'].
parameters_dict['Parameter Name'].value_str

circuit_schematics_dict['Circuit Schematic Name'].elements_dict['Element Name'].parameters_dict['Parameter
Name'].value_str = 1
```

Parameters

parameter_value: float. Setting parameter value as a string with no unit information. Value will be scaled according to project global units set in lpf of schematic

Returns

parameter_value: string. Calling value method returns the value of the parameter as a string with no unit information. The project global units are assumed.

Schematic Expressions List

Returns a list of equation expressions

```
circuit_schematic = circuit_schematics_dict['Circuit Schematic Name']
expression_list = circuit_schematic.expression_list
```

Returns

expression_list: list. Each list item is a string representing each equation expression in the system diagram

Schematic Equation Dictionary

System diagram equation objects are accessed using a dictionary using the equations_dict method as shown:

```
equations_dict = circuit_schematic.equations_dict #Read equations dictionary
for eqn_idx, equation in equations_dict.items():  #Dictionary keys are indices,
    print(eqn_idx, ' : ', equation.expression)    #Dictionary values are equation objects
#end for

equation = equations_dict[1]                      #assign equation object
```

Returns

equations_dict: dictionary. Dictionary keys are indices (0 based indexing). Dictionary values are equation objects

Schematic Add Equation

Add equation to the system diagram. The following adds the equation A = 37.85 to the currently selected system diagram:

```
circuit_schematic.add_equation(variable_name='A', variable_type='Variable definition', variable_value=37.85,
x_pos=-1000, y_pos=-2200)
```

Parameters

variable_name: string

variable_type: string. Valid values are 'Variable definition', 'Parameter definition', 'Display value'

Variable definition adds and equation in the form: variable_name = variable_value

Parameter definition adds and equation in the form: variable_name << variable_value. This is a pass parameter in a subcircuit

Display value adds and equation in the form: variable_name: This displays the value of an equation

variable_value: string, int or float

x_pos, y_pos: float. The x and y location for the equation

Schematic Remove Equation

Removes an equation from the selected system diagram

```
equation_removed = circuit_schematic.remove_equation(expression='A = 37.85')
```

Parameters

expression: string. Equation to be removed is identified by its full expression

Returns

equation_removed: boolean. True if equation successfully removed

Schematic Set Equation Object

In order to perform operations on an individual equation, first an equation object varible must be set as follows:

```
equations_dict = circuit_schematic.equations_dict  #Read equations dictionary
equation = equations_dict[3]                       #Set equation object
```

First read in the equations dictionary. Next set the equation variable using the dictionary. The keys of the dictionary are indecies corresponding to each eqution in the system diagram.

Schematic Equation Expression

The full equation string is termed the expression. To read the selected equation's expression use the following:

```
expr = equation.expression
```

Returns

expression: string. Full equation string, for instance 'x = 10'

Schematic Equation Name

Equation name can be read and set as shown:

```
equation_name = equation.equation_name    #Read equation name
equation.equation_name = 'x1'             #Set equation name
```

Schematic Equation Enable/Disable

Enable/Disable the selected equation. Read the enabled state of the selected equation

```
equation.equation_enabled = True                  #Enable the equation
equation.equation_enabled = False                 #Disable the equation
enabled_state = equation.equation_enabled         #Read the equation enabled state
```

Schematic Equation Value

Equation value can be read and set as shown:

```
equation_value = equation.equation_value   #Read equation value
equation.equation_value = 42.52            #Set equation value (string, int, float are valid types)
```

Schematic Equation Variable Type

The equation variable type can be Variable definition ('='), Parameter definition ('<<') or Display Value (':'). These can be read and set as shown:

```
variable_type = equation.variable_type            #Read variable type
equation.variable_type = 'Parameter definition'   #Set variable type
```

Valid variable types:

Variable definition: This creates an equation in the form 'x = 10'.

Parameter definition: This creates an equation in the for 'x << 10'. This is used for pass parameters in a subcircuit.

Display value: This creates an equation in the form 'x:'. This is used for displaying the value of an equaiton

Schematic Equation xy Postition

The x and y coordinates associated with an equation can be read and set as shown:

```
xy_position = equation.xy_position                          #Read equation's x,y position
equation.set_xy_position(xy_position_tuple=(-900,-2200))   #Set equation's x,y position
```

xy_position is read back as a tuple: (x position, y position). x_positon and y_position are type float. When setting the xy position, the pass parameter is also a tuple.

System Diagrams

System Diagrams Names List

List of names of all the system diagrams in the project.

```
system_diagram_name_list = Project.system_diagram_name_list
```

Returns

system_diagram_name_list: list. Each item in the list is a name of a system diagram

System Diagrams Dictionary

System diagrams are stored in a dictionary that is accessed using the system_diagrams_dict method:

```
sys_diagrams_dict = Project.system_diagrams_dict
for sys_diagram_name in sys_diagrams_dict.keys():   #system diagram names are the dictionary keys
    print(sys_diagram_name)
#end for
sys_diagram = sys_diagrams_dict['AMP sys']          #system diagram objects are the dictionary values
```

Returns

sys_diagrams_dict: dictionary. Dictionary keys are the names of the system diagrams. Dictionary values are system diagram objects

Rename System Diagram

Change the name of an existing system diagram.

```
Project.rename_system_diagram(system_diagram_name='System Diagram Name', new_system_diagram_name='System
Diagram Name')
```

Parameters

system_diagram_name: string. Name of the existing system diagram

new_system_diagram_name: string. Name of the new system diagram

Copy System Diagram

Copies existing system diagram and assigns name to the new system diagram.

```
Project.copy_system_diagram(system_diagram_name='System Diagram Name', new_system_diagram_name='System Diagram
Name')
```

Parameters

system_diagram_name: string. Name of the existing system diagram

new_system_diagram_name: string. Name of the new system diagram

Add System Diagram

Add a system diagram to the project

```
System_Diagram_Name = Project.add_system_diagram(system_diagram_name='System Diagram Name')
```

Parameters

system_diagram_name: string. Name of the existing system diagram

Returns

system_diagram_name: string. Name of the system diagram. May get changed if the system diagram already exists

Remove System Diagram

Remove a system diagram from the project.

```
Project.remove_system_diagram(system_diagram_name='System Diagram Name')
```

Parameters

system_diagram_name: string. Name of the existing system diagram

Default System Frequencies

Reading and setting system frequencies found in Options > Default System Options > RF Frequencies tab

Reading default system frequencies:

```
freq_array = Project.system_frequencies
```

Returns

freq_array: numpy ndarray An array of default project frequencies in Hz

Setting default system frequencies

```
freq_array = np.linspace(0.5, 2, 21)                                    #numpy command to create a
vector
Project.set_system_frequencies(system_freq_ay=freq_array, units_str='GHz')
```

Parameters

system_freq_ay: numpy ndarray An array of default system frequencies

units_str: string, optional. Units for the values in freq_array. Valid strings: 'Hz', 'kHz', 'MHz', 'GHz' Default is 'GHz'

Default System Options

Reading and setting default system options in Options > Default System Options. Exception is frequencies under the RF Frequencies tab, for that refer to this link.

Reading system options:

```
sys_options_dict = Project.system_options_dict
for option_name, option_value in sys_options_dict.items():
    print(option_name,' : ', option_value)
#end for
```

Returns

sys_options_dict: dictionary. Dictionary keys are the option names and dictionary values are the option values

The above code returns:

```
SampleRate  :  8
DataBlockSize  :  100
StopTime  :  0.0
TimeFrame  :  1e-09
OutputBufferSize  :  10000
DefaultImpedance  :  50.0
SysIncludeRFMismatch  :  False
SysAmbientTemp  :  290.0
SysAnnotationSigDigits  :  6
SysRFNoiseMode  :  2
SysRFBMeasThermalNoiseUse  :  0
SysTDMeasThermalNoiseUse  :  0
SysMultithreadRFB  :  1
SysExtrapolateCoSimFreqs  :  1
SysMultithreadRFI  :  1
SysMultithreadTDBlock  :  1
SysMultithreadTDTopLevel  :  1
SamplesPerMeasurement  :  1000000
NormalizeToTimeUnits  :  False
WaveFormsStartFromZero  :  False
AlignWaveFormsToData  :  False
VSS_SamplingFreq  :  7999999999.999999
VSS_OversamplingRate  :  8
VSS_BlockSize  :  100
VSS_UseSimStopTime  :  0
VSS_SimStopTime  :  0.0
VSS_MaxNodeDataAccumulation  :  10000
VSS_MismatchModeling  :  0
VSS_Impedance  :  50.0
VSS_AmbientTemperature  :  290.0
VSS_AnnotationSigDigits  :  6
VSS_LowMemoryThresholdMB  :  500
VSS_RF_NoiseModeling  :  2
VSS_MeasNoiseFloorTemperature  :  290.0
RFB_MeasThermalNoiseFloorUse  :  0
VSS_TD_MeasSpectrumNoiseFloorUse  :  0
VSS_DirectUpConversionGainType  :  0
VSS_CosimS11UseRFB  :  0
VSS_CosimS11UseRFI  :  0
VSS_CosimS11UseTD  :  1
VSS_NonlinearSaturationModel  :  1
VSS_NLAmpFilterImpl  :  0
VSS_AutoRFLinearFreqInterp  :  0
VSS_FMULT_Bx_AutoCOMPATMODE  :  1
RFI_AMP_MaxSelfGenDepth  :  2
RFI_MIXER_MaxSelfGenDepth  :  2
VSS_RFB_UseMultiCore  :  1
VSS_RFI_UseMultiCore  :  1
VSS_TD_Block_UseMultiCore  :  1
VSS_ExtrapolateCoSimFreqs  :  1
VSS_TD_TopLevel_UseMultiCore  :  1
VSS_MaxMeasBufferSize  :  1000000
VSS_NormalizeWaveformsToSymbols  :  0
VSS_WaveformsXAxisFrom0  :  0
VSS_AlignToDataStart  :  0
RFI_UseHarmonicBand  :  1
RFI_HarmonicBand  :  2
RFI_UseFreqAboveFc  :  0
```

```
RFI_FreqAboveFc  :  5000000000.0
RFI_UseFreqBelowFc  :  0
RFI_FreqBelowFc  :  5000000000.0
RFI_UseMaxFreq  :  0
RFI_MaxFreq  :  10000000000.0
RFI_UseMinFreq  :  0
RFI_MinFreq  :  0.0
RFI_UseDBBelowNoise  :  1
RFI_DBBelowNoise  :  10.0
RFI_UseMinDBm_Hz  :  0
RFI_MinDBm_Hz  :  -184.0
RFI_UseMinDBV_sqrtHz  :  0
RFI_MinDBV_sqrtHz  :  -194.0
RFI_UseMaxHarmonic  :  0
RFI_MaxHarmonic  :  4
RFI_UseMaxComponents  :  0
RFI_MaxComponents  :  50
RFI_UseMaxContributionsTracked  :  0
RFI_MaxContributionsTracked  :  20
RFI_AmpsSuppressDC  :  1
RFI_MixersSuppressDC  :  0
RFA_MaxIterations  :  50
RFA_MinChangeInDB  :  0.02
RFA_IterationZeroTolerance  :  1e-12
RFA_MaxControlLoopIterations  :  1000
RFB_NoiseSamplePoints  :  3
VSS_RBW_NFFT_Use  :  0
VSS_RBW  :  1000000.0
VSS_NFFT  :  1000
VSS_VBW_NAVG_Use  :  0
VSS_VBW  :  1000000.0
VSS_NAVG  :  10
VSS_PhaseThreshold  :  -150.0
VSS_PWR_MTR_RMS_Peak  :  0
VSS_PWR_SPEC_RMS_Peak  :  0
```

Setting system options:

The following shows how to modify a single project option and then write the project options to AWRDE

```
Project.set_system_options(option_name, option_value)
```

Parameters

option_name: string

option_value: float, int, boolean

System Diagram

Set System Diagram Object

This section covers properties and methods associated with an individual system diagram. A system diagram object is accessed through the system diagram dictionary as shown:

```
sys_diagrams_dict = Project.system_diagrams_dict     #Read dictionary of system diagrams
sys_diagram = sys_diagrams_dict['AMP sys']                  #Set system diagram object
```

System Diagram Name

Read the name of the selected system diagram

```
sys_diag_name = sys_diagram.system_diagram_name
```

Returns

system_diagram_name: string

System Diagram Add Wire

Add a wire to the selected system diagram

```
sys_diagram.add_wire(start_xy=(100,0), end_xy=(200,0))
```

Parameters

start_xy: tuple. x,y values for wire start point

end_xy: tuple. x,y values for wire end point

System Diagram Remove Wire

Remove a wire from the selected system diagram

```
sys_diagram.remove_wire(start_xy=(100,0), end_xy=(200,0))
```

Parameters

start_xy: tuple. x,y values for wire start point

end_xy: tuple. x,y values for wire end point

System Diagram Wire Segments Dictionary

Returns a dictionary or wire segment x,y positions for all the wires in the selected system diagram

```
wire_segements_dict = sys_diagram.wire_segments_dict
```

Returns

wire_segments_dict: dictionary

keys are wire segment indices

values are a tuple of the wire segment x,y postions: (start x, start y, end x, end y)

System Diagram Element Name List

Returns list of the element names in the selected system diagram

```
elements_list = sys_diagram.element_names_list
```

Returns

elements_list: list. Each item in the list is an element name and ID. Format is: 'Element Name.ID'

System Diagram Element Dictionary

System diagram element are stored in a dictionary that is accessed using the elements_dict method:

```
elements_dict = sys_diagram.elements_dict
for element_name in elements_dict.keys():
    print(element_name)                         #element names are the dictionary keys
#end for
bpf = elements_dict['BPFB.F1']                  #element objects are the dictionary values
```

Returns

elements_dict: dictionary. Dictionary keys are the names of the elements. Dictionary values are element objects

System Diagram Add Element

Add an element to the selected system diagram

```
sys_diagram.add_element(element_name='Name', x_pos, y_pos, rotation=0, flipped=False)
```

Parameters

element_name: string

x_pos, y_pos: float. The x and y location for the equation

rotation: float Angle in degrees. Default=0

flipped: boolean. Default=False

System Diagram Remove Element

Remove an element from the currently selected system diagram

```
sys_diagram.remove_element(self, element_name_n_id='Name.ID')
```

Parameters

element_name_n_id: string. Must be in form 'Name.ID'

System Diagram Element xy Position

The x and y coordinates associated with an element can be read and set as shown:

```
elements_dict = sys_diagram.elements_dict                                   #Read elements dictionary
element = elements_dict['Name.ID']                                          #Set element object variable
xy_position = element.xy_position                                           #Read element's x,y position
element.set_xy_position(xy_position_tuple=(-900,-2200))                     #Set element's x,y position
```

xy_position is read back as a tuple: (x position, y position). x_positon and y_position are type float. When setting the xy position, the pass parameter is also a tuple.

System Diagram Element Enable/Disable

Enable/Disable selected element in the system diagram. Read enabled state of the selected element.

```
elements_dict = sys_diagram.elements_dict                                   #Read elements dictionary
element = elements_dict['Name.ID']                                          #Set element object variable
element.element_enabled = True                                              #Enable element
element.element_enabled = Fasle                                             #Disable element
enabled_state = element.element_enabled                                     #Read enabled state of the
selected element
```

System Diagram Element Nodes Dictionary

Returns dictionary of x,y posistion of the selected element nodes

```
elements_dict = sys_diagram.elements_dict                                    #Read elements dictionary
element = elements_dict['Name.ID']                                           #Set element object variable
nodes_dict = element.element_nodes_dict                                      #Get dictionary of node xy
posistions
```

Returns

nodes_dict: dictionary

keys are the node indices

values are a tuple of the x,y position values

System Diagram Parameter Name List

Returns list of the parameter names for the selected element

```
elements_dict = sys_diagram.elements_dict          #Read elements dictionary
bpf = elements_dict['BPFB.F1']                     #Assign element object variable
parameter_names_list = bpf.parameter_names_list    #Read list of parameter names
```

Returns

parameter_names_list: list. Each item in the list is a parameter name

System Diagram Parameter Dictionary

Element parameters are stored in a dictionary that is accessed using the parameters_dict method:

```
elements_dict = sys_diagram.elements_dict                    #Read elements dictionary
bpf = elements_dict['BPFB.F1']                               #Assign element object variable
bpf_parameters_dict = bpf.parameters_dict                    #Read parameters dictionary for the selected
element
for param_name, param_object in bpf_parameters_dict.items():
    print(param_name, '   ', param_object.value_str)         #dictionary keys are the parameter names and
dictionary values are parameter objects
#end for
```

Returns

parameters_dict: dictionary. Dictionary keys are the names of the parameters. Dictionary values are parameter objects. Parameter values can be accessed using param_object.value_str property

Note: For VSS elements the parameter values must be returned as strings using .value_str property

For the VSS element BPFB (Bandpass Butterworth Filter) the above code produces this output:

```
ID     F1
LOSS     0
N     3
FP1     500
FP2     1500
AP     3.0103
RS     50
RL     50
QU     10000
T_PHY     _TAMB
NOISE     3
NFREQ
NFREQFIR
IMPL     0
SIGMODEL     0
MAXNPOLE
UPRATE     8
FRQALIGN
DIAGDSP     0
GUI     ""
```

System Diagram Modify Parameter Value

Use the parameters dictionary to modify an element parameter value using the modify_parameters_values method as shown:

```
elements_dict = sys_diagram.elements_dict                #Read elements dictionary
bpf = elements_dict['BPFB.F1']                           #Assign element object variable
bpf_parameters_dict = bpf.parameters_dict                #Read parameters dictionary for the selected element
param = bpf_parameters_dict['FP1']                       #set parameter object
param.value_str = '600'                                  #Update paramter value. For system diagram elements,
only string values are valid

value_str = param.value_str                              #Read parameter value as string
```

Float Value

Be careful with parameter float values in system diagram elements. Some parameters will report unexpected results. If float is needed then use:

```
value_float = param.value_float            #Read parameter value as float type
param.value_float = 10                     #Set parameter value as float type
```

System Diagram Expressions List

Returns a list of equation expressions

```
expression_list = sys_diagram.expression_list
```

Returns

expression_list: list. Each list item is a string representing each equation expression in the system diagram

System Diagram Equation Dictionary

System diagram equation objects are accessed using a dictionary using the equations_dict method as shown:

```
equations_dict = sys_diagram.equations_dict       #Read equations dictionary
for eqn_idx, equation in equations_dict.items():  #Dictionary keys are indices,
    print(eqn_idx, ' : ', equation.expression)    #Dictionary values are equation objects
#end for

equation = equations_dict[1]                      #assign equation object
```

Returns

equations_dict: dictionary. Dictionary keys are indices (0 based indexing). Dictionary values are equation objects

System Diagram Add Equation

Add equation to the system diagram. The following adds the equation A = 37.85 to the currently selected system diagram:

```
sys_diagram.add_equation(variable_name='A', variable_type='Variable definition', variable_value=37.85, x_pos=-
1000, y_pos=-2200)
```

Parameters

variable_name: string

variable_type: string. Valid values are 'Variable definition', 'Parameter definition', 'Display value'

Variable definition adds and equation in the form: variable_name = variable_value

Parameter definition adds and equation in the form: variable_name << variable_value. This is a pass parameter in a subcircuit

Display value adds and equation in the form: variable_name: This displays the value of an equation

variable_value: string, int or float

x_pos, y_pos: float. The x and y location for the equation

System Diagram Remove Equation

Removes an equation from the selected system diagram

```
equation_removed = sys_diagram.remove_equation(expression='A = 37.85')
```

Parameters

expression: string. Equation to be removed is identified by its full expression

Returns

equation_removed: boolean. True if equation successfully removed

System Diagram Set Equation Object

In order to perform operations on an individual equation, first an equation object varible must be set as follows:

```
equations_dict = sys_diagram.equations_dict      #Read equations dictionary
equation = equations_dict[3]                     #Set equation object
```

First read in the equations dictionary. Next set the equation variable using the dictionary. The keys of the dictionary are indecies corresponding to each eqution in the system diagram.

System Diagram Equation Expression

The full equation string is termed the expression. To read the selected equation's expression use the following:

```
expr = equation.expression
```

Returns

expression: string. Full equation string, for instance 'x = 10'

System Diagram Equation Name

Equation name can be read and set as shown:

```
equation_name = equation.equation_name    #Read equation name
equation.equation_name = 'x1'             #Set equation name
```

System Diagram Equation Enable/Disable

Enable/Disable equation. Read the enabled state of the selected equation

```
equation.equation_enabled = True                  #Enable the equation
equation.equation_enabled = False                 #Disable the equation
enabled_state = equation.equation_enabled         #Read the equation enabled state
```

System Diagram Equation Value

Equation value can be read and set as shown:

```
equation_value = equation.equation_value   #Read equation value
equation.equation_value = 42.52            #Set equation value (string, int, float are valid types)
```

System Diagram Equation Variable Type

The equaiton variable type can be Variable definition ('='), Parameter definition ('<<') or Display Value (':'). These can be read and set as shown:

```
variable_type = equation.variable_type            #Read variable type
equation.variable_type = 'Parameter definition'   #Set variable type
```

Valid variable types:

Variable definition: This creates an equation in the form 'x = 10'.

Parameter definition: This creates an equation in the for 'x << 10'. This is used for pass parameters in a subcircuit.

Display value: This creates an equation in the form 'x:'. This is used for displaying the value of an equaiton

System Diagram Equation xy Postition

The x and y coordinates associated with an equation can be read and set as shown:

```
xy_position = equation.xy_position                        #Read equation's x,y position
equation.set_xy_position(xy_position_tuple=(-900,-2200))   #Set equation's x,y position
```

xy_position is read back as a tuple: (x position, y position). x_positon and y_position are type float. When setting the xy position, the pass parameter is also a tuple.

System Diagram Use Default Frequencies

Individual system diagrams can either use project default frequencies found under Options > Default System Options > RF Frequencies tab or frequencies set in the individual system diagram. Reading and settin the Use Project Default Frequencies is as shown:

```
use_project_freqs = sys_diagram.use_project_frequencies     #Read Use Project Frequencies
sys_diagram.set_use_project_frequencies(set_state=False)           #Set Use Project Frequencies
```

Reading returns a boolen (True means used project default frequencies). Setting uses a boolen pass parameter.

System Diagram Frequencies

Reading system diagram frequencies:

```
freq_array = sys_diagram.frequencies
```

Returns

freq_array: numpy ndarray An array of default project frequencies in Hz

Setting system diagram frequencies

```
freq_ay = np.linspace(0.5, 1, 11)                          #numpy command to create a vector
sys_diagram.set_frequencies(freq_array=freq_ay, units_str='GHz')
```

Parameters

freq_array: numpy ndarray An array of frequencies

units_str: string, optional. Units for the values in freq_array. Valid strings: 'Hz', 'kHz', 'MHz', 'GHz' Default is 'GHz'

Set Grid Visibility

Turns on or off the schematic grid visibility

```
sys_diagram.set_grid_visible(visibility_boolean)
```

Parmeters

visibility_boolean: Boolean

System Diagram Use Default Options

In the system diagram, the options are organized by tabs in the Options diaglog box. Within each tab are subheading for option groups. The options corresponding to tabs is termed Option Set and the options corresponding to the groups within a tab are termed Option Sub Sets. Each Option Sub Set can be set to either use system project default options or not. The option use project defaults for each Option Sub Set is contained in a dictionary. To read the dictionary use the option_sets_use_project_defaults method as shown

```
use_project_defaults_dict = sys_diagram.option_sets_use_project_defaults
```

Values

use_project_defaults_dict: dictionary of dictionaries

first level dictionary: keys are the Option Set names, values are Option Sub Set objects

second level dictionary: keys are the Option Sub Set name, values are booleans (True means to use project default options for that particular Option Sub Set)

To display the options use project defaults dictionary, use the show_option_sets_use_project_defaults method

```
sys_diagram.show_option_sets_use_project_defaults
```

The above code produces this output:

```
Use Project Defaults
---------------------------------------
Advanced
   Advanced Options : False
   Measurement Options : True
   Multi-core Options : True
Basic
   Measurement Options : True
   Simulation Bandwidth Options : False
   Stopping Options : True
Equations
   Options : False
RF Options
   Co-simulation Options : True
   Default Block Options : True
   Impedance Options : False
   Noise Options : True
RFI/RFB Settings
   Frequency Above Fc : True
   Frequency Below Fc : True
   Harmonic Band : True
   Maximum Components Generated : True
   Maximum Contributions Tracked : True
   Maximum Frequency : True
   Maximum Harmonic Generated : True
   Minimum Frequency : True
   Minimum dBV/sqrt(Hz) : True
   Minimum dBm/Hz : True
   RFB/RFI Convergence : True
   RFB/RFI Measurements : True
   RFI DC Suppression : True
   dB Below Ambient Noise : True
Subcircuit Symbol
```

Setting the dictionary for options use project defaults is as shown:

```
use_project_defaults_dict = sys_diagram.option_sets_use_project_defaults    #Read options use project defaults
dictionary
use_project_defaults_dict['Basic']['Stopping Options'] = True               #Modify dictionary entry
sys_diagram.set_option_sets_use_project_defaults(use_project_defaults_dict)         #Write options use project
defaults dictionary
```

First read the dictionary. Next update the dictionary with syntax use_project_defaults_dict['Option Set Name']['Option Sub Set Name'] = True | False

System Diagram Options

In the system diagram, the options are organized by tabs in the Options diaglog box. Within each tab are subheading for option groups. The options corresponding to tabs is termed Option Set and the options corresponding to the groups within a tab are termed Option Sub Sets. To acces the options themselves, the option sets, option sub sets and options are organized in a dictionary that is read in as shown:

```
sys_diagram_opt_set_dict = sys_diagram.option_sets
```

Values

sys_diagram_opt_set_dict: dictionary of dictionaries of dictionaries

first level dictionary: keys are the Option Set names, values are Option Sub Set objects

second level dictionary: keys are the Option Sub Set name, values are options

thrid level dictionary: keys are the option names, dictionary values are the option values

To display the options , use the show_option_sets method

```
sys_diagram.show_options_sets
```

The above command results in this output:

```
System Diagram Options
---------------------------------------
Advanced
   Advanced Options
      VSS_BlockSize : 110
      VSS_MaxNodeDataAccumulation : 10000
      VSS_AnnotationSigDigits : 6
      VSS_LowMemoryThresholdMB : 500
   Measurement Options
      VSS_MaxMeasBufferSize : 1000000
      VSS_NormalizeWaveformsToSymbols : 0
      VSS_WaveformsXAxisFrom0 : 0
      VSS_AlignToDataStart : 0
   Multi-core Options
      VSS_RFB_UseMultiCore : 1
      VSS_RFI_UseMultiCore : 1
      VSS_TD_Block_UseMultiCore : 1
      VSS_TD_TopLevel_UseMultiCore : 1
Basic
   Measurement Options
      VSS_RBW_NFFT_Use : 0
      VSS_RBW : 1000000.0
      VSS_NFFT : 1000
      VSS_VBW_NAVG_Use : 0
      VSS_VBW : 1000000.0
      VSS_NAVG : 10
      VSS_PhaseThreshold : -150.0
      VSS_PWR_MTR_RMS_Peak : 0
      VSS_PWR_SPEC_RMS_Peak : 0
   Simulation Bandwidth Options
      VSS_SamplingFreq : 10000000000.0
      VSS_OversamplingRate : 10
   Stopping Options
      VSS_UseSimStopTime : 0
      VSS_SimStopTime : 0.0
Equations
   Options
      SpecGlobalEquations : Global Definitions
RF Options
   Co-simulation Options
      VSS_ExtrapolateCoSimFreqs : 1
   Default Block Options
      VSS_DirectUpConversionGainType : 0
      VSS_CosimS11UseRFB : 0
      VSS_CosimS11UseRFI : 0
      VSS_CosimS11UseTD : 1
      VSS_NonlinearSaturationModel : 1
      VSS_NLAmpFilterImpl : 0
      VSS_AutoRFLinearFreqInterp : 0
      VSS_FMULT_Bx_AutoCOMPATMODE : 1
      RFI_AMP_MaxSelfGenDepth : 2
      RFI_MIXER_MaxSelfGenDepth : 2
   Impedance Options
      VSS_MismatchModeling : 0
      VSS_Impedance : 75.0
   Noise Options
      VSS_AmbientTemperature : 290.0
      VSS_RF_NoiseModeling : 2
      VSS_MeasNoiseFloorTemperature : 290.0
      RFB_MeasThermalNoiseFloorUse : 0
      VSS_TD_MeasSpectrumNoiseFloorUse : 0
RFI/RFB Settings
   Frequency Above Fc
      RFI_UseFreqAboveFc : 0
      RFI_FreqAboveFc : 5000000000.0
   Frequency Below Fc
      RFI_UseFreqBelowFc : 0
      RFI_FreqBelowFc : 5000000000.0
```

```
   Harmonic Band
      RFI_UseHarmonicBand : 1
      RFI_HarmonicBand : 2
   Maximum Components Generated
      RFI_UseMaxComponents : 0
      RFI_MaxComponents : 50
   Maximum Contributions Tracked
      RFI_UseMaxContributionsTracked : 0
      RFI_MaxContributionsTracked : 20
   Maximum Frequency
      RFI_UseMaxFreq : 0
      RFI_MaxFreq : 10000000000.0
   Maximum Harmonic Generated
      RFI_UseMaxHarmonic : 0
      RFI_MaxHarmonic : 4
   Minimum Frequency
      RFI_UseMinFreq : 0
      RFI_MinFreq : 0.0
   Minimum dBV/sqrt(Hz)
      RFI_UseMinDBV_sqrtHz : 0
      RFI_MinDBV_sqrtHz : -194.0
   Minimum dBm/Hz
      RFI_UseMinDBm_Hz : 0
      RFI_MinDBm_Hz : -184.0
   RFB/RFI Convergence
      RFA_MaxIterations : 50
      RFA_MinChangeInDB : 0.02
      RFA_IterationZeroTolerance : 1e-12
      RFA_MaxControlLoopIterations : 1000
   RFB/RFI Measurements
      RFB_NoiseSamplePoints : 3
   RFI DC Suppression
      RFI_AmpsSuppressDC : 1
      RFI_MixersSuppressDC : 0
   dB Below Ambient Noise
      RFI_UseDBBelowNoise : 1
      RFI_DBBelowNoise : 10.0
Subcircuit Symbol
```

Setting an option is as shown:

```
sys_diagram_opt_set_dict = sys_diagram.option_sets                                    #Read the option set
dictionary
sys_diagram_opt_set_dict['RF Options']['Impedance Options']['VSS_Impedance'] = 75     #Modify an option
sys_diagram.set_option_sets(sys_diagram_opt_set_dict)                                 #Set the option set
dictionary
```

First read the dictionary. Next update the dictionary with syntax use_project_defaults_dict['Option Set Name']['Option Sub Set Name']['Option Name'] = Option value. Option value type must be a float or int. Then use the set_options_sets method to update AWRDE with the new option values.

Output Equations

Output Equations List

Returns a list of the names of the Output Equation documents in the project.

```
OutputEquations_list = Project.output_equations_list
```

Output Equations Dictionary

Returns a dictionary of output equation documement objects

```
OutputEquations_dict = Project.output_equations_dict
for output_eqn_name in OutputEquations_dict.keys():       #Dictionay keys are output equation document names
    print(output_eqn_name)
#end for
```

Returns

output_equations_dict: dictionary. Dictionary keys are the output equation documents names. Dictionary values are the output equation document objects

Add Output Equation Document

Add an Output Equation document

```
Project.add_output_equations_document(output_eqn_doc_name)
```

Parameters

output_eqn_doc_name: sring. Name of the Output Equation document

Remove Output Equation Document

Remove an output equation document.

```
Project.remove_output_equations_document(output_eqn_doc_name)
```

Parameters

output_eqn_doc_name: sring. Name of the Output Equation document

Output Equation

Set Output Equation Object

This section covers properties and methods associated with an individual output equation document. An output equation document object is accessed as shown:

```
OutputEquations_dict = Project.output_equations_dict      #Read the output equations documents dictionary
output_eqn = output_equations_dict[output_eqn_doc_name]   #Set the output equation document object
```

Output Equation Document Name

Returns name of the selected output equation document.

```
OututEquationDocName = output_eqn.output_eqn_name
```

Equations

Equations within an output equation document can be accessed in same manner as in circuit schematics and system diagrams. Please refer to those sections for details.

Graphs

Graph Name List

Returns a list of all the graph names in the currently opened project.

```
graph_name_list = Project.graph_name_list
```

Returns

graph_name_list: list. Each list item is a graph name

Graphs Dictionary

To access graph objects where each object is an individual graph, use the graphs dictionary as shown:

```
graphs_dict = Project.graph_dict
for graph_name in graphs_dict.keys():
    print(graph_name)
#end for
```

Returns

graphs_dict: dictionary. Dictionary keys are the graph names. Dictionary values are the graph objects

Add Graph

Adding graph to the project uses a unique method call for each graph type as shown. The pass parameter is the graph name as a string type.

```
Project.add_rectangular_graph('Rectangular Graph Name')                          #Add Rectangular graph
Project.add_rectangular_real_imag_graph('Rectangular RealImag Graph Name')       #Add Rectangular Real/Imag
graph
Project.add_smith_chart_graph('Smith Chart Graph Name')                          #Add Smith Chart
Project.add_polar_graph('Polar Graph Name')                                      #Add Polar graph
Project.add_histogram_graph('Histogram Graph Name')                              #Add Histogram graph
Project.add_antenna_plot_graph('Antenna Plot Graph Name')                        #Add Antenna Plot graph
Project.add_tabular_graph('Tabular Graph Name')                                  #Add Tabular graph
Project.add_constellation_graph('Constellation Graph Name')                      #Add Constellation graph
Project.add_3D_graph('3D Graph Name')                                            #Add 3D graph
```

Rename Graph

Rename graph as follows

```
Project.rename_graph(graph_name='Existing Graph Name', new_graph_name='New Graph Name')
```

Parameters

graph_name: string. Name of the existing graph

new_graph_name: string. New graph name

Copy Graph

Copy graph as shown:

```
Project.copy_graph(graph_name='Existing Graph Name', new_graph_name='New Graph Name')
```

Parameters

graph_name: string. Name of the existing graph to be copied

new_graph_name: string. Name of the new graph

Remove Graph

Remove graph as shown:

```
Project.remove_graph(graph_name='Graph Name')
```

Parameters

graph_name: string

Graph

Set Graph Object

This section covers properties and methods associated with an individual graph. A graph object is accessed through the graphs dictionary as shown:

```
graphs_dict = Project.graph_dict                    #Read dictionary of graphs
graph = graphs_dict['Graph Name']                   #Set graph object
```

Graph Name

Return name of selected graph

```
graph_name = graph.graph_name
```

Graph Type

Return graph type:

```
graph_type = graph.graph_type
```

Returns

graph_type: string. Valid types: 'Rectangular', 'Rectangular-Real/Imag', 'Smith Chart', 'Polar', 'Histogram', 'Antenna Plot', 'Tabular', 'Constellation', '3D Plot'

Freeze/Unfreeze Traces

Freeze and Unfreeze traces on currently selected graph

```
graph.freeze_traces(True)    #Freeze traces
graph.freeze_traces(False)   #Unfreeze traces
```

Meaurements

Measurement Name List

Returns a list of measurment names in the selected graph

```
meas_name_list = graph.measurement_name_list
```

Returns

meas_name_list: list Each item is a string representing the full measurment name.

For example:

```
['LPF:DB(|S(2,1)|)', 'LPF:DB(|S(1,1)|)', 'LPF:DB(|S(1,2)|)', 'LPF:DB(|S(2,2)|)']
```

Measurements Dictionary

A dictionary is used to store measurement objects. The dictionary can be accessed as follows:

```
meas_dict = graph.measurements_dict
for meas_idx, meas in meas_dict.items():
    print(meas_idx,' : ', meas.measurement_name)
#end for
```

Returns

meas_dict: dictionary. Dictionary keys are indecies. Dictionary values are the measurment objects

The above code would generate this output:

```
0  :  LPF:DB(|S(2,1)|)
1  :  LPF:DB(|S(1,1)|)
2  :  LPF:DB(|S(1,2)|)
3  :  LPF:DB(|S(2,2)|)
```

Add Measurement

Adding a measurement to the selected graph:

```
graph.add_measurement(source_doc='LPF',measurement='DB(|S(2,2)|)')
```

Parameters

source_doc: string. Name of the document (Schematic, System Diagram, EM Structure, Data File, etc.)

measurement: string.

Remove Measurement

Removeing a measurement from the selected graph

```
measurement_removed = graph.remove_measurement(measurement_name='LPF:DB(|S(2,2)|)')
```

Parameters

measurement_name: string. Full measurement name in the form of source_doc : measurement

Returns

measurement_removed: boolean. True if measurement successfully removed.

Measurement

Set Measurement Object

This section covers properties and methods associated with an individual measurement. A measurement object is accessed through the measurements dictionary as shown:

```
graphs_dict = Project.graph_dict                #Read graphs dictionary
graph = graphs_dict['LPF Frequency Response']   #Set graph object
meas_dict = graph.measurements_dict             #Read measurements dictionary
meas = meas_dict[0]                             #Set measurement object
```

Measurement Name

Returns full measurement name of the selected measurement object

```
measurement_name = meas.measurement_name
```

Returns

measurement_name: string. Full measurement name in the form of source_doc : measurement

Measurement Source Document

Returns the name of the source document (schematic name or system diagram name) associated with the selected measurement

```
source_document = meas.measurement_doc_source
```

Returns

source_document: string

Enable/Disable Measurement

Read measurement enabled state

```
meas_enabled = meas.measurement_enabled
```

Returns

meas_enabled: boolean. True=enabled, False=disabled

Set measurement enabled state:

```
meas.measurement_enabled = False    #Measurement disabled
meas.measurement_enabled = True     #Measurement enabled
```

Parameters

meas_enabled: boolean. True=enabled, False=disabled

Modify Measurement

Change selected measurement

```
meas.modify_measurement(measurement_name)
```

Parmeters

measurement_name: string. The new measurement name in the form source_doc : measurement

Measurement Axis

Read which axis and whether the measurement uses the left of righ axis for the currently selected measurement

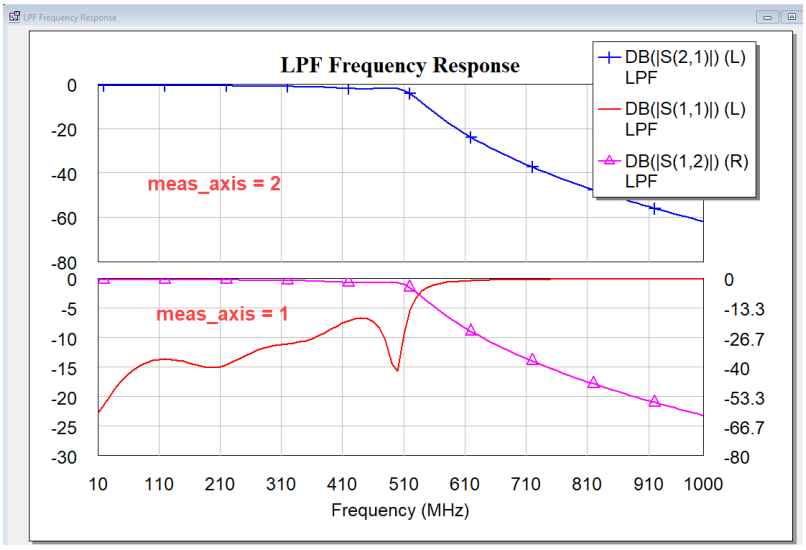
```
meas_axis, meas_on_left_axis = meas.measurement_axis
```

Returns

meas_axis: int. The axis number. Minimum is 1

meas_on_left_axis: boolean. True=measurement is on the left axis, False=measurement is on the right axis

In the following example, the bottom axis is meas_axis=1 and the top axis is meas_axis=2



Set the axis number and left or right axis for the currently selected measurement:

```
meas.set_measurement_axis(axis_index=2, on_left_axis=True)
```

Parameters

axis_index: int. axis number. Minimum=1

on_left_axis: boolean. True=measurement is on the left axis, False=measurement is on the right axis

Number of Traces

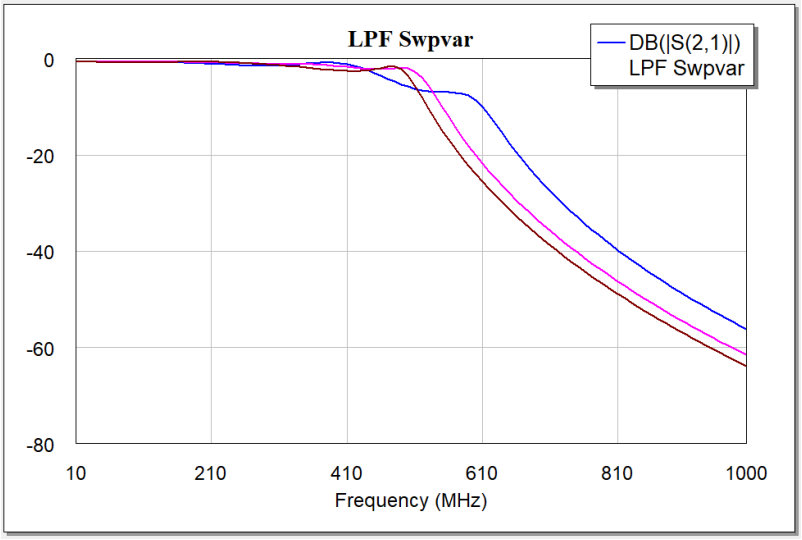
Read the number of traces for the currently selected measurement. Multiple traces are the result of a swept variable simulation using the SWPVAR element.

```
number_traces = meas.num_traces
```

Returns

number_traces: int. Number of traces for the selected measurement

In the following graph, the number of traces is 3:



Read Trace Data

Trace data for the selected measurment is read back as a list of arrays using the trace_data method:

```
import matplotlib.pyplot as plt                         #Import matplotlib
trace_data_list = meas.trace_data                       #Read list of trace data arrays

plt.figure()
plt.grid(True, linestyle=':')
for trace_data_ay in trace_data_list:                   #Extract trace data array from list
    Xdata = trace_data_ay[:,0]                          #X-axis data is column 0
    Ydata = trace_data_ay[:,1]                          #Y-axis data is column 1
    plt.plot(Xdata, Ydata)
#end for
plt.show()
```

Returns

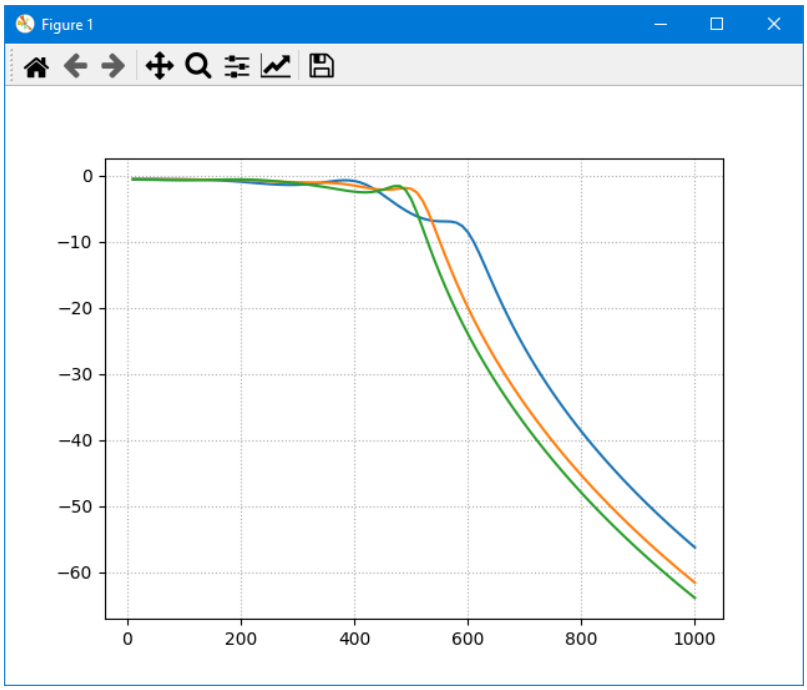
trace_data_list: list. Each item in the list is trace data array. For each array:

column 0: x-axis data

column 1: y-axis data. For complex data, this is the real data

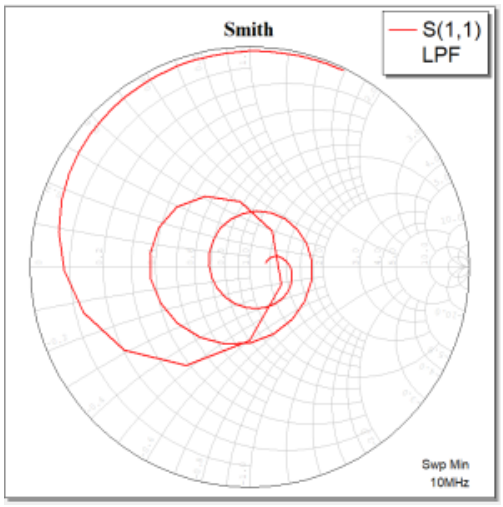
column 2: y-axis data. Only for non scalar data. For complex data this is the imaginary data

The above code generates this output:



Complex Data:

For complex data, for example data returned from a Smith Chart, see the example code below




```
trace_data_list = meas.trace_data                #Read trace data list
for trace_data_ay in trace_data_list:            #Extract trace data array from list
    Xdata = trace_data_ay[:,0]                   #X-axis data is column 0
    Ydata_real = trace_data_ay[:,1]              #Y-axis real is column 1
    Ydata_imag = trace_data_ay[:,2]              #Y-axis imaginary is column 2
#end for
```

Read SWPVAR Parameters

If a document contains a SWPVAR element, the variables associated with each SWPVAR can be read using sweep_var_labels method

```
sweep_var_list = meas.sweep_var_labels
```

Returns

sweep_var_list: list. Each item in the list is a string representing the SWPVAR element VarName parameter

Example ouput:

```
['L2', 'C2']
```

For each trace in the measurment, the SWPVAR information can be read using the sweep_var_trace_info method:

```
trace_info_list = meas.sweep_var_trace_info                              #Read list of dictionaries. Each list
item is information of a trace
for trace_info_dict in trace_info_list:                                  #Extract the dictionary for each trace
    for sweep_var_name, sweep_var_value in trace_info_dict.items():      #Dictionary keys are the variable
names, Dictionary values are the SWPVAR
                                                                         #values
        print(sweep_var_name, ' : ',sweep_var_value)
    #end for
    print('')
#end for
```

Returns

trace_info_list: list of dictionaries.

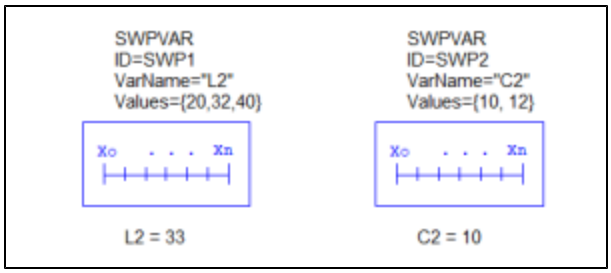
Each list item is a dictionary for each trace in the measurement

Each trace dictionary: dictionary keys are the variable names, dictionary values are the variable values

Eaxmple ouput using the above code:

```
L2  :  20.0
C2  :  10.0

L2  :  32.0
C2  :  10.0

L2  :  40.0
C2  :  10.0

L2  :  20.0
C2  :  12.0

L2  :  32.0
C2  :  12.0

L2  :  40.0
C2  :  12.0
```

This is associated with a schematic with these two SWPVAR elements:



Axes

Axis Name List

Returns a list of the axis names for the currently selected graph:

```
axes_name_list = graph.axes_name_list
```

Returns

axes_name_list: list. Each list item is the name of the axis

Example output

```
['X1', 'Left1', 'Right1', 'Left2', 'Right2']
```

Axis Dictionary

Each axis is stored as an object in a dictionary. The axes dictionary can be read with axes_dict method:

```
axes_dict = graph.axes_dict
```

Returns

axes_dict: dictionary. Dictionary keys are the axis names. Dictionary values are axis objects

Axis

Set Axis Object

This section covers properties and methods associated with an individual axis for the selected graph. An axis object is accessed through the axes dictionary as shown:

```
graphs_dict = Project.graph_dict                        #Read graphs dictionary
graph = graphs_dict['LPF Frequency Response']           #Set graph object
axes_dict = graph.axes_dict                             #Read axes dictionary
axis = axes_dict['X1']                                  #set axis object
```

Valid keys for axes_dict: 'X1', 'Left1', 'Right1' For multiple axes in a graph then 'Left2', 'Right2', etc. are valid

Axis Name

Read name of the selected axis

```
axis_name = axis.axis_name
```

Auto Scale

Read auto scale setting of the selected axis

```
auto_scale = axis.axis_auto_scale
```

Set auto scale of the selected axis

```
axis.axis_auto_scale = False       #auto scale disabled
axis.axis_auto_scale = True        #auto scale enabled
```

Min Scale

Read minimum scale value of the selected axis

```
min_scale = axis.axis_minimum_scale
```

Set minimum scale value of the selected axis

```
axis.axis_minimum_scale = 20   #Parameter type: int or float
```

Max Scale

Read maximum scale value of the selected axis

```
max_scale = axis.axis_maximum_scale
```

Set maximum scale value of the selected axis

```
axis.axis_maximum_scale = 900    #Parameter type: int or float
```

Grid Step

Read grid step value of the selected axis. This corresponds to the "Divisions > Step" parameter in the Graph Options

```
grid_step = axis.axis_grid_step
```

Set grid step value of the selected axis

```
axis.axis_grid_step = 50    #Parameter type: int or float
```

Markers

Marker Name List

Returns a list of the marker names of all the markers in the selected graph

```
marker_name_list = graph.marker_name_list
```

Returns

marker_name_list: list. Each list item is the name of the marker

Example output:

```
['m2', 'm1']
```

Marker Dictionary

Marker objects are stored in a dictionary. To access the dictionary, use the markers_dict method:

```
marker_dict = graph.markers_dict
for marker_index, marker in marker_dict.items():
    print(marker_index, ' : ', marker.marker_name)
#end for
```

Returns

maker_dict: dictionary. Dictionary keys are indices. Dictionary values are marker objects

The above code produces this output:

```
0  :  m2
1  :  m1
```

Add Marker

Add marker to measurement in the selected graph

```
graph.add_marker(measurement='LPF:DB(|S(2,1)|)', sweep_value=500e6, trace_index=0)
```

Parameters

measurement: string. Full measurement name in the form: source_doc : measurement

sweep_value: float, int. The x-axis value for the marker placement. Must be in base units

trace_index: int,optional. For multiple traces due to SWPVAR in document, sets the trace index in which to place the marker. Default = 0

Remove Marker

Remove an individual marker

```
marker_removed = graph.remove_marker(marker_name='m3')
```

Parameters

marker_name: string

Remove all markers from the graph

```
graph.remove_all_markers
```

Marker

Set Marker Object

This section covers properties and methods associated with an individual marker for the selected graph. An axis object is accessed through the markers dictionary as shown:

```
graphs_dict = Project.graph_dict                    #Read graphs dictionary
graph = graphs_dict['LPF Frequency Response']       #Set graph object
marker_dict = graph.markers_dict                    #Read markers dictionary
marker = marker_dict[0]                             #Set marker object
```

marker_dict keys are indices.

Marker Name

Name of the selected marker

```
marker_name = marker.marker_name
```

Returns

marker_name: string. Name of the selected marker

Marker Measurement

Measurement name associated with the selected marker

```
marker_measurement = marker.marker_measurement
```

Returns

marker_measurement: string. Measurement in the form: source_doc : measurement

Marker Sweep Value

Read the x-axis value of the marker

```
marker_sweep_value = marker.marker_sweep_value
```

Returns

marker_sweep_value: float. x-axis value of the marker in base units

Set the x-axis value of the marker

```
marker.marker_sweep_value = 500e6
```

Parameters

sweep_value: float,int. x-axis value of the marker in base units

Marker to Max

Set marker to maximum y-axis value

```
marker.marker_to_max
```

Marker to Min

Set marker to mimimum y-axis value

```
marker.marker_to_min
```

Marker Trace Index

Read the trace index of the selected marker.

```
trace_index = marker.trace_index
```

Set the trace index of the selected marker

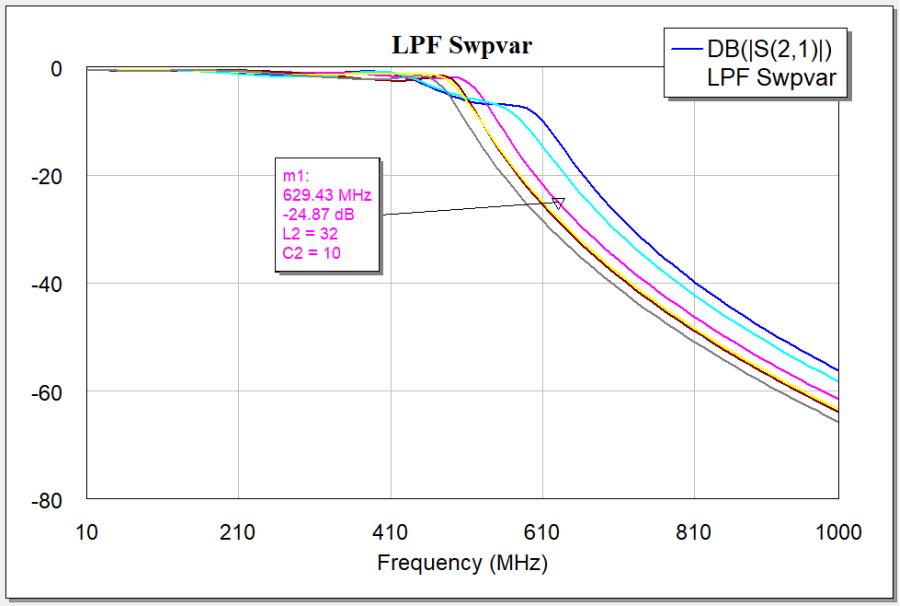
```
marker.trace_index = 1    #Parameter must be integer. Zero based
```

Read Marker Value

Read marker data text

```
marker_data_text = marker.marker_data_text
```

For the following marker, the output is as shown:

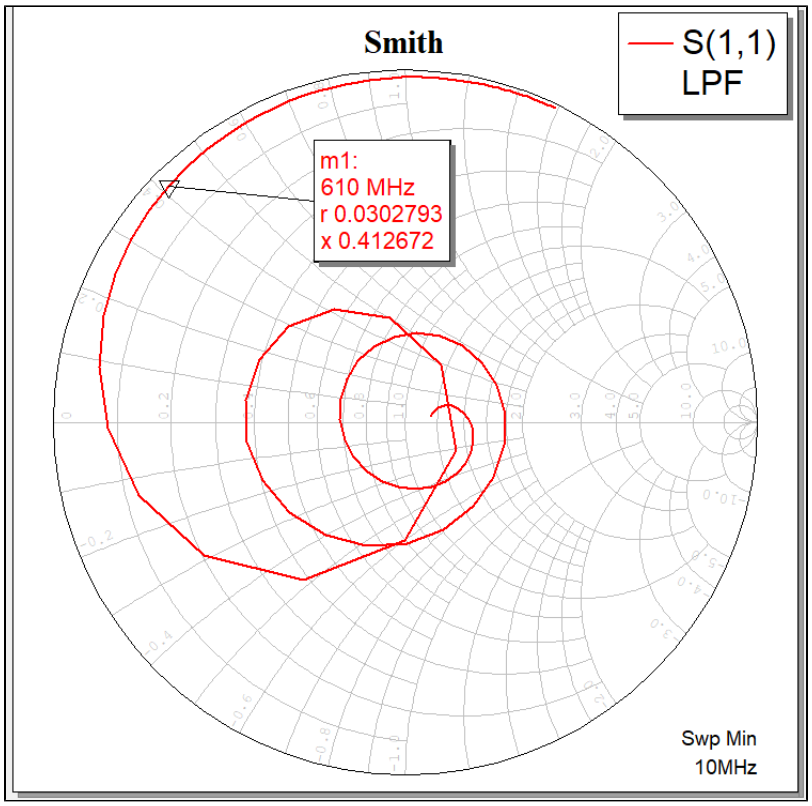



```
629.43 MHz
-24.87 dB
L2 = 32
C2 = 10
```

Read marker x and y data:

```
marker_data_value_x = marker.marker_data_value_x
print(marker_data_value_x)

marker_data_value_y1 = marker.marker_data_value_y1
print(marker_data_value_y1)

marker_data_value_y2 = marker.marker_data_value_y2     #Does not apply to scalar data
print(marker_data_value_y2)
```

For the following graph, the output of the above code is as shown:




```
610000000.0
0.030279273683722244
0.41267239478914347
```

Data Files

Data File Names List

Returns a list of all the data files names in the currently opened project.

```
data_file_names_list = Project.data_file_names_list
```

Returns

data_file_name_list: list. Each list item is a data file name

Data Files Dictionary

To access data file objects where each object is an individual data file, use the data files dictionary as shown:

```
data_files_dict = Project.data_files_dict           #Read data files dictionary
for data_file_name in data_files_dict.keys():       #Dictionay keys are data file names
    print(data_file_name)
#end for
```

Returns

data files_dict: dictionary. Dictionary keys are the data file names. Dictionary values are the data file objects

Add Data File

Adding a data file to the project uses a unique method call for each data file type as shown. The pass parameter is the data file name as a string type.

```
Project.add_dc_iv_file('DC-IV File Name')                        #Add DC-IV File
Project.add_dscr_file('DSCR File Name')                          #Add DSCR File
Project.add_gmdif_file('GMDIF File Name')                        #Add Generalized MDIF File
Project.add_gmdif_nport_file('GMDOF N-Port Name')                #Add Generalized N-Port File
Project.add_mdif_file('MDIF Name')                               #Add MDIF File
Project.add_raw_port_parameters_file('Raw Port Parameter Name')  #Add Raw Port Parameters File
Project.add_text_file('Text Name')                               #Add Text File
Project.add_touchstone_file('Touchstone Name')                   #Add Touchstone File
```

Remove Data File

Remove a data file as shown:

```
Project.remove_data_file(data_file_name='Data File Name')
```

Parameters

data_file_name: string

Rename Data File

Rename data file as follows

```
Project.rename_data_file(data_file_name='Existing Data File Name', new_data_file_name='New Data File Name')
```

Parameters

dtata_name: string. Name of the existing data file

new_data_file_name: string. New data file name

Copy Data File

Copy data file as shown

```
Project.copy_data_file(data_file_name='Existing Data File Name', new_data_file_name='New Data File Name')
```

Parameters

data_file_name: string. Name of the existing data file to be copied

new_data_file_name: string. Name of the data_file

Import Data File

Imports a data file into the project from a file in a directory

```
Project.import_data_file(data_file_name='imported file', file_path='Full path name', data_file_type='Text File')
```

Parameters

data_file_name: string. Name of the data file as it will appear in the project

file_path: string. Full data file path

data_file_type: string. Valid values:

'DC-IV File', 'DSCR File', 'Generalized MDIF Data File', 'Generalized MDIF N-Port File', 'MDIF File', 'Raw Port Parameters File', 'Text File', 'Touchstone File'

Data File

Set Data File Object

This section covers properties and methods associated with an individual data file. A data file object is accessed through the data files dictionary as shown:

```
data_files_dict = Project.data_files_dict         #Read dictionary of data files
data_file = data_files_dict['Data File Name']     #Set the data file object
```

Data File Name

Return name of selected data file

```
data_file_name = data_file.data_file_name
```

Data File Type

Returns the data file type

```
data_file_type = data_file.data_file_type
```

Returns

data_file_type: string. Valid values

'DC-IV File', 'DSCR File', 'Generalized MDIF Data File', 'Generalized MDIF N-Port File', 'MDIF File', 'Raw Port Parameters File', 'Text File', 'Touchstone File'

Export Data File

Export selected data file.

```
data_file.export_data_file(file_path='Full path name')
```

Parameters

file_path: string. Full data file path

Global Definitions

Global Definitions Documents List

Returns a list of all the Global Definitions Documents in the project.

```
global_def_list = Project.global_definitions_list
```

Returns

global_definitions_list: list. Each item is the name of a global definitions document

Global Definitions Documents Dictionary

To access global defintions document objects where each object is an individual global definition document, use the global definitions document dictionary as shown:

```
global_def_dict = Project.global_definitions_dict
for global_def_name in global_def_dict.keys():       #Dictionay keys are global definition document names
    print(global_def_name)
#end for
```

Returns

global_definitions_dict: dictionary. Dictionary keys are the global definitions documents names. Dictionary values are the global definition document objects

Add Global Definitions Document

Add a global definitions document to the project.

```
Project.add_global_definitions_document(global_def_doc_name)
```

Parameters

global_def_doc_name: string

Remove Global Definitions Document

Remove a global definitions document from the project

```
Project.remove_global_definitions_document(global_def_doc_name)
```

Parameters

global_def_doc_name: string

Global Definitions Document

Set Global Definitions Document Object

This section covers properties and methods associated with an individual global definitions document. A global definitions document object is accessed as shown:

```
global_def_dict = Project.global_definitions_dict    #Read the global definitions documents dictionary
global_def = global_def_dict['Global Definitions']   #Set the global defintions document object
```

Elements and Equations

Elements and Equations in a global definitions document can be accessed in the same manner as in circuit schematics and system diagrams. Please refer to those sections for details.

Optimization

Optimization Variables Dictionary

Read a dictionary of all variables that are configured for optimization

```
opt_var_dict = Project.optimization_variables_dict
```

Returns

optimization_variables_dict: dictionary

Keys: Index

Values: Dictionary of fields:

Document Name

Document Type: Schematic, System Diagram, Global Definition

Element Type: Element or Equation

Element Name

Variable Name

Nominal Value

Constrained: True or False

Lower Constraint

Upper Constraint

Step Size

Optimization Print Variables

Print the dictionary of variables that are configured for optimization

```
opt_var_dict = Project.optimization_variables_dict       #Read the dictionary of optimization variables
Project.print_optimization_variables(opt_var_dict)       #Print the dictionary of optimization variables
```

Example output:

```
Optimization Variable Index =  0
    Document Name = LPF
    Document Type = Circuit Schematic
    Element Type = Element
    Element Name : Parameter Name = CAP.C1 : C
    Constrained = False
    Nominal Value = 5.5999999999999996e-12
    Lower Constraint = 0.0
    Upper Constraint = 0.0
    Step Size = 0.0

Optimization Variable Index =  1
    Document Name = LPF
    Document Type = Circuit Schematic
    Element Type = Element
    Element Name : Parameter Name = CAP.C2 : C
    Constrained = False
    Nominal Value = 4.8e-12
    Lower Constraint = 0.0
    Upper Constraint = 0.0
    Step Size = 0.0
```

Optimization Type List

Read list of all the optimization types available

```
opt_type_list = Project.optimization_type_list   #Read list of all the optimization types available
```

Returns

optimization_type_list: list. Items are the names of the optimization types.

Optimization Type

Read/set the optimization type

```
opt_type_list = Project.optimization_type_dict         #Read the list of optimization types
optimization_type_name = opt_type_list[0]              #Assign the optimization type name
Project.optimization_type = optimization_type_name     #Set the optimization type.
optimization_type = Project.optimization_type          #Read the optimization type.
```

Optimization Type Properties

For the optimization types that have unique properties, the properties can be read in as a dictionary

```
opt_properties_dict = Project.optimization_type_properties
```

Returns

optimization_type_properties: dictionary. Keys are the property names and dictionary values are the property values

Read/Set example

```
#Read the properties names and values
for prop_name, prop_value in opt_properties_dict.items():
    print(prop_name, prop_value)
#end for

#Set a property value
opt_properties_dict['Converge Tolerance'] = 0.002     #Update the dictionary with the new value
Project.optimization_update_type_properties()         #Apply the property changes to the project
```

Optimization Maximum Iterations

Read/set the optimizer maximum iterations

```
Project.optimization_max_iterations = 100               #Set the optimizer maximum iterations
max_iterations = Project.optimization_max_iterations    #Read the optimizer maximum iterations
```

Optimization Show All Iterations

Enable/Disable the optimizer *Show all iterations* option

```
Project.optimization_show_all_iterations = True|False       #Set enabled state
enabled_state = Project.optimization_show_all_iterations    #Read enbabled state
```

Optmization Stop At Minimum Error

Enable/Disable the optimizer *Stop at minimum error* option

```
Project.optimization_stop_at_minimum_error = True|False       #Set enabled state
enabled_state = Project.optimization_stop_at_minimum_error    #Read enbabled state
```

Optimization Stop On Simulation Error

Enable/Disable the optimizer *Stop on simulation errors* option

```
Project.optimization_stop_on_simulation_error = True|False       #Set enabled state
enabled_state = Project.optimization_stop_on_simulation_error    #Read enbabled state
```

Optimization Cancel On Stop Request

Enable/Disable the optimizer *Cancel current iteration on stop request* option

```
Project.optimization_cancel_on_stop_request = True|False       #Set enabled state
enabled_state = Project.optimization_cancel_on_stop_request    #Read enbabled state
```

Optimization Log To File

Enable/Disable the optimizer *Log to file* option

```
Project.optimization_log_to_file = True|False       #Set enabled state
enabled_state = Project.optimization_log_to_file    #Read enbabled state
```

Optimization Read Best Cost

Read the best optimization cost

```
cost = Project.optimization_best_cost
```

Optimization Read Current Cost

Read the current optimization cost

```
cost = Project.optimization_cost
```

Optimzation Start/Stop

Start and Stop the optimization. Read the current state of the optimizer running.

```
Project.optimization_start = True           #Start the optimization
Project.optimization_start = False          #Stop the optimization
start_state = Project.optimization_start    #Read the current state of the optimizer running as boolean
```

Optimization Round Variables

Round the optimization variable values.

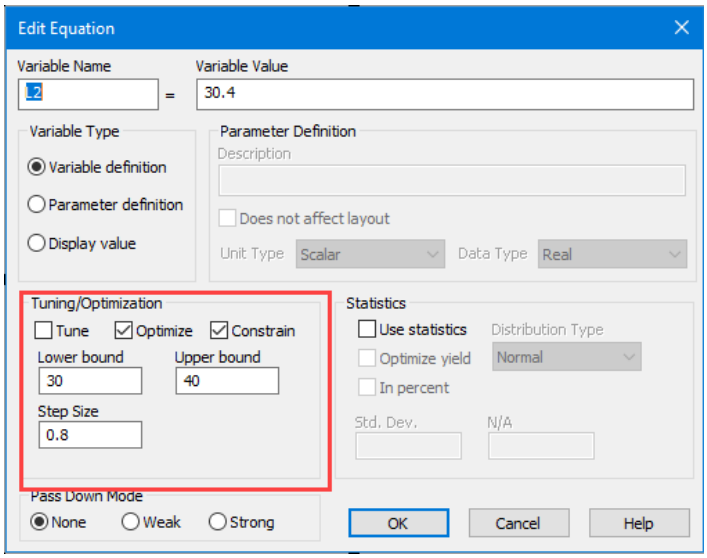
```
Project.optimization_round(num_significant_digits)
```

Parameters

num_significant_digits: int.

Parameter/Equation Optimization Settings

Element parameters and equations can both be configured for optimization setting highlighted here:



Elements and Equations can originate from circuit schematics, system diagrams and global equations (EM schematics will be supported in the future). Refer to those sections for establishing object variables for parameters or equations. For instance if the elements and equations are contained in a Global Definitions document, the procedure is:

```
global_def_dict = Project.global_definitions_dict       #Get dictionary of Global Definitions Documents
global_def = global_def_dict['Global Definitions']      #Assign global definition document object variable
elem_dict = global_def.elements_dict                   #Get dictionary of elements in the global definitions
document
elem = elem_dict['MSUB.Msub']                           #Assign element object variable
param_dict = elem.parameters_dict                       #Get parameter dictionary for the selected element
param = param_dict['Rho']                               #Assign param object variable

equations_dict = global_def.equations_dict              #Get dictionary of equations in the global definitions
document
equation = equations_dict[0]                            #Assign equation object variable
```

A similar proceedure is also used for circuit schematics and system diagrams

Parameter/Equation Optimization Enable/Disable

Enable or disable the parameter or equation for opitimization

```
param.optimize_enabled = True|False         #Set optimization enabled or disabled for parameters
enabled_state = param.optimize_enabled      #Read the optimization enabled state for parameters

equation.optimize_enabled = True|False      #Set optimization enabled or disabled for equations
enabled_state = equation.optimize_enabled   #Read the optimization enabled state for equations
```

Paramter/Equation Optimization Constraint Enable/Disable

Enable or disable the parameter or equation Constrain state

```
param.constrain = True|False           #Set constrain state enabled or disabled for parameters
enabled_state = param.constrain        #Read the constrain enabled state for parameters

equation.constrain = True|False        #Set constrain state enabled or disabled for equations
enabled_state = equation.constrain     #Read the constrain enabled state for equations
```

Parameter/Equation Optimization Lower/Upper Constraint

```
param.lower_constraint = 0.7                #Set lower contraint value for parameters
param.upper_constraint = 1.3                #Set upper contraint value for parameters
lower_constraint = param.lower_constraint   #Read lower contraint value for parameters
upper_constraint = param.upper_constraint   #Read upper contraint value for parameters

equation.lower_constraint = 0.7                #Set lower contraint value for equations
equation.upper_constraint = 1.3                #Set upper contraint value for equations
lower_constraint = equation.lower_constraint   #Read lower contraint value for equations
upper_constraint = equation.upper_constraint   #Read upper contraint value for equations
```

Parameter/Equation Optimization Constraint Step Size

```
param.step_size = 0.1               #Set step size contraint for parameters
step_size = param.step_size         #Read step size contraint for parameters

equation.step_size = 0.1               #Set step size contraint for equations
step_size = equation.step_size         #Read step size contraint for equations
```

Optimization Goals

Optimization Goals Dictionary

Read a dictionary of optimization goals

```
opt_goals_dict = Project.optimization_goals_dict
```

Returns

optimization_goals_dict: dictionary. Keys are index values, dictionary values are goal objects

Add Optimization Goal

Add an optimization goal

```
Project.add_optimization_goal(measurement_name, goal_type, goal_xrange, goal_yrange, goal_weight, goal_l_factor)
```

Parameters

measurement_name: string. Name of an active measurement in one of the graphs in the project. A list of active measurement can be read using the Project.get_all_active_measuments command below

goal_type: string. '>','<', or '='

goal_xrange: tuple (xStart, xStop)   Values are in base units

goal_yrange: tuple (yStart, yStop)

goal_weight: float. The weight parameter for the goal

goal_l_factor: float: the L Factor parameter for the goal

```
active_meas_list = Project.get_all_active_measurements   #Get list of all the active measurment names
```

Remove Optimization Goal

Remove an optimization goal from the project

```
Project.remove_optimization_goal(optimization_goal_name)
```

Parameters

optimization_goal_name: string. Full name of the goal (see below for obtaining the goal name)

Set Optimization Goal Object

Reading/Setting optimization goal parameters requires that an individual goal object variable is established.

```
opt_goals_dict = Project.optimization_goals_dict     #Read dictionary of optimization goals in the project
goal = opt_goals_dict[0]                             #Assign optimization goal object variable
```

Optimization Goal Name

Read the

```
goal_name = goal.goal_name      #Read goal name as a string
```

Optimization Goal Measurement Name

Returns the measurement name associated with the selected goal.

```
measurement_name = goal.goal_measurement_name
```

Optimization Goal Source Docuement Name

Returns the souce document name associated with the selected goal.

```
source_doc_name = goal.goal_source_document
```

Optimization Goal Type

Read the goal type

```
goal_type = goal.goal_type      #Read goal type as a string. Values are: '<', '>', or '='
```

Note: goal type cannot be set by the API. To change type, the goal must be removed and then added with the correct goal type

Optimization Goal Enable/Disable

```
goal.goal_enabled = True|False    #Set the goal enabled state
enabled_stage = goal.goal_enables #Read the goal enabled state
```

Optimization Goal X Range

Read/set the goal X Range

```
goal.goal_xrange = (xStart, xStop)    #Set the goal X Range. Values are in base units. Parameter passed as a
tuple (xStart, xStop)
gaol_xrange = goal.goal_xrange        #Read the gaol X Range. Values are in base units. Parameter read as a
tuple (xStart, xStop)
```

Optimization Goal Y Range

Read/set the goal Y Range

```
goal.goal_yrange = (yStart, yStop)    #Set the goal Y Range. Values are in base units. Parameter passed as a
tuple (yStart, yStop)
gaol_yrange = goal.goal_yrange        #Read the gaol Y Range. Values are in base units. Parameter read as a
tuple (yStart, yStop)
```

Note: by setting yStart, yStop to different values, the goal slope indicator will be checked

Optimization Goal Weight

Read/set the gaol weigth

```
goal.goal_weight = 3             #Set the goal weight
goal_weight = goal.goal_weight   #Read the goal weight
```

Optimization Goal L Factor

Read/Set the goal L Factor

```
goal.goal_l_factor = 3               #Set the goal L Factor
goal_l_factor = goal.goal_l_factor   #Read the goal L Factor
```

Optimization Goal Cost

Read the cost for the selected goal. Cost is computed after an optimization

```
cost = goal.goal_cost
```

Optimization Log File Parser

Functions that read and parse the JSON fomatted optimization log file.

Optimization Log File Selection

Calls WIndows File Browser set to the project logs file. User selects desired log file (.json file name extension)

```
Dir_n_File = Project.optimization_log_file_select
```

Returns

Dir_n_File: string. Full path and file name of the selected optimization log file

Optimization Log File Extract

Returns JSON formated text file for the selected optimization log file name

```
OptLogFile = Project.optimization_extract_log_file(Dir_n_File)
```

Parameters

Dir_n_File: string. Full path and file name of the selected optimization log file. Must have .json extension

Returns

OptLogFile: Text File

Optimization Log File Parameters

Returns dictionary of parameters from the optimization log file

```
OptParam_dict = Project.optimization_log_file_params_dict(OptLogFile)   #Read optimization log file parameters
dictionary
#
for param_name, param_val in OptParam_dict.items():                     #Print dictionary
    if param_name == 'Properties':
        print('Properties')
        for prop_name, prop_val in param_val.items():
            print('    ',prop_name+':   ',prop_val)
        #end for
    else:
        print(param_name+':  ',param_val)
    #end if
#end for
```

Parameters

OptLogFile: Text File. JSON fomatted optimization log file

Returns

OptParam_dict: dictionary

Keys: Optimization Type, MaxIterations, NumIterations, Properties, Stop at min error, Stop on simulation error, Cancel on stop request, Show all iterations

Example output

```
Optimization Type:   Random
MaxIterations:   100
NumIterations:   100
Properties
     Number of Parallel Jobs:    1.0
     Variation Factor:    0.2
Stop at min error:   False
Stop on simulation error:   True
Cancel on stop request:   True
Show all iterations:   False
```

Optimization Log File Goals

Returns dictionary of optiization goals as read from the optimization log file.

```
OptGoal_dict = Project.optimization_log_file_goals_dict(OptLogFile)  #Read optimization log file goals
dictionary
#
for goal_name, goal_dict in OptGoal_dict.items():                    #Print dictionary
    print(goal_name)
    for goal_val_name, goal_val in goal_dict.items():
        print('   ',goal_val_name+':  ',goal_val)
    #end for
#end for
```

Parameters

OptLogFile: Text File. JSON fomatted optimization log file

Returns

OptGoal_dict: dctionary

Keys: Type, Meas, Enabled, Weight, L, Xstart, Ystart, Xstop, Ystop, Tag

Example output:

```
Goal1
    Type:   gt
    Meas:   Lumped_Elliptic_LPF:DB(|S(2,1)|)
    Enabled:   False
    Weight:   2.0
    L:   2.0
    Xstart:   1000000000.0
    Ystart:   -0.5
    Xstop:   2000000000.0
    Ystop:   -2.5
    Tag:
Goal2
    Type:   lt
    Meas:   Lumped_Elliptic_LPF:DB(|S(1,1)|)
    Enabled:   False
    Weight:   1.0
    L:   2.0
    Xstart:   200000000.0
    Ystart:   -18.0
    Xstop:   2000000000.0
    Ystop:   -18.0
    Tag:
```

Optimization Log File Variables

Returns dictionary of otimization variables from the optimizaation log file.

```
OptVariables_dict = Project.optimization_log_file_opt_variables_dict(OptLogFile)
for var_name, var_val_dict in OptVariables_dict.items():
    print(var_name)
    for var_val_name, var_val in var_val_dict.items():
        print('   ',var_val_name+':  ',var_val)
    #end for
#end for
```

Parameters

OptLogFile: Text File. JSON fomatted optimization log file

Returns

OptVariables_dict: dictionary

Keys: Variable Name

Values: Dictionary of values

Keys: Initial Value, Min Constrain Value, Max Constrain Value, Step Size, Final Value

Example Output

```
@Distributed_Elliptic_LPF\\C2_Ro
    Initial Value:   299.0
    Min Constrain Value:   100.0
    Max Constrain Value:   300.0
    Step Size:   0.0
    Final Value:   299.0
@Distributed_Elliptic_LPF\\C2_Theta
    Initial Value:   119.0
    Min Constrain Value:   30.0
    Max Constrain Value:   120.0
    Step Size:   0.0
    Final Value:   120.0
```

Optimization Log File Cost

Returns array of optimization costs for each iteration. Read from the optization log file.

```
Cost_ay = Project.optimization_log_file_cost_array(OptLogFile)
```

Parameters

OptLogFile: Text File. JSON fomatted optimization log file

Returns

Cost_ay: array of floats. Optimization cost for each optimization iteration

Yield Analysis

Yield Variables Dictionary

Read a dictionary of all variables that are configured to use statistics.

```
yield_var_dict = Project.yield_variables_dict
```

Returns

yield_variables_dict : Dictionary

Keys: Index

Values: Dictionary of fields

Document Name

Document Type: Schematic, System Diagram, EM Structure, Global Definition

Element Type: Element or Equation

Element Name

Variable Name

Optimize Yield : Boolean

Tolerance in Pct : Boolean

Statistical Variation : float

Statistical Variation 2 : float

Distribution : string (Uniform, Normal, Log-Normal, Discrete, Normal Minus Tol, Normal Clipped)

Print Yield Variables Dictionary

Print the yield variables dictionary

```
yield_var_dict = Project.yield_variables_dict      #Get yield variables dictionary
Project.print_yield_variables(yield_var_dict)      #Print yield variables dictionary
```

Yield Analysis Type List

Read list of all Yield Analysis types

```
yield_type_list = Project.yield_type_list   #Read list of all the yield analysis types available
```

Yield Analysis Type

Read/set the yield analysis type

```
yield_type_list = Project.yield_type_dict         #Read the list of optimization types
yield_type_name = yield_type_list[0]              #Assign the yield type name
Project.yield_type = yield_type_name              #Set the yield analysis type.
yield_type = Project.yield_type                   #Read the yield analysis type.
```

Yield Analysis Type Properties

For the yiled analysis types that have unique properties, the properties can be read in as a dictionary

```
yield_properties_dict = Project.yield_type_properties
```

Returns

yield_type_properties: dictionary. Keys are the property names and dictionary values are the property values

Read/set example

```
#Read the properties names and values
for prop_name, prop_value in yield_properties_dict.items():
    print(prop_name, prop_value)
#end for

#Set a property value
yield_properties_dict['Dampening (0.5 - 1.0)'] = 0.6    #Update the dictionary with the new value
Project.yield_update_type_properties()           #Apply the property changes to the project
```

Yield Analysis Maximum Iterations

Read/set the optimizer maximum iterations

```
Project.yield_max_iterations = 100              #Set the yield analysis maximum iterations
max_iterations = Project.yield_max_iterations   #Read the yield analysis maximum iterations
```

Yield Analysis Start/Stop

Start and Stop the optimization. Read the current state of the optimizer running.

```
Project.yield_analysis_start = True           #Start the yield analysis
Project.yield_analysis_start = False          #Stop the yield analysis
start_state = Project.yield_analysis_start    #Read the current state of the yield analysis running as boolean
```

Yield Goals

Yield Goals Dictionary

Read a dictionary of yield goals

```
opt_goals_dict = Project.yield_goals_dict
```

Returns

yield_goals_dict: dictionary. Keys are index values, dictionary values are goal objects

Add Yield Goal

Add a yield goal

```
Project.add_yield_goal(measurement_name, goal_type, goal_xrange, goal_yrange)
```

Parameters

measurement_name: string. Name of an active measurement in one of the graphs in the project. A list of active measurement can be read using the Project.get_all_active_measuments command below

goal_type: string. '>','<', or '='

goal_xrange: tuple (xStart, xStop)   Values are in base units

goal_yrange: tuple (yStart, yStop)

```
active_meas_list = Project.get_all_active_measurements   #Get list of all the active measurment names
```

Remove Yield Goal

Remove a yield goal from the project

```
Project.remove_yield_goal(yield_goal_name)
```

Parameters

yield_goal_name: string. Full name of the goal (see below for obtaining the goal name)

Set Yield Goal Object

Reading/Setting yield goal parameters requires that an individual goal object variable is established.

```
opt_goals_dict = Project.yield_goals_dict     #Read dictionary of optimization goals in the project
goal = opt_goals_dict[0]                            #Assign optimization goal object variable
```

Yield Goal Name

Read the

```
goal_name = goal.goal_name      #Read goal name as a string
```

Yield Goal Measurement Name

Returns the measurement name associated with the selected goal.

```
measurement_name = goal.goal_measurement_name
```

Yield Goal Source Docuement Name

Returns the souce document name associated with the selected goal.

```
source_doc_name = goal.goal_source_document
```

Yield Goal Type

Read the goal type

```
goal_type = goal.goal_type      #Read goal type as a string. Values are: '<', '>', or '='
```

Note: goal type cannot be set by the API. To change type, the goal must be removed and then added with the correct goal type

Yield Goal Enable/Disable

```
goal.goal_enabled = True|False    #Set the goal enabled state
enabled_stage = goal.goal_enables #Read the goal enabled state
```

Yield Goal X Range

Read/set the goal X Range

```
goal.goal_xrange = (xStart, xStop)    #Set the goal X Range. Values are in base units. Parameter passed as a
tuple (xStart, xStop)
gaol_xrange = goal.goal_xrange        #Read the gaol X Range. Values are in base units. Parameter read as a
tuple (xStart, xStop)
```

Yield Goal Y Range

Read/set the goal Y Range

```
goal.goal_yrange = (yStart, yStop)    #Set the goal Y Range. Values are in base units. Parameter passed as a
tuple (yStart, yStop)
gaol_yrange = goal.goal_yrange        #Read the gaol Y Range. Values are in base units. Parameter read as a
tuple (yStart, yStop)
```

Note: by setting yStart, yStop to different values, the goal slope indicator will be checked

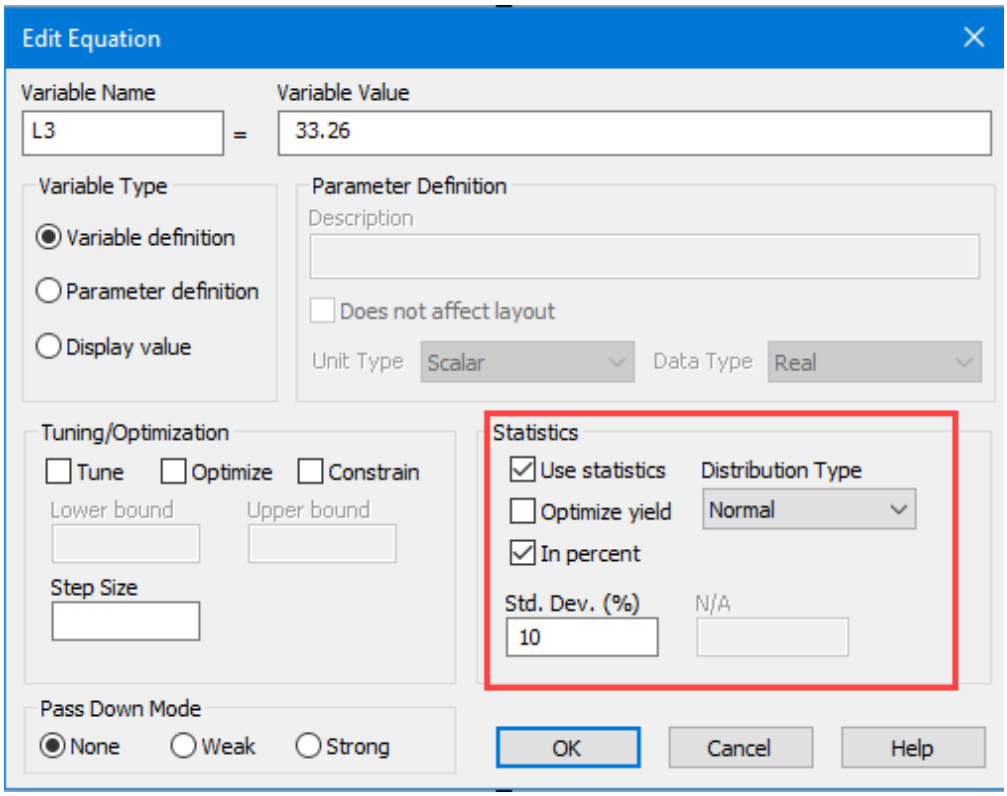
Yield Goal Cost

Read the cost for the selected goal. Cost is computed after an optimization

```
cost = goal.goal_cost
```

Yield Parameter/Equation Settings

Element parameters and equations can both be configured for yield analysis setttings highlighted here:



Elements and Equations can originate from circuit schematics, system diagrams and global equations (EM schematics will be supported in the future). Refer to those sections for establishing object variables for parameters or equations. For instance if the elements and equations are contained in a Global Definitions document, the procedure is:

```
global_def_dict = Project.global_definitions_dict       #Get dictionary of Global Definitions Documents
global_def = global_def_dict['Global Definitions']      #Assign global definition document object variable

elem_dict = global_def.elements_dict                    #Get dictionary of elements in the global definitions
document
elem = elem_dict['MSUB.Msub']                           #Assign element object variable
param_dict = elem.parameters_dict                       #Get parameter dictionary for the selected element
param = param_dict['Rho']                               #Assign param object variable

equations_dict = global_def.equations_dict              #Get dictionary of equations in the global definitions
document
equation = equations_dict[0]                            #Assign equation object variable
```

A similar proceedure is also used for circuit schematics and system diagrams

Yield Parameter/Equation Use Statistics

Enable or Disable the parameter or equaiton for yield analysis

```
param.use_statistics = True|False          #Set enabled state for use statistics for element parameters
UseStatistics = param.use_statistics       #Read use statistics enabled state

equation.use_statistics = True|False       #Set enabled state for use statistics for equations
UseStatistics = equation.use_statistics    #Read use statistics enabled state
```

Yield Parameter/Equation Yield Optimization

Enable or Disable the parameter or equaiton for yield optimization

```
param.yield_optimize = True|False          #Set enabled state for yield optimization
YieldOptimize = param.yield_optimize       #Read yield optimization state

equation.yield_optimize = True|False       #Set enabled state for yield optimization
YieldOptimize = equation.yield_optimize    #Read yield optimization state
```

Yield Parameter/Equation Yield Distribution

Set and read the Yield Distribution

```
param.yield_distribution = yield_distribution_str       #Set distribution for element parameters
yield_distribution_str = param.yield_distribution       #Read distribution enabled state

equation.yield_distribution = yield_distribution_str    #Set distribution for equations
yield_distribution_str = equation.yield_distribution    #Read distribution enabled state
```

Valid yield_distribution_str:

'Uniform', 'Normal', 'Log-Normal', Discrete', 'Normal Minus Tol', 'Normal Clipped'

Yield Parameter/Equation Yield Tolerance in Percent

Read and set the Tolerance In Percent enabled state

```
param.yield_tol_in_percent = True|False              #Set enabled state for tolerance in percent for element
parameters
ToleranceInPercent = param.yield_tol_in_percent      #Read tolerance in percent enabled state

equation.yield_tol_in_percent = True|False           #Set enabled state for tolerance in percent for equations
ToleranceInPercent = equation.yield_tol_in_percent   #Read tolerance in percent enabled state
```

Yield Parameter/Equation Statistical Variation

```
param.yield_statistical_variation = statistical_variation_val      #Set statistical variation value for element
parameters
StatisticalVariation = param.yield_statistical_variation           #Read statistical variation vlaue
param.yield_statistical_variation_2 = statistical_variation_val    #Set 2nd statistical variation value for
element parameters
StatisticalVariation2 = param.yield_statistical_variation_2        #Read 2nd statistical variation vlaue

equation.yield_statistical_variation = statistical_variation_val      #Set statistical variation value for
equations
StatisticalVariation = equation.yield_statistical_variation           #Read statistical variation vlaue
equation.yield_statistical_variation_2 = statistical_variation_val    #Set 2nd statistical variation value for
equations
StatisticalVariation2 = equation.yield_statistical_variation_2        #Read 2nd statistical variation vlaue
```

statistical_variation_val is an integer or float