AWR Scripting in Python: Using the AWRDE API Scripting Guide

Introduction

The AWRDE API Scripting Guide is written using SAX Basic which is compatible with the Visual Basic for Applications (VBA) syntax and coding style. For Python programming, the scripting guide will still serve as the API reference and for the most part the AWR specific command syntax will still apply. This document shows how to interpret the AWRDE API Scripting Guide for Python coding.

This document assumes that **pyawr** library has been installed and that you are using a Python IDE that supports code-completion. Please refer to the AWR Scripting in Python: Getting Started and Installation document for **pyawr** installation instructions.

Importing pyawr

The syntax for importing **pyawr** is shown here:

Importing pyawr

```
import pyawr.mwoffice as mwo                    #import pyawr module
```

First import **pyawr.mwoffice** into the Python code and give it the alias, **mwo**.

Linking to AWRDE

Assign **pyawr.mwoffice.CMWOffice()** class to object variable. Object variable **awrde** will be used in these examples.

There are three different ways to link to AWRDE

1. mwo.CMOffice()

   This is the default method and should be used when only one version of AWRDE is installed on the computer. The latest installed version of AWRDE on the computer is opened with this command

```
awrde = mwo.CMWOffice()                         #create awrde object
```

2. Version number

   If multiple versions of AWRDE are installed on the computer, use the version option as shown:

```
awrde = mwo.CMWOffice(version='16.0')     #create awrde object
```

3. Class ID (CLSID)

   If multiple instances of AWRDE are running simultaneously, then a unique identifier for each AWRDE instance is the CLSID number. The syntax for linking to a specific AWRDE instance is shown here:

```
awrde = mwo.CMWOffice(clsid='{4D6BF14E-A8FB-4AF2-85F3-AA9141DC7310}')
```

   The CLSID number for the AWRDE instance can be determined from a VBA script. Create a code module using the scripting editor within AWRDE. Copy the following code and run. The CLSID number will be displayed in the scripting editor's immediate window

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

Project Collection Object

In the API Scripting Guide, most collection objects are found under the Project collection object. For instance the graphs collection object in SAX Baisc would use this syntax:

Project.Graphs #SAX Basic

In Python, the object variable must precede the Project collection object. So, the same command in Python would be:

awrde.Project.Graphs #Python

In this Python example the **Schematic 1** is being assigned to the **s** collection object variable:

```
s = awrde.Project.Schematics('Schematic 1')   #using the awrde object to access schematic in the open project
```

SAX Basic Collection Objects

The AWRDE API is strongly written around the VBA compatible SAX Basic collection of objects style of coding. Python will still use these objects, however there are some Python syntax conventions that take precedence. The list of objects can be found in the AWRDE API Scripting Guide > AWR Design Design Component API > Objects List

VBA uses the **For Each** looping structure and most examples in the AWRDE API Scripting Guide are written using this construct. Here is an example from the API document for looping through all the schematics in the project and printing their names:

SAX Basic Print Schematic Names

```
        Dim s As Schematic
        For Each s In Project.Schematics
                Debug.Print s.Name
        Next s
```

In Python the same functionality can be written in two different ways:

Python Print Schematic Names

```
#Method 1
for s in awrde.Project.Schematics:
    print(s.Name)     #Code-completion does not work here

#Method 2
NumSchematics = awrde.project.Schematics.Count #Number of schematics in open project
for i in range(NumSchematics):
    s = awrde.Project.Schematics[i]
    print(s.Name)     #Code-completion works here
```

In Method 1, the code is more compact, however inside the **for** loop, code-completion for AWRDE commands does not work. In Method 2, code-completion inside the **for** loop does work.

In SAX Basic, and extensively in the API Scripting Guide, the **Set** command is used to assign a collection object to a variable:

SAX Basic Set Collection Object

```
Dim s As Schematic
Set s = Project.Schematics("Schematic 1")
```

In Python, the syntax is as shown

Python Set Collection Object

```
s1 = awrde.Project.Schematics("Schematic 1") #Double quotation marks
s2 = awrde.Project.Schematics('Schematic 2') #Single quotation marks
```

Note that the string literal for SAX Biasic is enclosed double quotation marks. For Python, either single quotation or double quotation marks are acceptable.

Indexing Collection Objects

Indexing associated with collection objects is supported with two different styles. When using parentheses () , the index starts at 1 and when when using square brackets [], the index starts at 0. The following two commands both print the name of the first schematic in the project:

Indexing Compare

```
print(awrde.Project.Schematics(1).Name)  #Indexing starts with 1
print(awrde.Project.Schematics[0].Name)  #Indexing starts with 0
```

Here are examples using both styles for printing the names of all the schematics in the project:

Index Looping

```
#Indexing using parentheses ()
for i in range(1, awrde.Project.Schematics.Count + 1):
    print(awrde.Project.Schematics(i).Name)     # i = 1,2,...

#Indexing using brackets []
for i in range(awrde.Project.Schematics.Count):
    print(awrde.Project.Schematics[i].Name)     # i = 0,1,...
```

The square bracket indexing styling allows lists to be created with constructs that comply with Python methods of accessing elements within a list or array. For instance, using an index of [-1] references the last schematic in the project

```
s = awrde.Project.Schematics[-1]  #last schematic
```

Lists can be created from collection of objects using standard range constructs as shown here:

Indexing Examples

```
schem_array = awrde.Project.Schematics[:]      #Creates list of all the schematics
schem_array = awrde.Project.Schematics[::-1]   #Creates list of all the schematics in reverse order
schem_array = awrde.Project.Schematics[::2]    #Creates list of every 2nd schematic
schem_array = awrde.Project.Schematics[:-1]    #Creates list of all schematics except the last one
schem_array = awrde.Project.Schematics[1:4]    #Creates list of schematics 2 through 4
schem_array = awrde.Project.Schematics[2::]    #Creates list of all schematics except the 1st and 2nd ones
```

Enumerations

Function parameters use enumerations. The list of enumerations can be found in AWRDE API Scripting Guide > AWR Design Design Component API > Enumerations List. Enumeration List functions are found directly pyawr.mwoffice module. Shown here is an example of reporting the file type of an existing data file:

Enumeration

```
import pyawr.mwoffice as mwo                 #import pyawr module
awrde = mwo.CMWOffice()                      #create awrde object

DataFile = awrde.Project.DataFiles('Data File 1')   #Assign Data File to object variable
DataFileType = mwo.mwDataFileType(DataFile.Type)    #Use enumeration function for Data File Types
print(DataFileType)
```

Print output: mwDataFileType.mwDFT_SNP

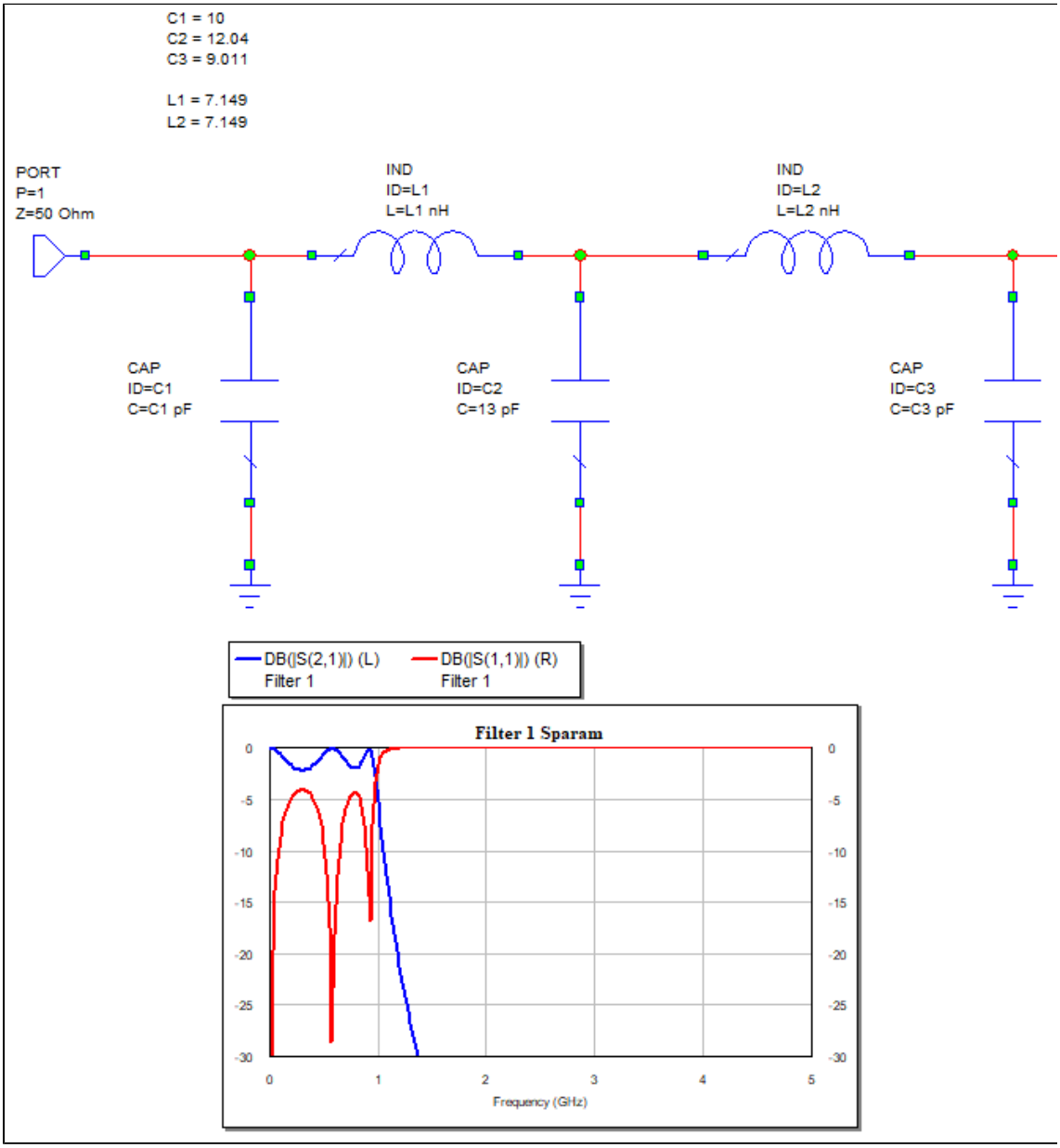
This indicates that the data file type is a **Touchstone File**

Here is an example of adding a new data file using the enumeration for setting the data file type to 'text'.

```
awrde.Project.DataFiles.AddNew('Data File 2', mwo.mwDataFileType.mwDFT_TXT)
```

Example Code

The following example shows some basic operations using **pyawr** to interface with AWRDE. Included are updating global frequencies, updating schematic equation, updating schematic element value, simulating the project, reading graph data, and plotting the data. This example assumes that AWRDE is running and is open to a project labelled *pyawr_PlotMeasurementData.emp.* This project includes a simple filter structure with S-parameters as the graphed data. Variable *C1* will be updated in the example code.



This first section imports pyawr, matplotlib and numpy. Then awrde object is created. Next, the active project name is read and the project name read back is compared against the desired project name.

```
import pyawr.mwoffice as mwo
import matplotlib.pyplot as plt
import numpy as np

#Establish AWRDE link----------------------------------------------------------
DesiredProjectName = 'pyawr_PlotMeasurementData.emp'
awrde = mwo.CMWOffice()               #create awrde object
ProjectName = awrde.Project.Name      #Read active project name
if ProjectName != DesiredProjectName: #Check to make sure active project is the desire one
    raise RuntimeError('Incorrect Opened Project')
```

This section demonstrates creating a frequency array using the numpy linspace function. The frequency array is then used to update global project frequencies.

```
#Change Project Frequencies---------------------------------------------------------
StartFreq = 0
StopFreq = 5e9
NumFreqPts = 201
Freq_ay = np.linspace(StartFreq, StopFreq, NumFreqPts)  #Create Frequency vector
awrde.Project.Frequencies.AddMultiple(Freq_ay)          #Write frequency vector to
                                                        #Options > Project Options > Freqencies tab
```

This section shows how to update an equation located in a schematic as well as updating an element parameter.

```
#Update Schematic Equation----------------------------------------------------------
SchematicName = 'Filter 1'
New_C1_Value = 10
schem = awrde.Project.Schematics(SchematicName) #assign schematic object to SchematicName
NumEquations = schem.Equations.Count            #Get number of equations in schematic
for i in range(NumEquations):                   #Loop through equations
    if 'C1' in schem.Equations[i].Expression:   #Locate equation for variable C1
        schem.Equations[i].Expression = 'C1 = '+str(New_C1_Value) #Update equation value

#Update element value---------------------------------------------------------------
New_C2_Value = 13
elem = schem.Elements('CAP.C2')                          #assign element object to capaicitor C2
elem.Parameters('C').ValueAsDouble = New_C2_Value*1e-12  #update C2 value
```

In this final section, the project is simulated, measurement data is read from a graph, and then matplotlib is used to plot the data.

```
#Simulate---------------------------------------------------------------------------
awrde.Project.Simulator.Analyze()      #Run simulation

#Read Measurement Data from AWRDE Graph---------------------------------------------
graph = awrde.Project.Graphs('Filter 1 Sparam')  #assign graph object
meas = graph.Measurements[0]                     #assign measurment object

NumTracePts = meas.XPointCount                   #Get nummber of points  in measurement data
Xdata_ay = np.zeros((NumTracePts))               #Allocate X data array
Ydata_ay = np.zeros((NumTracePts))               #Allocate Y data array

Xdata_ay = meas.XValues                          #Read in X trace data
Ydata_ay = meas.YValues(1)                       #Read in Y trace data

#Plot Data--------------------------------------------------------------------------
plt.plot(Xdata_ay, Ydata_ay)                #Plot using matplotlib method
plt.show()
```