AWR Scripting in Python: Getting Started and Installation

Introduction

The AWR Design Environment (AWRDE) contains a built-in Scripting Development Environment (SDE) based on the SAX Basic programming language. SAX Basic is compatible with Visual Basic for Applications. Scripting automates tasks by controlling items such as schematics, EM structures, VSS system diagrams, and layout as well as retrieving data from the measurements. Python gives the user an alternate programming language in which to write scripts taking advantage of Python's access to a large body of readily available, open-source libraries. AWRDE has an extensive Application Programming Interface (API) to facilitate scripting and the API commands available with VBA can also be accessed using Python. Unlike the built-in SDE, Python is scripted using an external Integrated Development Environment (IDE) of the user's choosing. The Python executable as well the Python library modules reside outside of AWRDE, giving the user flexibility in configuring the Python environment as desired.

SAX Basic versus Python for AWR Scripting

SAX Basic and Python both have their advantages. Choosing which language to use depends on the user's experience level and the requirements that the script must perform.

Advantages of using SAX Basic within AWRDE

- For users who do not know Python.
- For users that do not know any programming languages, SAX Basic integrated within AWRDE can be quicker to develop a working script.
- Easy to create a user interface (UI).
- Can call the script from a menu pick or toolbar icon within AWRDE.
- Script can be saved within the project.
- AWR has a large body of example scripts. A developed script may already exist, or parts of an existing script can be used as a starting place.

Advantages of using Python

- For users who know Python and do not want to learn VBA.
- Access to Python libraries and utilities not available with SAX Basic
    - high level math functions
    - digital signal processing functions
    - highly flexible plotting
- Faster compute speed on large data arrays.
- Python as a programming language is growing in popularity and as such, there is a large community in which to get help.
- Analysis in a Jupyter Notebook.

Interfacing Between AWRDE and Python

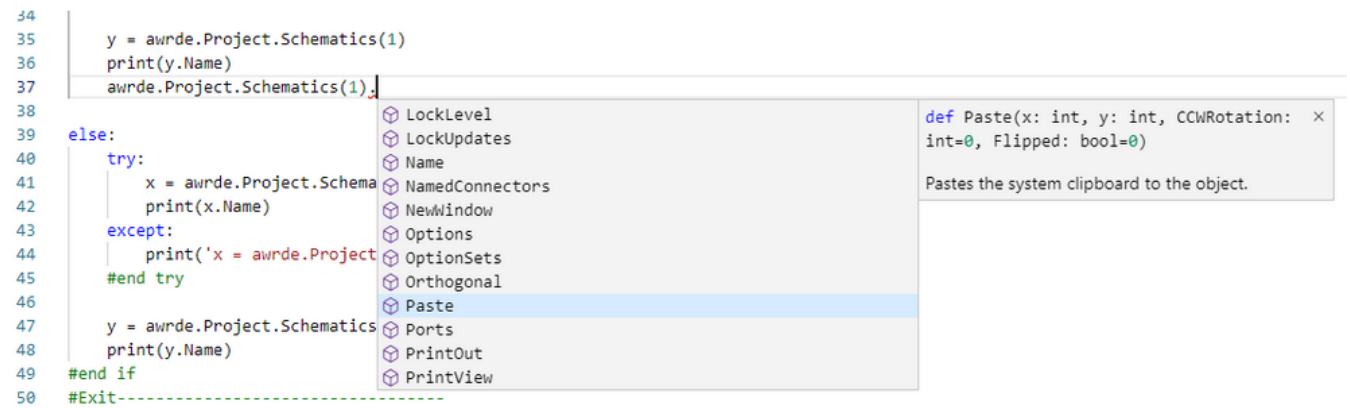
AWRDE uses the Common Object Model (COM) interface as a means of interacting with external applications. COM is a well established standard for many popular applications. One of these applications is Python through the standard win32com library.

The ability to script AWRDE functions using Python has been available for quite some time. With this method, the win32com module is imported directly into the Python script. One disadvantage of this approach is the lack of many convenience items such as code-completion.

An alternate approach for interfacing between Python and AWRDE is by means of the **pyawr** libary. The **pyawr** library incorporates win32com and adds AWR specific API functionality. Some of the features that **pyawr** provide include:

- Code-completion showing lists of API methods and variables associated with a particular API function
- Code-completion description and tips for the methods and variables
- Vector and array indexing that complies with Python coding conventions
- Descriptive error messages

The following is an example showing code-completion that highlights the list of variables and methods associated with the Project.Schematics function. Description associated with the Paste method is also shown



The **pyawr** library is installed using standard Python package installers such as *pip*. Once installed, the **pyawr** module is imported into the Python code similar to other commonly used Python modules. Unlike built-in VBA scripts, Python scripts are developed and executed from within an external IDE that supports code-completion. Some examples of IDE's that support code-completion with **pyawr** are Visual Studio Code (VS Code), PyCharm, and Spyder, however this list is not exhaustive.

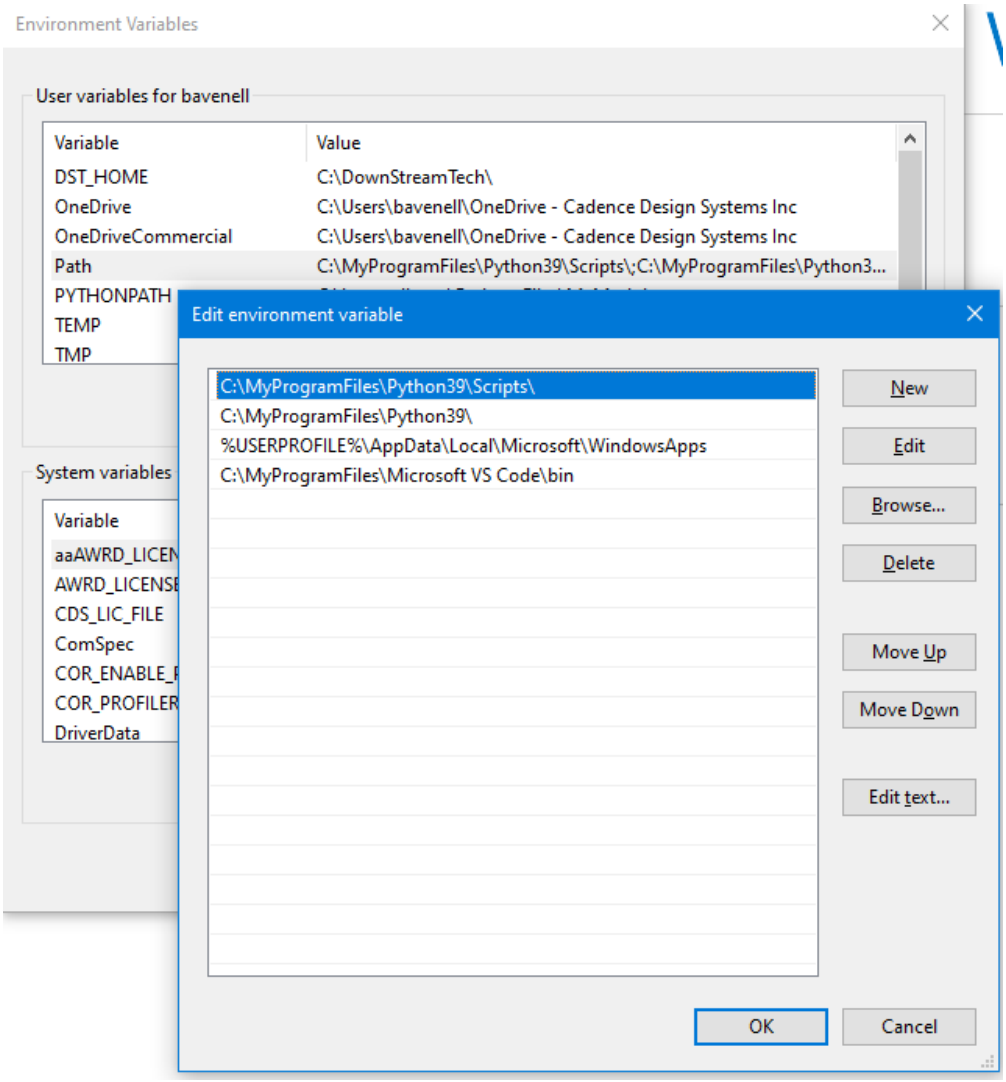
Installation Prerequisites

Before installing the **pyawr** library, there are a number of required programs and libraries that need to be installed first.

Python and pip

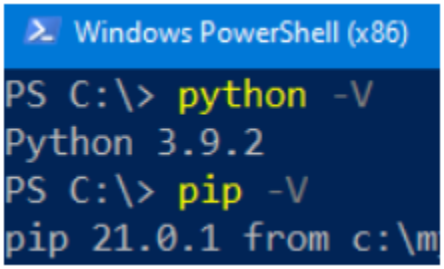
Python 3.7 or later must be already installed. Python installer can either be downloaded from Python Org or from a distribution package such as Anaconda. The instructions here will follow using Python Org as the Python installation where pip is the primary means of installing individual modules.

Once Python is installed, you will need to add <Python install directory> and <Python install directory>/Scripts to the Windows path Environmental Variables



To verify Python and pip are installed, in a command window type

```
python -V
pip -V
```



If during a subsequent module installation process, a message is given to upgrade pip, use this command

```
pip install --upgrade pip
```

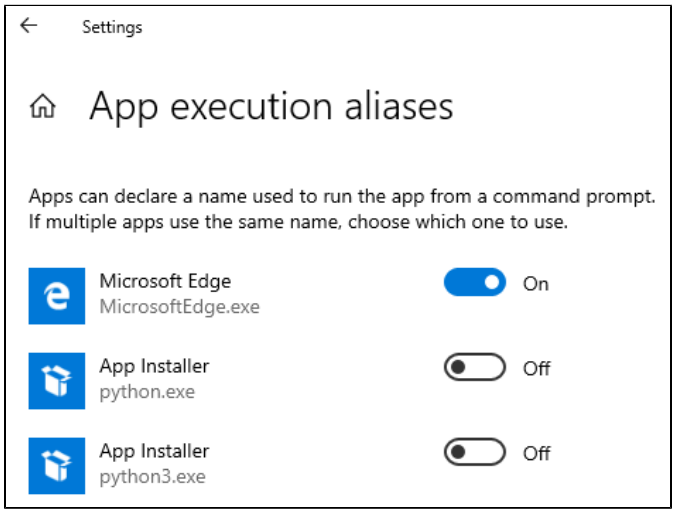
Note when using the Anaconda distribution package:

If the above commands are not recognized, then you will need to disable Python application execution aliasing

In Windows, Start > Settings > Apps > Apps & features

Click on **App execution aliases**

Set **python.exe** and **python3.exe** to the off position

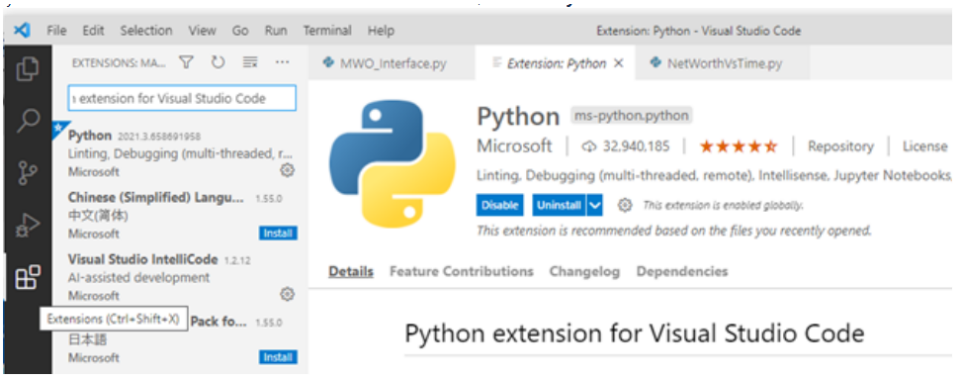


Python IDE

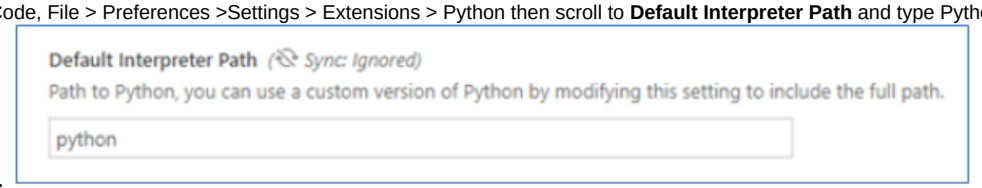
There are several IDEs that support code-completion and you can use your preferred IDE if you would like. For users who do not currently use a particular IDE, Visual Studio Code is the recommended IDE, however PyCharm, Spyder, and Wing are IDEs that have been verified to work with **pyawr** and code-completion. The following are instructions for installing these IDES and configuring for **pyawr** code-completion.

VS Code

1. Download from https://code.visualstudio.com, then install
2. Install Python extension from within VS Code: File > Preferences > Extensions, search for **Python extension for Visual Studio Code** and click **In stall**

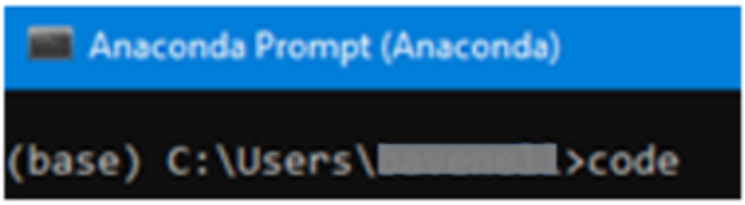
a. 

3. In VS Code, File > Preferences >Settings > Extensions > Python then scroll to **Default Interpreter Path** and type Python into the entry area

a. 

Pylance with VS Code is not recommended when using **pyawr**. If installed, please uninstall from File > Preferences > Extensions and search for **Pylance**

If the Anaconda distribution package was used to install Python, open the Anaconda prompt from the Windows Start Menu > Anaconda. In the Anaconda prompt window, type *code*



PyCharm

Download PyCharm from https://www.jetbrains.com/pycharm and install

pyawr is automatically recognized and there are no additional settings required to enable code-completion

Wing

Download Wing Python IDE from https://wingware.com and install

pyawr is automatically recognized and there are no additional settings required to enable code-completion

Spyder

1. Download Spyder from https://www.spyder-ide.org and install
2. In a command window, type *pip install spyder-kernels*
3. In Spyder, Tools > Preferences > Completion and linting > Completion tab, check **Enable code completion**, **Show completion details**, and **Show completions on the fly**
4. In Spyder, Tools > Preferences > Completion and linting> Advanced tab, uncheck **Enable Kite**

pyawr Installation Procedure

First install **win32com** and configure it for the AWR Design Environment using this procedure:

1. In a command window, type

```
pip install pywin32
```

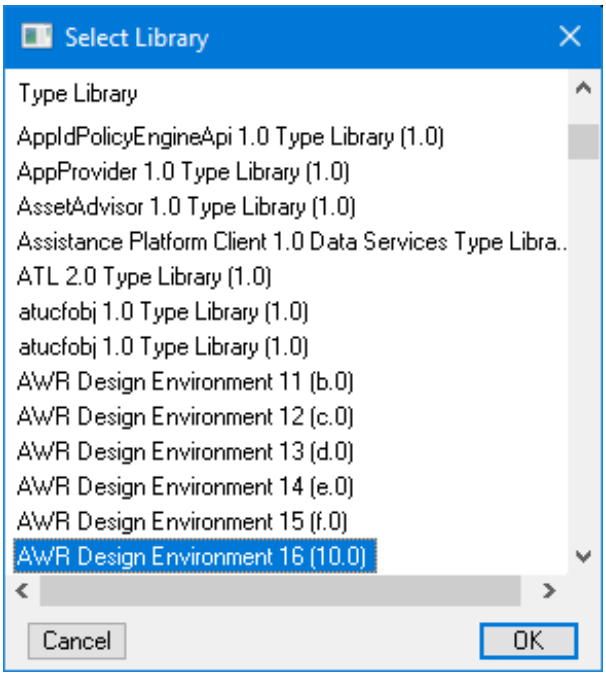
For Python versions older than 3.9.5:

1. cd to <python install directory>/Lib/site-packages/win32com/client

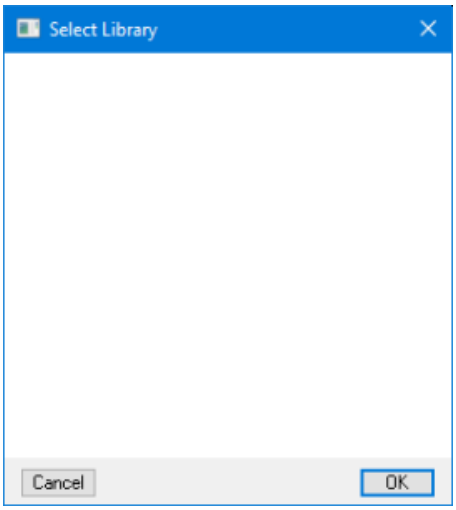
2. type

```
python .\makepy.py
```

3. In the **Select Library** dialog box, select the AWR Design Environment that corresponds with the AWRDE version you are using



If the **Select Library** dialog box is blank, then ignore this step and click **OK**



Next, install the **pyawr** library:

In a command window, type

```
pip install pyawr
```

If upgrading from previous installation of pyawr.

```
pip install --upgrade pyawr
```

If upgrading and a message is displayed that requirement is already satisfied, then uninstall and install pyawr

```
pip uninstall pyawr
pip install pyawr
```

To view pyawr change history, see https://pypi.org/project/pyawr/

Example Python Script to Verify pyawr

Copy and paste the following into the Python editor

```
import pyawr.mwoffice as mwo
awrde = mwo.CMWOffice()

NumSchem = awrde.Project.Schematics.Count
for s_idx in range(NumSchem):
    schem = awrde.Project.Schematics[s_idx]
    print(schem.Name)
```

Open AWRDE and then open the built-in example LPF_Lumped.emp

Run the Python script

In the Python output terminal, the text output should be: **LPF**