Using Microsoft Powershell to Control AWRDE

Summary

Windows Powershell is a new command line shell and task-based scripting technology which is designed to work directly with objects inside the Windows operating system. It is designed to work with both .NET language objects as well as COM Components with consistent syntax and naming conventions. Powershell's ability to work with objects makes it particularly well suited to interacting with the AWR Design Environment Component object model. More information about PowerShell can be obtained from the PowerShell Documentation and PowerShell Scripting pages on the Microsoft Web site.

Once you've downloaded and installed PowerShell you can get started right away working with the AWR Design Environment. To create and instance of AWR DE enter the following at the command line:

Code Snippets

```
PS C:\> $MWOffice = new-object -comobject "AWR.MWOffice"
```

This line tells powershell to create a new object of type comobject with the Program ID (ProgID) of "AWR.MWOffice" which is the Program ID for the latest version of MWOffice installed on your system. If you have more than one version and would like create an instance of a particular version you can do so with the following commands.

```
PS C:\> $MWOffice = new-object -comobject AWR.MWOffice.14.0
```

will create an instance of version 14.0.

Now that we have a reference to a running instance of AWR DE in the variable $MWOffice we can use it to see what members this object has. Enter the command:

```
PS C:\> $MWOffice | get-member
```

This prints a table formatted list of all the methods and properties. In addition, for each property it lists in curly braces {get} and possibly {set} depending on whether the property is read-only or read/write.

Next lets add a new schematic to the project:

```
PS C:\> $Schematic = $MWOffice.Project.Schematics.Add("MySchematic")
```

This uses the $MWOffice variable to access the schematics collection of the current project and add a schematic names "MySchematic" to that collection. The return value is a reference to the schematic so we capture that in the variable $Schematic. Now we can list the schematics in the project just by evaluating the value of the schematics collection off the project.

```
PS C:\> $MWOffice.Project.Schematics

Name                   : MySchematic

Equations              : \{\}

Elements               : \{MLIN.TL1\}

Wires                  : \{\}

Layout                 : System.__ComObject

LockUpdates            : False

UseProjectFrequencies  : True

Frequencies            : \{System._ComObject,
                         System._ComObject\}

UseProjectOptions      : True

(...)

ProcessDefinition      : Default

SelectedElements       : \{\}

SelectedWires          : \{\}
```

PowerShell enumerates each schematic in the collection and prints a summary of the properties of schematic object. From the name property we can see that we have successfully added the "MySchematic" schematic to the project.

Next let's add an MLIN element to the schematic using our schematic reference variable:

```
PS C:\> $Element = $Schematic.Elements.Add("MLIN", 0, 0)
```

Here we add an MLIN element to the Elements collection of the schematic. Now we can take a look at the elements currently in the schematic by evaluating the Elements collection:

```
PS C:\> $Schematic.Elements

Name                   : MLIN.TL1

Symbol                 : TLINE@SYSTEM.SYF

x                      : 0

y                      : 0

RotationAngle          : 0

Flipped                : False

Parameters             : \{ID, W, L, MSUB\}

Nodes                  : \{System._ComObject,
                         System._ComObject\}

Enabled                : True

Selected               : False

PartNumber             :

Left                   : 0

Top                    : -550

Width                  : 1001

Height                 : 650

ParameterFrame         : System.__ComObject

CellName               : MLIN*

Properties             : \{PartNumber, PartHelpPath,
                         XmlPath\}

Options                : \{SpiceModelLevel,
                         ExtractModelType\}

UseDefaultModelOptions : True

DrawingObject          : System.__ComObject

NameVisible            : True

DrawingObjects         : \{System.__ComObject\}

VectorInstanceName     :

UniqueID               : 1
```

Again PowerShell enumerates all the elements and for each prints a summary of the element properties. From the name property we can see that we've added MLIN.TL1 to the schematic elements and that it is using a symbol of TLINE@SYSTEM.SYF.

From here we can easily look at the parameters for the element using:

```
PS C:\> $Element.Parameters
```

This enumerates the element parameters and prints a summary for each. However this type of output is fairly verbose so lets just look at the parameter name and value as string.

```
PS C:\> $Element.Parameters | foreach-object { $_.Name + " = "+ $_.ValueAsString }
ID = TL1
W = 40
L = 100
MSUB =
```

This gives us a cleaner summary. Now lets set the width to 40:

```
PS C:\> $Element.Parameters.Item("W").ValueAsString = "20"
PS C:\> $Element.Parameters | foreach-object { $_.Name + " = "+ $_.ValueAsString }
ID = TL1
W = 20
L = 100
MSUB =
```

So in the first line we've set the parameter value as string to 40 and the second line lists the parameter names and values again to show the updated value.

PowerShell's ability to interact with COM components makes it particularly well suited as a command shell for interacting with the AWR Design Environment API. PowerShell provides a powerful interactive command-line interface for working with AWR DE as well as other applications within the Windows Environment.