Windows PowerShell Desired State Configuration Overview

Windows PowerShell Desired State Configuration (DSC) is a new management system in Windows PowerShell that enables the deployment and management of configuration data for software services and the environment on which these services run. To use DSC, first create a **configuration script** as shown below. Note that **Configuration** is a new keyword, which is part of the Windows PowerShell extensions for DSC. Each Configuration can have one or more **Node** blocks. Each Node block can have one or more resource blocks. You can use the same resource more than once in the same Node block, if you wish.

```
Configuration MyWebConfig
{
   # Parameters are optional
   param ($MachineName, $WebsiteFilePath)

   # A Configuration block can have one or more Node blocks
   Node $MachineName
   {
      # Next, specify one or more resource blocks
      # WindowsFeature is one of the resources you can use in a Node block
      # This example ensures the Web Server (IIS) role is installed
      WindowsFeature IIS
      {
          Ensure    = "Present" # To uninstall the role, set Ensure to "Absent"
          Name      = "Web-Server" # Name property from Get-WindowsFeature
      }

      # You can use the File resource to manage files and folders
      # "WebDirectory" is the name you want to use to refer to this instance
      File WebDirectory
      {
         Ensure          = "Present"  # You can also set Ensure to "Absent“
         Type            = "Directory“ # Default is “File”
         Recurse         = $true
         SourcePath      = $WebsiteFilePath
         DestinationPath = "C:\inetpub\wwwroot"
         DependsOn       = "[WindowsFeature]IIS"  # Use for dependencies
      }
   }
}
```

To create a configuration, invoke the Configuration block the same way you would invoke a Windows PowerShell function, passing in any expected parameters you may have defined (two in the example above). For example, in this case:

```
MyWebConfig -MachineName "TestMachine" -WebsiteFilePath "\\filesrv\WebFiles" `
         -OutputPath "C:\Windows\system32\temp" # OutputPath is optional
```

This creates a MOF file known as the **configuration instance document** at the path you specify. You can run it using the Start-DscConfiguration cmdlet (more on that cmdlet on the flipside of this sheet).

Archive Resource Example

The Archive resource gives you a mechanism to unpack archive (.zip) files at a specific path.

```
Archive ArchiveExample {
    Ensure       = "Present"  # You can also set Ensure to "Absent"
    Path         = "C:\Users\Public\Documents\Test.zip"
    Destination  = "C:\Users\Public\Documents\ExtractionPath"
}
```

Script Resource Example

The Script resource gives you a mechanism to run Windows PowerShell script blocks on target nodes. The TestScript block runs first. If it returns False, the SetScript block will run. The GetScript block will run when you invoke the Get-DscConfiguration cmdlet (more on that cmdlet on the flipside of this sheet). GetScript must return a hash table.

```
Script ScriptExample
{
    SetScript = {
        $sw = New-Object System.IO.StreamWriter("C:\TempFolder\TestFile.txt")
        $sw.WriteLine("Some sample string")
        $sw.Close()
    }
    TestScript = { Test-Path "C:\TempFolder\TestFile.txt" }
    GetScript  = { <# This must return a hash table #> }
}
```

Registry Resource Example

The Registry resource gives you a mechanism to manage registry keys and values.

```
Registry RegistryExample
{
    Ensure      = "Present"  # You can also set Ensure to "Absent"
    Key         = "HKEY_LOCAL_MACHINE\SOFTWARE\ExampleKey"
    ValueName   ="TestValue"
    ValueData   ="TestData"
}
```

Package Resource Example

The Package resource gives you a mechanism to install and manage packages, such as MSI and setup.exe packages, on a target node.

```
Package PackageExample
{
    Ensure      = "Present"  # You can also set Ensure to "Absent"
    Path        = "$Env:SystemDrive\TestFolder\TestProject.msi"
    Name        = "TestPackage"
    ProductId   = "663A8209-89E0-4C48-898B-53D73CA2C14B"
}
```

Environment Resource Example

The Environment resource gives you a mechanism to manage system environment variables.

```
Environment EnvironmentExample
{
    Ensure    = "Present"  # You can also set Ensure to "Absent"
    Name      = "TestEnvironmentVariable"
    Value     = "TestValue"
}
```

Group Resource Example

The Group resource gives you a mechanism to manage local groups on the target node.

```
Group GroupExample
{
    # This will remove TestGroup, if present
    # To create a new group, set Ensure to "Present“
    Ensure      = "Absent"
    GroupName   = "TestGroup"
}
```

User Resource Example

The User resource gives you a mechanism to manage local user accounts on the target node.

```
User UserExample
{
   Ensure     = "Present" # To delete a user account, set Ensure to "Absent"
   UserName   = "SomeName"
   Password   = $passwordCred # This needs to be a credential object
   DependsOn  = “[Group]GroupExample" # Configures GroupExample first
}
```

Service Resource Example

The Service resource gives you a mechanism to manage services on the target node.

```
Service ServiceExample
{
    Name         = "TermService"
    StartupType  = "Manual"
}
```

Desired State Configuration Cmdlets

After you create a configuration as described in the Overview section on the flipside of this sheet, you need to enact (apply) it using the Start-DscConfiguration cmdlet. Use the following command to parse the configuration at the specified path, send each node its corresponding configuration, and enact those configurations. This cmdlet will return a Windows PowerShell Job object which can be useful for configurations that are long-running.

```
Start-DscConfiguration -Path "C:\MyFolder" # Generated MOF file location
```

To send a configuration to a specific node and enact that configuration:

```
Start-DscConfiguration -ComputerName "TestMachine" -Path "C:\MyFolder"
```

To make Start-DscConfiguration interactive, use the Wait parameter:

```
Start-DscConfiguration –Verbose -Wait -Path "C:\MyFolder"
```

To get the current configuration:

```
Get-DscConfiguration -CimSession $session
```

To restore the previous configuration:

```
Restore-DscConfiguration -CimSession $session
```

Suppose you want to compare the current and actual configurations. This cmdlet returns True if the current and actual configurations match exactly and False otherwise:

```
Test-DscConfiguration -CimSession $session
```

Advanced Resource Properties

To see all the properties for a given resource, as well as the types of these properties, set the cursor on the resource keyword and press Ctrl + Spacebar. (The resource keywords are Registry, Script, Archive, File, WindowsFeature, Package, Environment, Group, User, Log, Service, and WindowsProcess.) All resources have a property called **DependsOn** that you can use to indicate when a given resource should be configured before another. See the User resource example for how to use it.

Configuration Data

This is an example of separating the node data from configuration logic. You can add more node hash tables to the AllNodes array.

```
$ExampleConfigData = @{
    AllNodes = @(
          # NodeName "*" applies globally to all nodes in this array
          @{ NodeName = "*"; RecurseValue = $true },
          @{ NodeName = "Server101"; Role = "Web"; RolesToBePresent = "Web-Server";
             SourceRoot = "\\Server106\source\presentation\"; Version = "1.0";
             WebDirectory = "c:\inetpub\wwwroot\"; RecurseValue = $false; }
    );
}
Configuration CloudService
{   # The $AllNodes and $Node (current node) variables are automatic variables
    Node $AllNodes.Where("Role -eq Web").NodeName {
        WindowsFeature IIS
        { Ensure = "Present"; Name = $Node.RolesToBePresent }
    }
}
CloudService –ConfigurationData $ExampleConfigData
```

Local Configuration Manager

Local Configuration Manager is the DSC engine. It runs on all nodes and is responsible for calling the resources in the configuration script. You can modify the Local Configuration Manager settings of a target node by including a "LocalConfigurationManager" block inside the Node block.

```
LocalConfigurationManager
{
    RebootNodeIfNeeded   =  $true # Automatically reboots if required by config
    ConfigurationMode    = “ApplyAndAutoCorrect" # Corrects configuration drift
}
```

Set the cursor on the LocalConfigurationManager keyword and press Ctrl + Spacebar to see all the properties you can set and their types. Only one Local Configuration Manager settings block can exist per Node block. When you invoke a configuration that includes a Local Configuration Manager settings block, this will create a separate MOF file for the Local Configuration Manager settings. You can then enact these settings using the following cmdlet:

```
Set-DscLocalConfigurationManager -Path "C:\MyFolder" # Generated MOF file location
```

To set Local Configuration Manager settings using the MOF file for a specific node:

```
Set-DscLocalConfigurationManager -ComputerName "MyNode" –Path "C:\MyFolder"
```

To get the Local Configuration Manager settings:

```
Get-DscLocalConfigurationManager -CimSession $session
```