ScriptingAnswers.com

Windows PowerShell Advanced

LAB GUIDE

Lab 1-1
Working with Databases

Task #1

Here's your task:
Create a database with a table named SPInventory. In it, create a text column named ComputerName, and another named SPVersion. Populate at least two rows of data by filling in the ComputerName column (use `localhost` and your computer's name).

Write a PowerShell script that queries all of the computers from the ComputerName column, connects to each computer via WMI, and retrieves the computer's service pack major version number. Store that version number in the database's SPVersion column.

Connection Strings:

Access 2007
```
Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\myFolder\myAccess2007file.accdb;Persist Security Info=False;
```

Access 2000-2003
```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\mydatabase.mdb;
```

Excel 2000-2003 (requires that Access be installed)
```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\MyExcel.xls;Extended
Properties="Excel 8.0;HDR=Yes;IMEX=1";
```

Excel 2007 (requires that Access be installed)
```
Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=c:\myFolder\myExcel2007file.xlsx;Extended Properties="Excel
12.0;HDR=YES";
```

Microsoft SQL Server 7-2000 (using SqlConnection)
(standard security)
```
Data Source=myServerAddress;Initial Catalog=myDataBase;User
Id=myUsername;Password=myPassword;
```

(integrated security)
```
Data Source=myServerAddress;Initial Catalog=myDataBase;Integrated Security=SSPI;
```

Microsoft SQL Server 2005 (using SqlConnection)
(standard security)
```
Data Source=myServerAddress;Initial Catalog=myDataBase;User
Id=myUsername;Password=myPassword;
```

(integrated security)
```
Data Source=myServerAddress;Initial Catalog=myDataBase;Integrated Security=SSPI;
```

Lab 2-1
Custom Format Views

Task #1

Here's your task:
Create a table layout for the Win32_OperatingSystem class, so that it displays the CSName, ServicePackMajorVersion, ServicePackMinorVersion, and BuildNumber properties. The column names should be Computer, SPMajor, SPMinor, and Build.

Name the view VersionView. Configure PowerShell so that your new view is used by default when displaying instances of Win32_OperatingSystem.

Write a one-liner that uses **Get-Content** to retrieve computer names from a file (one computer name per line – use `localhost` and your computer's name for testing), retrieves Win32_OperatingSystem from each computer, and displays the results using your new view. Do not use a Format-* cmdlet in your one-liner.

Here's a tip:
Remember that the order in which format data is loaded determines which view is "seen first" by PowerShell – and the view it "sees" first is the one it will use by default.

Use the existing DotNetTypes.format.ps1xml (in PowerShell's installation folder) as an example, if you need one.

If you modify your ps1xmlfile, you'll need to close PowerShell, re-open it, and re-load the file to see your changes.

Lab 3-1
Custom Type Extensions

Task #1

Here’s your task:
Extend the [string] type to have a ScriptMethod called Reachable. This method should attempt to ping the contents of the string (assuming it is an IP address or hostname) and return either $True or $False.

Here’s a tip:
Remember that the ScriptMethod code can use the variable $this to access the contents of the object.

Task #2

Here’s your task:
Extend the [string] type to have a ScriptProperty called IsUNC. This property should be read only (e.g., have only a Get block), and should return $True if the contents of the string are formatted as a UNC.

Here’s a tip:
A regular expression to check for UNCs is `^\\\\\w*\\\w*`

If you modify your ps1xmlfile, you’ll need to close PowerShell, re-open it, and re-load the file to see your changes.

Lab 4-1
Working with Windows Forms

Task #1

Here's your task:
Create a PowerShell script that displays a dialog box which accepts a computer name.

When the user types a computer name and clicks OK, a second form should be displayed that lists all of the services running on that computer in a list box.

An OK button will close the second form, while a Stop button will stop the selected service.

The Stop button should have no effect if no service name is selected in the list box.

Here's a tip:
You can use a System.Windows.Forms.ListBox control on the second form. To add an item to a list box (assuming the list box control is in variable $listbox), do this: $listbox.items.add("*text_to_add*").

To access the text of the selected list box item, use $listbox.SelectedItem.

To see if anything is selected in the listbox, check $listbox.SelectedItemIndex. If that property's value is -1, then nothing is selected.

Use WMI to retrieve the services. The second form will have to retrieve just a specific service, based on the service name selected, and execute its StopService() method.

Lab 5-1
Practicing Advanced Techniques

Task #1

Here's your task:
Write a function named Ping-Port which accepts an array of IP addresses, a lower TCP port number, and an upper TCP port number.

For each IP address, display the TCP ports within the specified range which are open. For IP addresses which are not reachable (pingable), display a "not reachable" message.

Sample output should look like this:

```
192.168.4.1 : Not reachable
192.188.4.2 : Reachable
  Port 20 open
  Port 52 open
```

Display a progress bar while the function is running.

Here's a tip:
Simplify your output code by using the –f operator to replace tokens, rather than concatenating strings.

Here's a tip:
You'll get an exception if you try to connect to a port and the remote computer actively refuses the connection. You can write a trap handler which simply issues a Continue command – this will handle the exception and allow your script to continue. Proper placement of the trap handler, however, is crucial to it having the desired effect!

Here's a tip:
Not getting the results you want? Use PowerShell's interactive debugger to follow your script. Suspend the script frequently to check variable values to see if they're what you expected.

If you run out of time to complete this lab, you may have additional time to work on it after the final lab of the day.

Lab 6-1
Advanced PowerShell Practical Lab

Task #1

Here's your task:
Create a text file that contains multiple computer names, with one computer name per line. This will be your project's *input file.*

Create a database that contains a table named DriveInventory. In the table, create a text column named ComputerName, a text column named DriveLetter, a text column named AvailableMB, a text column named TotalMB, and a date/time column named LastUpdated.

Read the computer names from an input text file containing one computer name per line (use **localhost** and your computer's name, for testing purposes). For each computer name found, execute a function named Get-DriveInventory. This function should use WMI to retrieve the amount of free space, in megabytes, on each *local hard disk* for the specified computer(s). The function should output a custom object which has the following properties:

ComputerName
DriveLetter
TotalSpace (This should be in MB, not bytes)
AvailableSpace (This should be in MB, not bytes)

Write a second function named Write-DriveInventory that accepts these custom objects and writes their information to the database. Be sure to populate all four columns of the Access table.

The "main line" of your script should look something like this:

```
Get-Content <filename> | Get-DriveInventory | Write-DriveInventory
```

Start here:
Create your input text file, and make sure you can display it using **Get-Content**.

Then, do this:
Write the Get-DriveInventory function. It should be a filtering function, accepting pipeline input and using a PROCESS scriptblock. It should output custom objects using **New-Object** and **Add-Member**. *Refer to the "Intermediate Windows PowerShell" course, or "Windows PowerShell: TFM," for details.*

Next, do this:
Write the **Write-DriveInventory** function. Again, this is a filtering function that uses a PROCESS scriptblock. You might consider using a BEGIN and END scriptblock to open and

Task #1

close the database connection, so that you're only doing so once.

Task #2

Here's your task:
Starting with your solution to Task #1, modify your script to save the drive inventory in a CSV format, rather than in a database. You should need to change only *one thing* in your code to accomplish this.

Here's a tip:
Remove **Write-DriveInventory** from your "main one-liner." What could you replace it with that would convert objects into a CSV file?

Task #3

Here's your task:
Starting with your solution to Task #2, modify your one-liner to simply *display* the results of Get-DriveInventory in a table. Have the table include a new column displaying the percentage of free space.

Here's a tip:
The **Format-Table** cmdlet can do this for you – review module 5 for a tip on creating custom calculated columns in a table.

SOLUTIONS

Lab 1-1 Solutions

Task #1

Here's the solution:

```
function TestPing([string]$address) {
   $wmi = Get-WmiObject -query "SELECT * FROM Win32_PingStatus `
    WHERE Address = '$address'"
   if ($wmi.statuscode -eq 0) {
     $true
   } else {
     $false
   }
}

[reflection.assembly]::loadwithpartialname("System.Data") | Out-Null

#make connection
#need one to read with, and one to make changes with
$readconn = New-Object System.Data.OleDb.OleDbConnection
$writeconn = New-Object System.Data.OleDb.OleDbConnection

#will need to update path to access db file
[string]$connstr = "Provider=Microsoft.ACE.OLEDB.12.0;Data
Source=C:\SPInventory.accdb;Persist Security Info=False;"
$readconn.connectionstring = $connstr
$writeconn.connectionstring = $connstr
$readconn.open()
$writeconn.open()

#create SQL query
$sql = "SELECT ComputerName from SPInventory"

#create read command
$readcmd = New-Object system.Data.OleDb.OleDbCommand
$readcmd.connection = $readconn
$readcmd.commandtext = $sql

#execute query
$reader = $readcmd.executereader()

#loop through data
while ($reader.read()) {
   $computer = $reader.getvalue($reader.getordinal("ComputerName"))
   if (TestPing $computer) {

     #query WMI
     $wmi = gwmi win32_operatingsystem -comp $computer
     foreach ($result in $wmi) {
       $spver = $result.servicepackmajorversion
     }
   } else {

     #computer not pingable
     $spver = "Unreachable"
   }
   #update database
   $sql = "UPDATE SPInventory SET SPVersion = '$spver' WHERE `
    ComputerName = '$computer'"
   $writecmd = New-Object system.Data.OleDb.OleDbCommand
   $writecmd.connection = $writeconn
   $writecmd.commandtext = $sql
   $writecmd.executenonquery() | out-null
}

#close reader
$reader.close()

#close connection
$readconn.close()
$writeconn.close()
```

Lab 2-1 Solutions

```
<Configuration>
    <ViewDefinitions>
        <View>
            <Name>VersionView</Name>
            <ViewSelectedBy>
    <TypeName>System.Management.ManagementObject#root\cimv2\Win32_OperatingSystem</TypeName>
            </ViewSelectedBy>
            <TableControl>
                <TableHeaders>
                    <TableColumnHeader>
                        <Label>Computer</Label>
                    </TableColumnHeader>
                    <TableColumnHeader>
                        <Label>SPMajor</Label>
                    </TableColumnHeader>
                    <TableColumnHeader>
                        <Label>SPMinor</Label>
                    </TableColumnHeader>
                    <TableColumnHeader>
                        <Label>Build</Label>
                    </TableColumnHeader>
                 </TableHeaders>
                <TableRowEntries>
                    <TableRowEntry>
                        <TableColumnItems>
                            <TableColumnItem>
                                <PropertyName>CSName</PropertyName>
                            </TableColumnItem>
                            <TableColumnItem>
                                <PropertyName>ServicePackMajorVersion</PropertyName>
                            </TableColumnItem>
                            <TableColumnItem>
                                <PropertyName>ServicePackMinorVersion</PropertyName>
                            </TableColumnItem>
                            <TableColumnItem>
                                <PropertyName>BuildNumber</PropertyName>
                            </TableColumnItem>
                        </TableColumnItems>
                    </TableRowEntry>
                 </TableRowEntries>
            </TableControl>
        </View>
    </ViewDefinitions>
</Configuration>
```

To use (adjust file paths as needed):

```
Update-FormatData -pre 'ILT-PSH-301 Lab 3-1.format.ps1xml'
gwmi win32_operatingsystem
gc c:\computers.txt | % { gwmi win32_operatingsystem -comp $_ }
```

Lab 3-1 Solutions

```
<Types>
    <Type>
        <Name>System.String</Name>
        <Members>
            <ScriptMethod>
                <Name>Responding</Name>
                <Script>
                    $wmi = gwmi -qu "SELECT StatusCode FROM Win32_PingStatus WHERE
Address='$this'"
                    if ($wmi.statuscode -eq 0) { $True } else { $False }
                </Script>
            </ScriptMethod>
            <ScriptProperty>
                <Name>IsUNC</Name>
                <GetScriptBlock>
                                    $this -match "^\\\\\w*\\\w*"
                </GetScriptBlock>
            </ScriptProperty>
               </Members>
    </Type>
</Types>
```

To use (adjust file paths as needed):

```
# You will need to update the path to match
Update-TypeData -pre 'ILT-PSH-301 Lab 4-1.ps1xml'
#Task 1
"string" | gm
"localhost".responding()
"unreachable".responding()
#Task 2
"\\server\share".IsUNC
"Not a UNC".IsUNC
```

Lab 4-1 Solutions

Task #1

Here's the solution:

```
[reflection.assembly]::loadwithpartialname("System.Windows.Forms") | Out-Null

# create form 1
$form1 = New-Object system.Windows.Forms.Form
$form1.text = "Computer name"
$form1.height = 120
$form1.width = 250
$form1.formborderstyle = 3

# create OK and cancel scriptblocks
$oksb = {
        $form1.dialogresult = 1
        $form1.hide()
}
$cancelsb = {
        $form1.dialogresult = 0
        $form1.hide()
}

# create OK button
$okbutton1 = New-Object system.Windows.Forms.Button
$okbutton1.text = "OK"
$okbutton1.height=25
$okbutton1.width=75
$okbutton1.top=51
$okbutton1.left=147
$okbutton1.add_click($oksb)
$form1.controls.add($okbutton1)

# create Cancel button
$cancelbutton1 = New-Object system.Windows.Forms.Button
$cancelbutton1.text = "Cancel"
$cancelbutton1.height=25
$cancelbutton1.width=75
$cancelbutton1.top=51
$cancelbutton1.left=66
$cancelbutton1.add_click($cancelsb)
$form1.controls.add($cancelbutton1)

# create label
$label1 = New-Object system.Windows.Forms.Label
$label1.text = "Enter computer name:"
$label1.left=12
$label1.top=9
$label1.width=205
$label1.height=13
$form1.controls.add($label1)

# create text box
$text1 = New-Object system.Windows.Forms.TextBox
$text1.left=15
$text1.top=25
$text1.height=20
$text1.width=205
$form1.controls.add($text1)
```

```
# show form 1
if ($form1.showdialog() -eq 0) {
        # cancelled
        break
}

# get services
$computer = $text1.text
$services = gwmi win32_service -comp $computer -filter "state='Running'"

# create form 2
$form2 = New-Object system.Windows.Forms.Form
$form2.text = "Computer name"
$form2.height = 256
$form2.width = 282
$form2.formborderstyle = 3

# create close/stop scriptblocks
$closesb = {
        $form2.hide()
}
$stopsb = {
        if ($list2.selectedindex -gt -1) {
                $service = $list2.selecteditem
                $filter = "name='$service'"
                $wmi = gwmi win32_service -comp $computer -filter $filter
                foreach ($result in $wmi) {
                        $return = $result.stopservice()
                }
        }
}

# create Stop button
$stopbutton2 = New-Object system.Windows.Forms.Button
$stopbutton2.text = "Stop"
$stopbutton2.height=25
$stopbutton2.width=75
$stopbutton2.top=156
$stopbutton2.left=179
$stopbutton2.add_click($stopsb)
$form2.controls.add($stopbutton2)

# create Close button
$closebutton2 = New-Object system.Windows.Forms.Button
$closebutton2.text = "Close"
$closebutton2.height=25
$closebutton2.width=75
$closebutton2.top=185
$closebutton2.left=179
$closebutton2.add_click($closesb)
$form2.controls.add($closebutton2)

# create label
$label2 = New-Object system.Windows.Forms.Label
$label2.text = "Services:"
$label2.left=12
$label2.top=9
$label2.width=205
$label2.height=13
$form2.controls.add($label2)

# create list box
$list2 = New-Object system.Windows.Forms.ListBox
$list2.top = 25
$list2.left = 15
$list2.height = 186
$list2.width = 159
```

```
# populate list box
foreach ($service in $services) {
        $list2.items.add($service.name) | Out-Null
}

# add list box to form
$form2.controls.add($list2)

# show form 2
$form2.showdialog() | Out-Null
```

Lab 5-1 Solutions

Task #1

Here's the solution:

```
Function Ping-Port {
  param (
    [string]$addresses = @("localhost"),
    [int]$startport = 1,
    [int]$endport = 100
  )

  # set up output templates
  $ipstatus = "{0} : {1}"
  $portstatus = "  Port {0} Open"

  # create ping object
  $ping = New-Object System.Net.NetworkInformation.Ping

  # go through addresses
  foreach ($address in $addresses) {

    # progress bar
    Write-Progress -activity "Checking $address" -status "Pinging..." `
      -current "Wait..." -percent 0 -id 1

    # reachable?
    if ($ping.send($address)) {
      # reachable: yes
      $ipstatus -f $address,"Reachable"

      # go through ports
      for ($port=$startport; $port -lt $endport; $port++) {

        # trap handler for refused exceptions
        trap {
                continue
        }

        # calculate percentage
        $numports = $endport-$startport
        $percent = $port / $numports * 100 -as [int]

        # progress bar
        Write-Progress -activity "Checking $address" -status "Probing..." `
           -current "Port $port" -percent $percent -id 1

        # attempt to create port
        $socket = New-Object system.Net.Sockets.TcpClient($address,$port)
        if ($socket -ne $null) {

                # write port open message
                $portstatus -f $port
        }
      }

    } else {
      # reachable: no
      $ipstatus -f $address,"Not reachable"
    }
  }

}

Ping-Port "mediaserver" 1 1024
```

Lab 6-1 Solutions

Tasks #1 and #2

Here's the solution:

```
[reflection.assembly]::loadwithpartialname("System.Data") | Out-Null

function Get-DriveInventory {
   PROCESS {
     #get drives from WMI
     $drives = gwmi win32_logicaldisk -comp $_ -filter "drivetype=3"

     #construct output objects
     foreach ($drive in $drives) {
       $obj = New-Object psobject
       $obj | Add-Member NoteProperty ComputerName $_
       $obj | Add-Member NoteProperty DriveLetter $drive.deviceid
       $free = $drive.freespace/1MB -as [int]
       $obj | Add-Member NoteProperty AvailableSpace $free
       $total = $drive.size/1MB -as [int]
       $obj | Add-Member NoteProperty TotalSpace $total

       #write output object
       Write-Output $obj
     }
   }
}

function Write-DriveInventory {
   BEGIN {
     #open database connection
     $conn = New-Object System.Data.OleDb.OleDbConnection
     #will need to adjust the path
     $connstr = "Provider=Microsoft.ACE.OLEDB.12.0;Data `
       Source=C:\DriveInventory.accdb;Persist Security Info=False;"
     $conn.connectionstring = $connstr
     $conn.open()

     #create database command
     $cmd = New-Object system.Data.OleDb.OleDbCommand
     $cmd.connection = $conn
   }
   PROCESS {
     #construct SQL statement using input objects' properties
     $now = Get-Date -form g
     $sql = "INSERT INTO DriveInventory `
       (ComputerName,DriveLetter,AvailableMB,TotalMB,LastUpdated) VALUES ("
     $sql += "'" + $_.ComputerName + "',"
     $sql += "'" + $_.DriveLetter + "',"
     $sql += "'" + $_.AvailableSpace + "',"
     $sql += "'" + $_.TotalSpace + "',"
     $sql += "'" + $now + "')"
     $cmd.commandtext = $sql
     $cmd.executenonquery() | Out-Null
   }
   END {
     $conn.close()
   }
}

function Get-FileName {
   $computer = Read-Host "Computer?"
   return $computer
}
```

```
# get the filename
$f = Get-FileName

# Use this command-line for task #1
Get-Content $f | Get-DriveInventory | Write-DriveInventory

# Use this command-line for task #2
Get-Content $f | Get-DriveInventory | Export-Csv C:\DriveInventory.csv

# Use this command-line for task #3
Get-Content $f | Get-DriveInventory | `
 Format-Table ComputerName,DriveLetter,AvailableSpace,TotalSpace,`
 @{Label="PercentFree";Expression={$_.AvailableSpace/$_.TotalSpace*100}}
```